Escuela Superior de Ingenieros - Universidad de Sevilla
Departamento de Ingeniería de Sistemas y Automática
Área de Ingeniería Telemática

Proyecto Fin de Carrera

Estado del Arte en la Gestión de Contenidos.

Comparativa con Lenya.

Autor: Rafael Pérez Luna

Director: Antonio J. Estepa Alonso

“a mis recuerdos les pregunté por ti.
aún discuten”

Sevilla, Junio de 2006

agradecimientos

Llega el final y no habría podido conseguirlo sin vosotros.

Papá, mamá, Dani, este proyecto es vuestro, yo sólo soy el emisario. Nunca un ser humano creará las palabras necesarias para agradeceros todo lo que os debo.

Gracias a Javi, mi hermano encontrado en Sevilla. Las cosas se ven distintas cuando se observan desde múltiples lugares.

A todos mis amigos egabrenses. María, Álvaro, Javi, Andrés, Jesús, Grego, parte de este proyecto también es vuestro por escucharme cuando lo necesitaba y acompañarme en el camino.

A mi gente de Sevilla. Ángel, Vigui, Antonio, Leo, compañeros de fatigas.

Por último, quiero agradecer la ayuda prestada por mi director de proyecto, Antonio Estepa.
Su orientación y trabajo han sido definitivos para llegar aquí.

Índice de contenido

Parte I: Introducción

Parte II: Aportaciones

Apéndices

Índice de ilustraciones

parte I: introducción

capítulo 1: introducción y objetivos

1.1.Introducción.

Desde la aparición de internet, el uso que se ha hecho de él por parte de las organizaciones ha ido en aumento. El abaratamiento de los costes de conexión y de los equipos necesarios para esta tarea, el incremento de la velocidad de datos, el aumento de la cobertura de las operadoras y el incremento del uso por parte de posibles clientes de esta tecnología han hecho que las organizaciones se planteen cada vez más como una necesidad estratégica el contar con un portal web que presente información sobre ellas o que pueda ser utilizado para la interacción con el usuario.

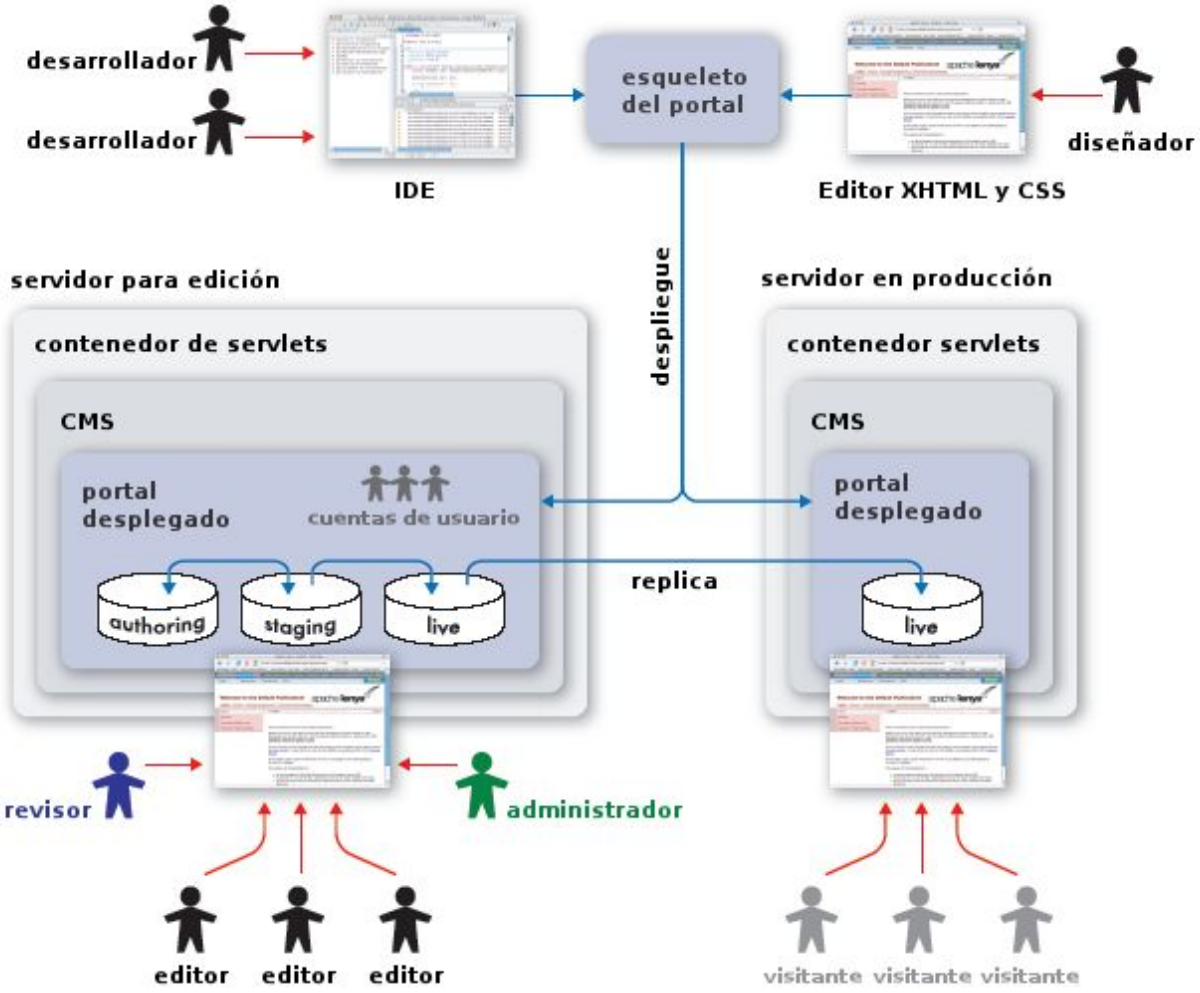

Figura 1: Esquema de colaboración en la Gestión de Contenidos.

Para las organizaciones que tienen un gran tamaño existe también la necesidad de gestionar una cantidad ingente de información, tarea que puede resultar muy complicada. Imaginemos una organización con sedes en 7 países y que tiene un portal web accesible en 7 idiomas distintos. Si esta organización necesita reflejar cambios en los contenidos de su portal web en estos 7 idiomas, y necesita hacerlo de una forma rápida puede verse en serios problemas si no dispone de una herramienta eficaz que le permita realizar este trabajo.

Hay otros aspectos que presentan problemas cuando una organización se enfrenta a la gestión de la información y que deben ser solucionados. Desorganización del trabajo, aumento de la cantidad de información, acceso a la misma, etc. son problemas a los que se han de enfrentar constantemente los desarrolladores de aplicaciones web.

Con el auge de las comunicaciones e internet, la gestión de contenidos se ha convertido en una necesidad no sólo como una facilidad para realizar cambios en los portales corporativos sino como una ventaja estratégica de las empresas. Este mismo incremento en el uso de las nuevas tecnologías ha aumentado la necesidad de información de las personas. Para una empresa, un portal web no permite sólo realizar una presentación de la misma a los visitantes del portal. Es una fuente de información que puede hacer que algún cliente o usuario se decida por ella en particular para realizar una tarea determinada. En este escenario, la claridad de los contenidos expuestos, la estructura de los mismos, la funcionalidad de la página, la facilidad de uso, etc. son puntos fundamentales para el éxito de la organización. Todos estos aspectos incrementan la complejidad del portal y el tiempo de puesta a punto e inserción de contenidos en un entorno clásico de desarrollo del mismo, es decir, impiden que la información que el usuario necesita pueda llegar a la velocidad que requiere.

A lo largo del capítulo trataremos la motivación de los Sistemas de Gestión de Contenidos y los objetivos del presente Proyecto Fin de Carrera. Como último punto del mismo, realizaremos algunos comentarios sobre la organización de la memoria.

1.2. Motivación.

Hay varios aspectos que han provocado la necesidad de la aparición de las herramientas para la gestión de los contenidos. La falta de organización, el aumento de la información o la necesidad de personal cualificado son sólo algunos de ellos. A lo largo de este punto intentaremos detallar un poco más estos problemas y justificar la necesidad de una herramienta que se encargue de mitigarlos.

El nuevo escenario en el que se mueven las organizaciones, con cantidades cada vez más grandes de información a gestionar y portales web más difíciles de mantener, ha provocado un cambio de mentalidad. Se hace necesaria la distribución de la carga de trabajo entre varias personas y la separación entre el contenido y la presentación de la información. El trabajo colaborativo también trae problemas asociados cuando varias personas intentan acceder al mismo recurso en el mismo momento. Se pueden provocar fallos debidos a la edición que deberían controlarse. En este modelo distribuido, se pueden crear varios roles que accedan, modifiquen o creen la información de distintas formas. Todo lo relatado hasta el momento sería irrealizable sin un sistema capaz de controlar que todo funciona correctamente. Éste sería un Sistema de Gestión de Contenidos.

Debido a las necesidades anteriormente citadas, han proliferado en el mercado varios Sistemas de Gestión de diferentes características que intentan ser la solución a los problemas de las organizaciones. Dentro de estos productos encontramos soluciones tanto propietarias como basadas en Software Libre. Debido a esta proliferación de CMS (*Content Management System*, en su nombre inglés), se hace necesario el análisis del estado del arte de la Gestión de Contenidos.

En los siguientes puntos intentaremos comentar con más detalle la problemática mencionada.

1.2.1.1.Aumento de la cantidad de información.

En los inicios de internet y el protocolo HTTP existía una figura única encargada de la gestión de todo los relacionado con las páginas web: el webmaster. Esta persona no sólo debía realizar todos los cambios que se proyectaran en las páginas (introducción de páginas nuevas, etc.) sino que tenía que encargarse también del diseño y presentación de la información contenida en las mismas. Además, si había algún problema, era la persona de contacto con la que se debía hablar para solucionarlo.

Como se comprenderá, conforme iba aumentando el número de páginas gestionadas y la cantidad de contenidos de las mismas, esta tarea se iba haciendo más y más complicada.

Imaginemos un escenario en el que se ha creado un portal de noticias. Este portal podía haber comenzado como un conjunto pequeño de páginas que hablaba sobre temas locales. Pero la afluencia de visitantes a las mismas hizo que además de estos temas se empezaran a tratar temas de ámbito nacional y posteriormente internacional. La persona que en un primer momento se bastaba para realizar los cambios necesarios (revisión de las noticias publicadas, creación de nuevas noticias, etc.) se vería desbordada en un tiempo muy corto por las nuevas necesidades del portal y necesitaría de ayuda para realizar todos los cambios. Esta ayuda debería tener conocimientos de HTML que le permitieran realizar este trabajo, lo que supondría contratar una mano de obra más o menos especializada en el tema.

También nos encontraríamos con el problema del aspecto general del portal. Cada una de estas personas puede tener una visión distinta de la presentación: colores, tamaño de letras, etc. Aunque se realizaran plantillas, siempre serían capaces de cambiar la imagen del portal sin tener en cuenta la presentación del mismo en conjunto, con lo que sería realmente complicado mantener una imagen conjuntada de todo lo que perteneciera al mismo.

1.2.1.2. Desorganización del trabajo y la información.

El incremento mismo de la cantidad de información que una empresa debe manejar conlleva problemas de desorganización del trabajo. Es muy normal que varias personas trabajen a la vez sobre los mismos contenidos de una web, un programa, un informe, etc. Cada una de estas personas está realizando tareas distintas que luego se deberán poner en común y puede que alguna de estas personas necesite del avance de uno de sus compañeros para proseguir su trabajo. En esta situación se hacen necesarios controles de bloqueo en el trabajo para que los datos o archivos que estén usando unos no puedan ser modificados por otras personas en ese momento. Si varias personas tocan a la vez el mismo archivo, las posibilidades de error crecen exponencialmente.

También podríamos encontrarnos con problemas si un empleado modifica un archivo que está publicado y lo hace visible sin que su supervisor lo haya comprobado. Si la modificación no era necesaria, o era errónea, nos encontraríamos con que una versión incorrecta del mismo ha estado disponible al público durante un tiempo indeterminado, contando con que haya sido descubierto el error.

1.2.1.3. Incremento de los costes.

Como hemos comentado al hablar de los problemas asociados al incremento de la cantidad de información, al haber más carga de trabajo se hace necesaria la contratación de personal más o menos experto para hacer frente a las necesidades de creación y edición así como de la gestión del portal. Como alternativa, también se podría contratar el mantenimiento a la misma empresa encargada de la realización del mismo, aunque esto conllevaría dos problemas: el primero y fundamental es que estamos atados a ellos para cualquier modificación o actualización del contenido del portal; el segundo es que estos cambios se facturan aparte.

Volviendo a la alternativa de la contratación del personal, éste debe tener conocimientos de HTML y programación, por lo que la empresa debe realizar un desembolso mayor que el que le sería necesario si sus mismos empleados pudieran añadir y gestionar los contenidos del portal con una herramienta simple y potente.

Tanto una alternativa como la otra implican un desembolso extra aparte del necesario para la creación y desarrollo del sitio web.

1.2.1.4.Control del Acceso.

En un portal web moderno se hace necesario llevar un control de las zonas accesibles o no accesibles por parte de los usuarios. Estos problemas se solucionan de forma más o menos simple guardando los usuarios en una base de datos y comprobando que cuando se solicita una página ésta es accesible por parte del mismo. También podría ser necesario añadir grupos de usuarios capaces de realizar determinadas tareas dentro del portal o capaces de ver ciertos contenidos a los que otros usuarios no podrían acceder.

La gestión de estos usuarios y grupos es una tarea ardua también. Los mismos desarrolladores del portal podrían dar de alta unos grupos y unos usuarios iniciales que serían más o menos estáticos. La modificación de estos aspectos y de las zonas accesibles sería bastante complicada para gente no especializada, por lo que, de nuevo, nos encontraríamos sujetos al personal que desarrolló el portal para realizar las modificaciones.

1.3.Objetivos.

El objetivo de este proyecto es el análisis del estado del arte de la Gestión de Contenidos, con especial atención a Apache Lenya, de la Apache Software Foundation. Así mismo, **se realizará una comparativa entre Sistemas de Gestión de Contenidos** con el objetivo de contar con un documento que nos permita realizar una elección sopesada entre los mismos. En base a esta comparativa se podrá realizar una elección en torno al CMS que más se ajuste a las necesidades de la empresa u organización interesada en su implantación.

El estudio se centrará en los CMS *OpenSource* debido a que son los más accesibles del mercado y no necesitan del pago de una licencia para su instalación y puesta en marcha. Además, los productos *OpenSource* están teniendo un gran auge y una gran aceptación entre las empresas, por lo que añade un motivo más para la elección de estas tecnologías como base de este Proyecto Fin de Carrera.

1.4.Organización de la memoria.

La memoria del presente Proyecto Fin de Carrera está estructurada en dos partes:

- Parte I: Introducción.
- Parte II: Aportaciones.

La primera parte está formada a su vez por dos capítulos, éste en el que nos hallamos y un capítulo dedicado a la Gestión de Contenidos. En él se intentarán dar ciertas definiciones sobre conceptos relacionados con la Gestión de Contenidos y se esbozará un esquema de la arquitectura de un Sistema de Gestión de Contenidos abstracto. Además se expondrán una serie de consideraciones a tener en cuenta en la decisión de la implantación de un Gestor de Contenidos en una empresa.

En la segunda parte se tratarán los aspectos tecnológicos y se resolverán la mayoría de los objetivos de este Proyecto Fin de Carrera. Para ello, en el Capítulo 3 se hará una pequeña introducción a las tecnologías utilizadas por Apache Lenya. En el Capítulo 4 se entrará en más profundidad en el estudio de la arquitectura y funcionalidades de Apache Lenya. Por último, en el Capítulo 5 se realizará una comparativa entre tres Sistemas de Gestión de Contenidos *OpenSource*: Apache Lenya, OpenCMS y Alfresco.

capítulo 2:
gestión de contenidos

2.1.Conceptos Previos.

Una de las principales prerrogativas a la hora de tratar con los gestores de contenidos es la de tratar de conseguir una separación completa entre el contenido y la presentación, tal como indicamos en la introducción a este capítulo. Para ello es conveniente contar con alguna definición sobre lo que, en vistas de un CMS, se puede considerar contenido. En esta sección trataremos de dar algunas definiciones para fijar los conceptos e intentaremos presentar algunas características de los contenidos, tal y como puede ser su estructura y formato. [CMSBIBLE2002]

2.1.1.Definición de Contenido, Datos, Información y Funcionalidad.

En este apartado vamos a intentar dar algunas definiciones acerca de los elementos objeto de gestión por parte de cualquier CMS. Estas definiciones intentarán aclarar conceptos e ideas sobre algunos términos que pueden dar lugar a confusión. Las definiciones aquí realizadas serán las que se tomarán en cuenta para cualquier discusión posterior en este capítulo.

Algunas veces, los términos contenido, información y datos pueden confundirse entre sí debido a las similitudes que tienen entre ellos, aunque desde el punto de vista de la Gestión de Contenidos sean elementos que se pueden diferenciar.

2.1.1.1.Datos.

Los datos son la representación abstracta del contenido mediante pequeñas piezas de información que han sido creadas para su tratamiento por ordenador. Esta abstracción nos permite su almacenamiento de forma más o menos estructurada y su procesado por medios automatizados.

Como ejemplo podemos poner una Base de Datos en la que se almacenan noticias asociadas a una determinada categoría. Las noticias en sí y las categorías se encontrarían en tablas distintas y estarían relacionadas por las claves de cada una de estas tablas. De esta forma, si quisiéramos acceder a las noticias que pertenecen a la categoría sucesos, deberíamos acceder a la tabla de las categorías para obtener el identificador numérico de la misma y a partir de este, buscar las entradas en la tabla de las noticias que casaran con él. Podemos ver pues, que estos datos no tienen ninguna información comprensible a simple vista, pero que permiten el acceso a la información y su almacenamiento de forma automatizada.

2.1.1.2.Contenido.

El contenido como ente abstracto, relaciona la información que se quiere mostrar a un usuario con la forma en la que se almacena como datos en un sistema informático. De esta forma, podemos asignar al contenido una serie de datos (metadatos o metainformación) que nos faciliten la labor de catalogación y acceso al mismo.

El contenido a gestionar puede estar dividido a su vez en trozos más pequeños llamados elementos de contenido. El conjunto de estos elementos y la forma en la que se interrelacionan entre sí puede dar lugar a distintos contenidos o distintas presentaciones de un mismo contenido.

2.1.1.3.Funcionalidad.

La funcionalidad (desde el punto de vista de la gestión de contenidos) es la capacidad de interactuar con una computadora para realizar una determinada tarea. La funcionalidad debe

poder ser gestionada mediante el CMS del mismo modo que se gestiona un artículo de noticias. Dependiendo del contexto, un portal debe poder mostrar distintas funcionalidades.

Podemos considerar que un botón, una imagen a la que se le ha asociado un hiper enlace en una página web o el menú que aparece y permite navegar por el sitio web, forman parte de los elementos que dotan de funcionalidad al portal web. Todos ellos, pues, deberían poder ser gestionados en mayor o menor medida.

2.1.2.Estructura del Contenido.

Desde el punto de vista de la gestión del contenido, la estructura puede considerarse una pieza de crucial importancia. Como hemos mencionado en el apartado anterior, lo que realmente queremos es que una parte de información que hemos despojado de su contexto original (despiezándola en partes más pequeñas) pueda procesarse de forma adecuada por un ordenador pero sin perder esa "unidad" que le hacía tener sentido. En este punto, la estructura del contenido es el pegamento necesario para mantener esa unidad.

La estructura es el centro de la gestión. Es un conjunto de relaciones entre piezas individuales de contenido, que permite su tratamiento de múltiples maneras. La estructuración del contenido puede permitir organizar la información para su procesado, aunque también debe ser lo suficientemente flexible como para poder servir a otros objetivos. Por ejemplo (desde un punto de vista simplista), un administrador puede querer ver los contenidos por categorías u ordenados por editores que han enviado la información, mientras que un usuario desearía ver los contenidos existentes sobre un tema determinado. El hacer la estructura del contenido lo suficientemente flexible no es simple, y puede llegar a convertirse en el principal problema de definición de datos del portal.

Para conseguir la estructuración del contenido, se pueden seguir varios modelos:

1. Uno, el ya comentado modelo de los metadatos, consiste en incluir información adicional en forma de etiquetas al contenido introducido. Estas etiquetas pueden contener información asociada tal y como puede ser el usuario que lo creó, la/s categoría/s a la/s que pertenece, la fecha en que fue creado, su estado dentro del flujo de trabajo... Toda esta información asociada permite añadir el contexto necesario al contenido para que sea comprendido.
2. Una base de datos también puede verse como una forma de conseguir la estructuración del contenido. El modelo entidad-relación de la misma y algunos datos almacenados nos permiten poner el contenido en su contexto y los diferentes campos dentro de las tablas que conforman la base de datos nos permiten una capacidad de almacenamiento superior.

Evidentemente la forma en la que se accede y la forma en que se tratan estos dos modelos es completamente distinta. Y debe ser el CMS el encargado de ofrecer las herramientas necesarias tanto para la estructuración como para la gestión de la estructura del contenido.

Para estructurar el contenido de una forma correcta deben tenerse en cuenta varios aspectos:

- Se ha de tener una comprensión del contenido en su conjunto. Una división estructural rápida sin tener en cuenta los detalles puede llevar a enormes problemas que hagan necesario incluso rehacer el estudio. Si esto sucede en un primer estadio no es demasiado problemático, pero si sucede en un estadio avanzado del proceso de estructuración de la información puede provocar que nada de lo que se haya hecho tenga utilidad.

- Resistencia a cambios futuros. Se debe crear una estructura que no sólo se adapte correctamente a lo que se necesita en el momento, sino que sea capaz de adecuarse a futuras necesidades del contenido a almacenar y procesar.
- Debe proporcionar flexibilidad. Puede que la que se definió en un principio pueda cambiar conforme avanza el proceso iterativo de creación.
- Enfatizar el uso de estándares. Si varios editores están acostumbrados a aplicar distintos estilos a los mismos elementos, al permitir la edición del contenido mediante una herramienta WYSIWYG podemos encontrarnos con elementos idénticos que se han renderizado de formas diferentes. Si se estandariza el formato de los hiperenlaces, la posición de las imágenes, etc. se evitarían estos inconvenientes.

De cualquier forma, un CMS debería proporcionar las herramientas necesarias para tratar con estos problemas. Por ejemplo, si en lugar de tener una herramienta WYSIWYG se tiene un formulario de edición en el que están claramente determinados todos los campos, se podría conseguir que el editor sólo tuviera que indicar cuál es la imagen que se ha de añadir al contenido, no la posición que ocupa. De eso ya se encargaría automáticamente el gestor. Y así quedaría solucionado el último problema de la lista anterior.

2.1.3.Formato del Contenido.

El formato del contenido nos proporciona la forma en la que éste se mostrará a un visitante/usuario. Si cierto texto debe ir centrado, o en cursiva o debe aparecer aquí o allí es decisión del formato del contenido. Es un punto trascendental en los sistemas actuales de cara a una clara comprensión de la información subyacente. Una enfatización en un cierto lugar del texto (con las letras en negrita, por ejemplo) permite centrar la atención del visitante sobre ese lugar más que sobre el contorno.

Uno de los objetivos claros de un gestor de contenidos debe ser el de proporcionar una separación clara y efectiva entre el formato, la lógica y la presentación. Es la llamada separación de conceptos (SoC). En el ámbito de las ciencias de la computación, la separación de conceptos es el proceso mediante el cual se consigue dividir un programa en distintas características que se superponen en funcionalidad lo menos posible. Para una mayor información sobre la separación de conceptos se puede consultar [SOC2006] y de un modo más enfocado al desarrollo web [COCOONSOC2006].

Mediante la separación de conceptos conseguimos que el mismo contenido pueda ser reutilizado de múltiples formas mediante la aplicación de lógicas distintas. Esta sería la interacción entre la lógica y el contenido. Además, este contenido puede ser formateado de distintas maneras mediante la interacción entre el estilo y el contenido. Por tanto, un cambio en la lógica no afectaría a la presentación del contenido y viceversa.

2.1.4.Componentes de contenido.

Cada documento o página web está formada por varios tipos de contenido diferentes, tales como imágenes, texto, audio o vídeo. Trabajar con cada uno de estos tipos por separado es relativamente simple, pero tener control sobre varios de ellos a la vez puede resultar bastante complicado. El texto puede ser tratado de forma rápida y eficiente mediante un editor de textos debido a que existen muchos en el mercado. Las imágenes también podrían ser tratadas de forma efectiva mediante un programa de edición. Pero pensemos ahora en una página web en la que, por ejemplo, los textos de los enlaces de toman de una tabla de la base de datos, los textos de los párrafos se toman de otra y las imágenes se toman de una tercera tabla. No hay ningún editor que permita tratar todos estos contenidos a la vez, por lo que debería crearse uno específico o editar cada uno de los aspectos de la página de forma independiente y luego agregarlos. Ambas soluciones añaden complejidad en comparación con la operación simple de edición de texto.

Un CMS debería conceder la oportunidad de trabajar con varias piezas de contenido de forma efectiva. A cada una de estas piezas las denominaremos componentes de contenido. La granularidad del componente está determinada por el tipo de CMS que se use y puede ir desde líneas hasta noticias enteras pasando por cualquiera de los elementos intermedios existentes. En general, los componentes de contenido deberían estar almacenados en un repositorio y deberían estar autocontenidos, es decir, cada uno de ellos debería tener significación por sí mismo y sin necesidad de relacionarse con los demás. En la Figura 2 podemos observar con más claridad qué es un componente de contenido. En la parte de la izquierda se muestra una página web completa. En la derecha, se muestra la misma página desmenuzada en sus componentes de contenido.

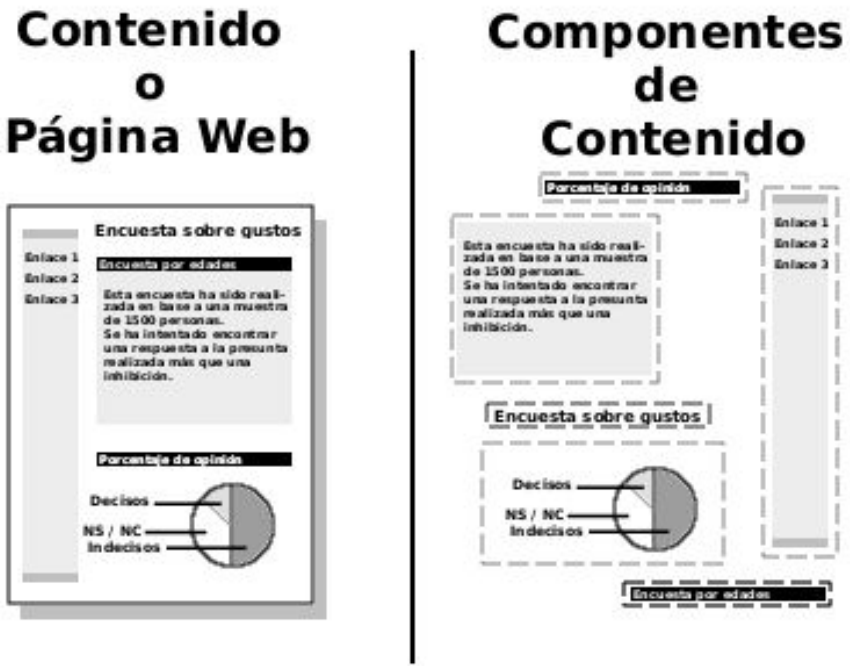

Figura 2: Separación del Contenido en sus Componentes.

Por otro lado, si agrupamos varios componentes de contenido tenemos un documento. Existen otros sistemas denominados *Sistemas de Gestión de Documentos* que proveen la misma funcionalidad que los CMS excepto por el hecho de que trabajan a nivel de documento, esto es, sin poder gestionar por separado las partes que lo componen. Como consecuencia de este hecho, pierden flexibilidad con respecto a los CMS. Aún así, algunas veces se tiende a confundir a ambos tipos de sistemas, aunque siempre se podría argumentar que un Sistema de Gestión de Documentos es un CMS con la granularidad de sus componentes llevada al máximo. De cualquier modo, el estudio de este tipo de sistemas escapa al alcance de esta memoria. Por tanto a lo largo de ella nos centraremos en los Sistemas de Gestión de Contenidos.

2.2.Gestión de Contenidos.

En el apartado anterior hemos visto los diferentes puntos de vista acerca de la información y de la gente que trata con ella. El formato y la estructura de los datos, el contenido, todos necesitan una herramienta capaz de limar al máximo los problemas relacionados con su tratamiento. Esa herramienta es el Sistema de Gestión de Contenidos (al que anteriormente nos hemos referido como CMS).

Una definición algo más formal de CMS podría ser la siguiente:

"Un Sistema de Gestión de Contenidos es un sistema fabricado a partir de un mínimo de tres aplicaciones: la de gestión del contenido, la de gestión del metacontenido y la de la publicación del contenido. Su propósito es el de gestionar por completo el ciclo de vida de los componentes de contenido y metacontenido en un repositorio mediante un Workflow, con el objetivo de mostrar dinámicamente el contenido de una forma agradable para el usuario del portal o sitio web".

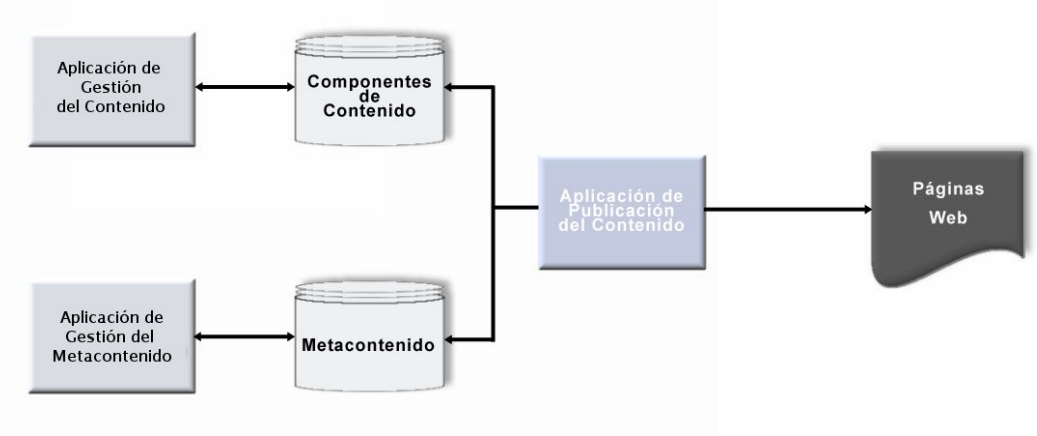

Figura 3: Diagrama Básico de un CMS.

Como se puede comprobar en la figura anterior, la aplicación de gestión del contenido mantiene todos los aspectos relacionados con los componentes de contenido. Del mismo modo, la aplicación de gestión del metacontenido se encarga de éste. La aplicación de publicación del contenido see encarga de generar las páginas web extrayendo los componentes de contenido y metacontenido de de sus respectivos repositorios.

A lo largo de este apartado comentaremos la problemática de la gestión de la información y haremos un esbozo sobre cuáles serían la estructura o los servicios que debería presentar un CMS al usuario. Así mismo trataremos en más detalle las tres aplicaciones básicas que todo CMS debería incluir. Se puede hacer un estudio con más detenimiento sobre estos apartados en [CMSBIBLE2002] y [ASPCMS].

2.2.1.Estructura de un CMS.

En general, un CMS se encuentra formado por un mínimo de tres elementos: la aplicación de gestión de contenido (AGC), la aplicación de gestión de metacontenido (AGM) y la aplicación de publicación del contenido (APC). Algunos CMS tienen más elementos, pero todos ellos tienen al menos los citados.

La AGC gestiona los componentes de contenido del CMS. La AGM, por otro lado, gestiona la información sobre los componentes de contenido. Por último, la APC se encarga de proveer una forma de mostrar los componentes de contenido a los usuarios o visitantes del sitio web.

2.2.1.1.La aplicación de gestión del contenido (AGC).

De un forma simple, podemos decir que la AGC se encarga de la gestión del ciclo de vida de los componentes del contenido, desde su principio hasta su fin. Una AGC será la encargada de crear, mantener y borrar los componentes de contenido dentro del repositorio. Éste puede ser una base de datos, un sistema de archivos o una combinación de ambos. El proceso de gestión es secuencial per natura y se encuentra moderado por el motor de workflow. En general, se habla de la AGC como la parte de administración del CMS.

La AGC permite al autor del contenido el desarrollo de los componentes de contenido sin tener que conocer HTML o entender la arquitectura web que soporta al gestor. En esta aplicación se suele implementar la seguridad a través de roles aplicados a los usuarios. Dependiendo del rol adquirido por el usuario tendrá unos permisos de creación o modificación distintos. Hablaremos sobre los roles con algo más de detalle en el punto 2.2.2.1. El propósito de la AGC es encargarse del control del ciclo de vida de los componentes de contenido y manejar las transiciones de estos componentes a otros estados dentro del workflow. En la Figura 4 se muestran algunos de los estados más comunes que la AGP debería tener en cuenta a lo largo del ciclo de vida.

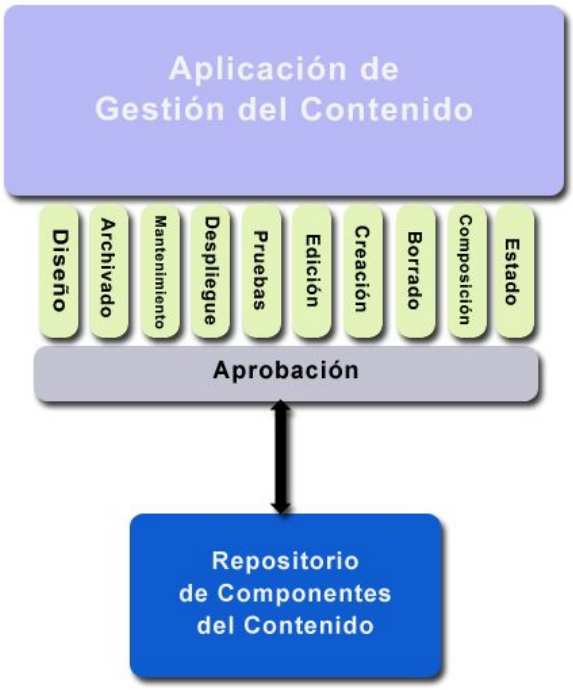

Figura 4: Aplicación de Gestión del Contenido.

2.2.1.2.La aplicación de gestión del metacontenido (AGM).

La AGM es la encargada de la gestión de todo el ciclo de vida del metacontenido (metadatos). Debe pensarse en éste como un conjunto de información sobre los componentes de contenido, en particular sobre cómo se estructuran estos componentes en el sitio web.

El proceso del ciclo de vida del metacontenido recuerda al gestionado por la AGC, pero está enfocado de una forma completamente diferente: la generación de metadatos en lugar de la generación de contenido, esto es, la generación y mantenimiento de la información sobre el contenido en lugar de la creación y mantenimiento del contenido mismo.

Tal y como se vio para la AGC, al final de cada uno de los estados del ciclo de vida del metacontenido, éste debería estar en un estado más estable y maduro que en el anterior. En la Figura 5 podemos observar algunos de los estados más comunes que el ciclo de vida de una AGM debería contemplar.

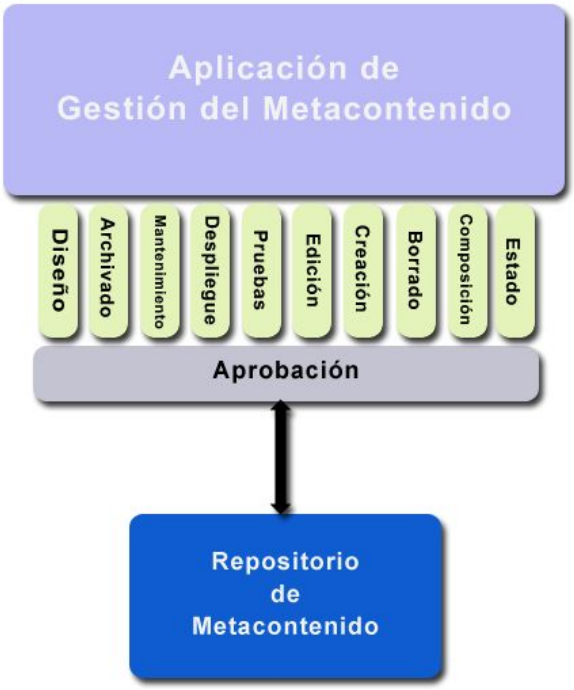

Figura 5: Aplicación de Gestión del Metacontenido.

2.2.1.3.La aplicación de publicación del contenido (APG).

La APG es la encargada de sacar los componentes del repositorio del CMS y mostrarlos a los diferentes visitantes y usuarios del portal. Para este fin hace uso de la información guardada en el metacontenido. Los usuarios del CMS generalmente no tienen nada más que ver con la APG que la configuración de sus características.

Una buena APG se gestiona completamente a través de los metadatos. Esto significa que el metacontenido determina qué se muestra y cómo se muestra. Virtualmente, hay un número ilimitado de formas en las que el metacontenido puede determinar cómo se realiza esta función, todo depende de cuán imaginativo sea el equipo de creación de las plantillas, scripts y la programación.

Debido a que no hay información guardada sobre el estilo de la presentación en la APG, el color, el espaciado, las fuentes, la disposición de los elementos, etc. pueden ser modificados de forma dinámica usando los metadatos, tal y como se hace para el contenido del sitio web cuando se modifica la presentación. Esto significa que, con una planificación cuidadosa, un sitio web no tiene por qué dejar de estar activo mientras se realizan cambios en el estilo del mismo. En el ejemplo de la página web, podemos configurar dentro de la metainformación de la misma la hoja de estilo css que le será de aplicación. El uso de esta hoja junto con las etiquetas asignadas a cada elemento de la página, permite que la presentación sea cambiada “al vuelo” sin tener que retirar el contenido hasta que los cambios surtan efecto. Este ejemplo es muy simple y muestra sólo una de las facetas de la APG, aunque sirve para aclarar un poco los conceptos.

La metainformación también determina la navegación del portal usando hiperenlaces y mapas de imágenes. Lo único que una buena APG necesita saber sobre la navegación del sitio web es cómo cargar la página de inicio por defecto y cómo cargar una URL con un formato correcto. A través del formato de la URL es capaz de saber dónde tiene que buscar el contenido y cómo añadirle los elementos de presentación, aunque en determinados casos, también puede indicar qué tipo de procesamiento se debe aplicar a este contenido.

La APG sólo debe tener acceso de lectura al repositorio. De esta forma se incrementa la seguridad del portal ya que un usuario no será capaz de cambiar los componentes de contenido que esté viendo. El acceso de lectura al sistema de archivos y a la base de datos también tiene el beneficio de que no es necesario el bloqueo del contenido, por lo que se permite el acceso a varios visitantes a la web al mismo tiempo sin problemas de colisiones. Además, puesto que los datos no cambiarán (a menos que se desplieguen desde el repositorio), se pueden implementar mecanismos de almacenamiento en caché para mejorar el rendimiento en la petición del contenido.

2.2.2.Servicios de un CMS.

A lo largo de este punto comentaremos varios servicios que todo gestor de contenidos debería implementar. Estos son necesarios para tratar los problemas expresados en el apartado anterior. En cada uno de los siguientes apartados se comentará un servicio o funcionalidad que el CMS debería presentar al usuario.

2.2.2.1.Gestión del Flujo de Trabajo (Workflow).

Este servicio se encarga de controlar los estados en que se encuentra el documento a publicar en el gestor. Debería controlar que el mismo documento no fuera modificado por dos personas diferentes al mismo tiempo, mediante algún mecanismo de bloqueo del archivo. Además, debería controlar que se cumplen todas las condiciones necesarias para la transición entre los estados del documento (por ejemplo, los roles de los usuarios o los grupos a los que pertenecen). En un modelo simple debería tener, al menos, tres estados:

- Edición: estado en el que el documento se encuentra cuando se ha creado o cuando se está modificando un documento ya existente.
- Revisión: estado en el que el documento se encuentra cuando se ha completado y espera validación por parte de otra persona para ser publicado.
- Publicación: estado en el que el documento se encuentra cuando ha sido revisado y encontrado satisfactorio. A partir de ese momento pasa a estar disponible para todos los visitantes del portal.

Como hemos comentado, este servicio se encarga de comprobar que se cumplen todas las condiciones necesarias para acceder a un estado determinado. Generalmente, en estas condiciones se encontrarán involucrados los roles de los usuarios.

Los roles son unos papeles que ciertos usuarios juegan dentro del gestor de contenidos. Son necesarios para poder distribuir el trabajo entre varias personas, uno de los objetivos fundamentales que un CMS debería cumplir, ya que la reducción de la carga de trabajo es fundamental. Dentro de estos roles podríamos definir tres básicos:

- Editores. Encargados de la creación y edición de prácticamente cualquier contenido del portal. Son los que alimentan al portal de datos y permiten un crecimiento rápido del mismo. No tiene por qué ser personal cualificado, ya que las herramientas que utilicen dentro del CMS deben abstraerles de la complejidad subyacente.

- Revisores. Encargados de comprobar que los contenidos creados son válidos y adecuados para la publicación en el portal. Son los encargados últimos de las decisiones sobre la idoneidad o no de los contenidos creados por los editores.
- Administradores. Encargados de la gestión de los usuarios y los roles de los mismos. También serán los encargados de la gestión de los grupos de afiliación de los mismos.

Como hemos indicado, estos roles son los básicos. Se deberían poder añadir y/o modificar la granularidad de los mismos, así como la asignación de varios roles a un mismo usuario, lo que le permitiría realizar todas las funciones asociadas a cada uno de ellos.

Del mismo modo que los roles, los grupos pueden ayudar a controlar el estado de los documentos y la transición entre estados dentro del workflow. Mediante la combinación de roles y grupos se debería poder alcanzar la granularidad necesaria y suficiente para el cumplimiento de todos los objetivos fijados en la creación del workflow de los documentos del portal. Todo esto conlleva una simplificación en la gestión del trabajo, ya que es posible dividir las tareas entre varias personas de una forma ordenada y sin conflictos.

2.2.2.2.Gestión de usuarios, grupos y administradores.

Este servicio se encarga de todo lo relacionado con el alta, la edición o el borrado tanto de los usuarios como de los grupos de usuarios y los administradores.

Por usuarios entendemos visitantes del portal que pueden tener varios roles dentro del gestor: editores, revisores, etc. Este servicio permitirá la creación de nuevos usuarios y la asignación de roles a los mismos. También debe implementar los mecanismos necesarios para la asignación de un usuario a uno o varios grupos. Debería permitir que a un usuario se le asignaran uno o varios roles. También debe permitir la modificación de los datos del usuario (incluyendo la modificación del rol asociado al mismo) y su eliminación del conjunto de usuarios del portal.

Por grupos entendemos grupos de usuarios que realizan una determinada función y tienen una visión distinta del portal a la que tendrían como usuarios individuales o miembros de otros grupos.

Por administradores entendemos las personas encargadas de la gestión de los grupos y de los usuarios. Se deben poder crear y modificar tal y como se haría con los usuarios. Además, estos administradores deberían poder crear grupos y asignar los usuarios a los mismos.

2.2.2.3.Servicio de Edición.

Este servicio sería el encargado de controlar la edición de los documentos del portal. Un portal web moderno puede estar formado por varios tipos de documentos distintos: artículos de noticias, listados de actividades, formularios de inscripción, etc. Todos estos documentos deben tener una interfaz de edición a través del CMS que permita a las personas responsables de esta tarea realizarla sin tener conocimientos de la complejidad que hay tras ella.

La interfaz debería ser lo más simple y homogénea posible. Simple para permitir a personas no especialistas en el desarrollo web la introducción de los datos y contenidos necesarios para que el portal pueda crecer sin problemas. Homogénea para que una persona que esté encargada de la introducción de datos en varios tipos de documentos distintos no tenga dificultades. También sería importante que la interfaz fuese lo más rígida posible en el sentido de que sólo se pudiera introducir información, y no se pudiera aplicarle estilo. Así se conseguiría que la imagen general del portal se mantuviera, ya que los contenidos almacenados se mostrarían renderizados con el estilo del sitio web.

Este servicio estaría controlado por el motor de workflow. Tomaría el estado del

documento y el rol o grupo del usuario que intenta modificarlo y comprueba que es posible la edición del mismo por parte del usuario. También debería permitir, una vez que el usuario ha acabado la edición, tanto el guardado de la información editada (si no se ha finalizado) como el envío del documento acabado al revisor o validador correspondiente.

2.2.2.4. Gestión y control de versiones.

Este servicio debería encargarse de controlar la versión del documento que actualmente se encuentra publicado, en revisión o archivado. De esta forma se tendría siempre un conocimiento pleno de qué es exactamente lo que tenemos disponible en cada uno de los estados del workflow o en el repositorio de versiones. Si hiciera falta, este servicio debería proporcionar la posibilidad de elegir una versión antigua de un documento para su publicación en lugar de una versión más reciente pero que tuviera fallos.

A este servicio se le puede encargar también la tarea de comprobar la integridad de lo publicado, es decir, la comprobación de que no existen enlaces rotos a documentos internos entre versiones distintas.

2.2.2.5. Rendimiento y optimización del tiempo de acceso.

Este servicio debería proporcionar herramientas suficientes para controlar los mecanismos de almacenamiento temporal de páginas (también llamados mecanismos de caché de páginas).

Si el portal gestionado está formado en su mayoría por páginas HTML estáticas, estos mecanismos no serían necesarios puesto que el acceso a las mismas es muy rápido (no se tienen que generar datos bajo demanda, todo se haya almacenado en un archivo HTML). Sin embargo, si el portal gestionado contiene páginas generadas dinámicamente, con accesos a formularios, consultas a bases de datos, etc., la devoluciones de las páginas pueden tardar cierto tiempo dependiendo del volumen de datos que se manejen. Si el usuario pulsa un enlace a otra página en la que hay parte de datos que ya se han solicitado, la petición y las consultas a la base de datos (si las hubiera) han de ser procesadas de nuevo, aunque la información que de ellas resulte no haya cambiado.

Con estos mecanismos de caché, si la información a mostrar no ha cambiado, el usuario recibe una copia de los datos sin tener que esperar a que se realicen las consultas.

2.2.2.6. Gestión de módulos y componentes.

Si el CMS utilizado soporta la adición de módulos, este servicio se encargaría de mantenimiento de los mismos. A través de él se podrían activar o desactivar (permitiendo añadir o eliminar la funcionalidad presente en ellos).

2.2.2.7. Planificación.

Este servicio sería el encargado de la realización de las tareas de activación y desactivación automática de contenidos.

Debería permitir tanto la programación de estas tareas como la modificación de los datos de las mismas. Idealmente debería ser independiente del tipo de contenido o documento tratado, consiguiendo así una capa de planificación flexible e independiente de lo publicado. Aún así, hay ciertos gestores que aunque no tienen integrado el planificador, poseen mecanismos para la publicación automática de cierto tipo de documentos, aunque estos mecanismos sí que son dependientes del mismo.

2.2.2.8.Motor de búsqueda.

Este servicio sería el encargado o bien de integrar un motor de búsqueda para nuestro portal o bien de gestionar el motor ya integrado.

Un motor de búsqueda ayuda a los visitantes de nuestro portal a encontrar la información que desean de una forma mucho más rápida que la que conseguirían visitando sección por sección. Además, la integración del motor puede ayudar a una mejor estructuración y categorización de la información contenida en el portal. Esto podría facilitar el desarrollo de los algoritmos necesarios para la realización de la consulta.

Habría dos aproximaciones: el uso de un motor externo (lo que generalmente conllevaría un desembolso extra al compra los derechos de uso de tal motor en nuestro sitio web) o la integración de un motor de búsqueda acorde a nuestras necesidades. Dependiendo del tamaño, la especialización del sitio web y la estructura de los datos almacenados en el mismo, puede ser más conveniente una solución que la otra. De cualquier forma, es necesario disponer de las herramientas necesarias para la configuración de cualquiera de las dos alternativas.

2.2.3.Aspectos a considerar en la elección de un CMS. Planes de implantación.

La introducción de un Gestor de Contenidos en un portal web ya existente o dentro de una organización puede resultar muy costosa tanto en tiempo como en fondos. Una mala estructuración del proceso, la elección de una herramienta incorrecta o la necesidad de una implantación rápida pueden ser grandes enemigos para la consecución de un resultado satisfactorio. A lo largo de este punto intentaremos dar una idea de los pasos que deberían seguirse así como comentaremos los diferentes problemas que podemos encontrar en la implantación del Gestor. Por último, haremos una pequeña descripción de acerca de las bases tecnológicas de algunos de los productos *OpenSource* que se pueden encontrar en el mercado.

2.2.3.1.Consideraciones Generales.

Conforme la sociedad de la información se ha ido asentando entre nosotros a través de medios tales como internet, hemos visto crecer de una forma desorbitada tanto el volumen como la necesidad de la información que podemos obtener a través de la red de redes.

La inversión a realizar en el CMS puede ser más o menos grande dependiendo del volumen de información que se pretenda manejar, la existencia de herramientas que se encarguen de este trabajo, el tamaño de la compañía que quiere implantarlo, etc. Aunque siempre será un desembolso importante, también es cierto que la aparición en el mercado de productos *OpenSource* altamente personalizables ha abaratado muchos de los costes asociados a la exclusividad de modificación de los programas de código cerrado. Estas modificaciones sólo podían ser realizadas por empresas con licencia del fabricante del producto, lo que nos hacía caer en mercados cerrados con el aumento de costes y la falta de estandarización que estos han traído siempre.

A lo largo de este punto asumiremos que la empresas en cuestión ya tiene unos sistemas para la publicación de los contenidos que serán reemplazados por este CMS y que éste gestionará tanto la intranet como el sitio web. Además tendremos en cuenta que puede ser publicado un amplio rango de contenidos a través de este CMS:

- Páginas simples.
- Páginas complejas con una presentación específica.
- Información obtenida de forma dinámica de una base de datos.
- Documentos (manuales en pdf, archivos office, etc).

- Manuales *online*.
- Uso extensivo de enlaces internos.
- Gran número de páginas a gestionar.

2.2.3.2.Análisis de los requisitos de la empresa.

2.2.3.2.1.Objetivos de negocio y estrategias.

Antes de identificar requerimientos específicos del sistema se deberían determinar los objetivos que se pretenden conseguir con la implantación del CMS. Así mismo, se deberían estudiar los objetivos a largo plazo del negocio.

Debería ser posible redactar estos objetivos de forma muy resumida en una página. Además deberían ser comprendidos y reconocidos por todos los componentes de la plantilla antes de comenzar el proceso de definición de requerimientos. De esta forma evitaremos una vuelta atrás en la decisión cuando estemos en un estado avanzado del proceso.

2.2.3.2.2.Identificación de requerimientos.

No hay una lista simple con los mejores requerimientos. Cada empresa tiene unas necesidades únicas.

Este estado debería ser realizado por todo el conjunto de la empresa, así como por los usuarios finales y los grupos de trabajo. Este punto es particularmente importante si el objetivo es el desarrollo de un CMS que gestione toda la información de la empresa. Debe hacerse de una forma estructurada para asegurarse de que la lista de requerimientos es a la vez suficiente y gestionable.

2.2.3.2.3.Estructuración de los requerimientos.

La lista completa de los requerimientos para una CMS empresarial crecerá mucho. Se deben agrupar en categorías para hacer la lista más fácilmente gestionable.

Un esquema clasificatorio podría ser el siguiente:

- Creación del Contenido
- Gestión del Contenido
- Publicación
- Presentación
- Gestión del proyecto y del CMS

Esta lista cubre todo el ciclo de vida del contenido, desde su creación inicial hasta su presentación a los usuarios finales.

2.2.3.3.Listado de requerimientos.

En este apartado intentaremos presentar algunos de los requerimientos que deberían encontrarse dentro de cada una de las categorías listadas en el apartado 2.2.3.2.3. Este apartado no trata de dar una lista completa, sino una primera aproximación que permita el desarrollo de una más exhaustiva.

2.2.3.3.1.Creación del Contenido.

Los requerimientos clave dentro de este apartado serían:

- Integración de un entorno de creación y autoría del contenido.
- Separación del contenido y la presentación.
- Edición multiusuario.
- Reutilización del contenido. El mismo contenido presentado en dos secciones diferentes sólo debería encontrarse almacenado una vez.
- Posibilidad de la creación de metadatos.
- Gestión de los enlaces internos. Deben ser estables ante reestructuraciones del portal.
- Facilidad de uso y eficiencia.

2.2.3.3.2.Gestión del Contenido.

Los requerimientos clave dentro de este apartado serían:

- Control de versión y archivado.
- Workflow.
- Seguridad (para asegurar la integridad del contenido).
- Facilidad o adaptabilidad para integración con sistemas externos.
- Sistema de notificación.

2.2.3.3.3.Publicación.

Los requerimientos clave dentro de este apartado serían:

- Hojas de estilo (la apariencia final debería estar controlada mediante ellas).
- Plantillas para las páginas.
- Escalabilidad (el sistema de publicación debería ser fácilmente extensible o mejorable).
- Soporte para múltiples formatos (HTML, PDF, WAP, etc.).
- Personalización.
- Estadísticas de uso.

2.2.3.3.4.Presentación.

Los requerimientos clave dentro de este apartado serían:

- Usabilidad (facilidad de uso, eficiencia, etc.).
- Accesibilidad (estándares como el WAI [WAI2006]).
- Soporte de varios navegadores.
- Velocidad de carga (debe mantenerse en unos términos aceptables dependiendo de la conexión de los usuarios finales).
- Metadatos (se debe proveer la cantidad suficiente de metadatos como para una indexación efectiva; el *Dublin Core Metadata* [DUBLIN2006], por ejemplo, sería un estándar a intentar aplicar).

2.2.3.3.5.Gestión del proyecto y del CMS.

Los requerimientos clave dentro de este apartado serían:

- Formación sobre el uso del CMS.
- Documentación sobre el CMS (apta para usuarios, administradores o desarrolladores).
- Período de Garantía.
- Mantenimiento (procesos de actualización y nivel de servicio).
- Recursos hardware, software y sistemas operativos requeridos por el CMS.
- Habilidades necesarias dentro de la organización para personalizar y mantener el CMS.
- Escalabilidad (niveles de carga que soporta el CMS y los recursos necesarios para incrementar su uso).
- Restricciones en los equipos y el software (puede ser necesario especificar los sistemas operativos o las bases de datos con las que se cuenta antes de la implantación del CMS).
- Sitios mantenidos por el CMS (una referencia de los sitios web en los que se ha implementado con éxito el gestor).

2.2.3.4.Otras consideraciones.

Para una correcta elección también se deberían tener en cuenta algunas consideraciones adicionales, relativas a la descripción del contenido a gestionar o al gestor a elegir.

2.2.3.4.1.Consideraciones adicionales relativas al contenido.

Se debería, antes de nada, entender el contenido de nuestra organización. Una encuesta sobre la información que debería ser publicada a través del CMS. La naturaleza del contenido a tratar puede influenciar grandemente la funcionalidad requerida del gestor. Es muy importante no perder de vista al contenido cuando se elige el CMS.

También deberían evitarse la inclusión de detalles técnicos. Se deberían enfatizar las necesidades de nuestro negocio, el funcionamiento y no los detalles de implementación. Los proveedores del CMS deberían tener la libertad suficiente como para proponer cualquier método o tecnología que fuera necesario para la consecución de los objetivos del gestor.

Unas descripciones detalladas harían que las necesidades fueran más fácilmente entendibles. Debería intentar evitarse el uso de jerga y especificar al máximo lo que se intenta explicar. Estas descripciones podrían ir acompañadas de ejemplos de situaciones específicas que se pueden dar dentro del negocio. Una visión de la necesidad mediante ejemplos también ayuda a su comprensión y es una manera eficaz de recalcar la importancia de la misma.

2.2.3.4.2.Consideraciones adicionales relativas al CMS.

En el momento de la elección del CMS se deberían tener en cuenta varias consideraciones adicionales:

- Madurez del producto. Es fundamental comprobar la madurez del Gestor de Contenidos que se proponga como alternativa a nuestras necesidades. Un producto demasiado joven puede no llegar a cubrirlas del todo o cambiar mucho su funcionamiento de una versión a otra más actual. Un producto con mucho tiempo en

el mercado puede llegar a estar obsoleto en breve o puede no verse soportado por una empresa o comunidad de usuarios en poco tiempo. Llegar a un compromiso entre estas dos alternativas es fundamental.

- Adecuación a necesidades. El CMS debe poder cubrir todas las necesidades recogidas en la lista de requerimientos que se ha elaborado. Si no se cumple alguna de ellas, se debería observar la posibilidad de una implementación de la misma o bien desechar directamente la herramienta.
- Facilidad de implantación. Entre varios CMS que cubran todos nuestros requisitos, deberíamos elegir el más fácilmente implantable dentro de las posibilidades de la empresa, teniendo en cuenta los datos tanto técnicos (hardware y software necesarios para soportarlo) como de plazos de implantación.
- Eficiencia. La generación dinámica del contenido puede hacer muy lentos algunos gestores cuando la carga es grande o cuando las bases de datos se hallan distribuidas. Habrá que tener en cuenta también si el rendimiento del CMS es suficiente o no para nuestras necesidades.

Evidentemente, a priori no hay una categoría o consideración más importante que otra. El orden en que deberían aplicarse las consideraciones o la ponderación que debería darse a las mismas dependen profundamente de las necesidades de la empresa o del contenido que se haya de tratar. Por esto, una vez más, es imprescindible recoger con claridad las necesidades y las restricciones a las que el gestor deberá enfrentarse antes de intentar entrar en cualquier valoración o elección del mismo.

2.2.3.5.Riesgos.

En esta sección trataremos de apuntar los riesgos a los que se puede enfrentar una empresa al instalar o implementar un CMS para su intranet o su sitio Web. Se trata de dar una visión general, no catastrofista, pero sí para tomar consciencia de la importancia de la reflexión antes de la implantación.

2.2.3.5.1.Tiempo de implantación extenso.

El tiempo de implantación puede ser una restricción importante a la hora de la migración a un CMS. Si la información es grande y no está bien clasificada, si existían varias herramientas de gestión de partes independientes del contenido de la empresa, si se tiene que empezar desde cero y hay que añadir todo el contenido, el tiempo de implantación puede crecer desorbitadamente.

Es importante tener todos estos datos en mente antes de montar una versión estable y visitable por los usuarios. El montaje de una versión inestable (en cuanto a contenidos se refiere) o vacía de contenido puede dar una muy mala impresión de la empresa, cuando el objetivo que se busca desde el principio de la instalación de este tipo de herramientas es el contrario.

2.2.3.5.2.Migración de Datos.

Uno de los grandes riesgos cuando se decide la implantación de un CMS es la posibilidad de pérdida o imposibilidad de traslación de datos almacenados por la empresa.

La existencia de diferentes modelos de datos dentro de la empresa, la gestión de los repositorios de los mismos, las diferentes herramientas utilizadas para gestionarlos pueden provocar que las tareas de migración sean muy complicadas o imposibles.

Tanto en uno como en otro caso, esto provocaría un retraso considerable en la puesta en marcha del gestor.

2.2.3.5.3.Elección incorrecta del CMS.

Este es, sin lugar a dudas, el riesgo más grande que se corre al seleccionar un Gestor de Contenidos. Evidentemente, aparte de la imposibilidad de implementación de alguno de los servicios o necesidades examinadas en la lista de requisitos, supondría una gran pérdida de tiempo y de recursos empresariales la adaptación del sistema a un modelo en el que no encaja plenamente, o la modificación del modelo para hacerlo encajar en el sistema.

Esto provocaría un coste adicional asociado muy importante para la empresa. Pero puede que no fuera el coste más grande. Si esta empresa planeaba su salto de negocio frente a sus competidores en base al portal web, esto provocaría una perdida de posición en el mercado, situación que tendría un coste de valor irreparable.

parte II: aportaciones

capítulo 3:
base tecnológica

En este apartado haremos una breve descripción de los estándares XML, XSLT y CSS, necesarios para el funcionamiento de Lenya. También haremos una breve descripción acerca de la arquitectura de las aplicaciones J2EE y del *framework* en el que se basa Lenya, Apache Cocoon.

3.1.XML.

XML son las siglas de Lenguaje de Marcado Extensible (e*Xtensible Markup Language*). Proviene de SGML [ISO 8879:1986] y al igual que éste es un metalenguaje utilizado para definir otros lenguajes. Para una profundización sobre XML y sus tecnologías relacionadas se pueden consultar [JAVAXML2002], [XMLEJEM2001]. La recomendación del W3C se encuentra en [RECOMXML2004].

Sin embargo, XML es mucho más sencillo que SGML. XML es un lenguaje de marcado que no especifica ni las etiquetas, ni la gramática del lenguaje. El conjunto de etiquetas de un lenguaje de marcado define las etiquetas que tienen significado para un analizador del lenguaje. Por ejemplo, HTML tiene un conjunto estricto de etiquetas permitidas. Se puede utilizar la etiqueta <TABLE> pero no la etiqueta <SILLA>. Mientras que la primera etiqueta tienen un significado específico para una aplicación que use los datos y se utiliza para indicar el inicio de una tabla en HTML, la segunda etiqueta no tiene significado específico, y aunque la mayoría de los navegadores la ignorarán, pueden ocurrir problemas inesperados cuando aparece. Esto ocurre porque cuando se definió HTML, se definieron con él el conjunto de etiquetas del lenguaje. Con cada nueva versión de HTML se definen nuevas etiquetas. Sin embargo, si no se define una etiqueta, no se puede utilizar como parte del lenguaje de marcado sin que se produzca un error al analizar el documento. La gramática de un lenguaje de marcado define la correcta utilización de las etiquetas del lenguaje. Una vez más, utilicemos HTML como ejemplo. Cuando se utiliza la etiqueta <TABLE> se pueden incluir varios atributos, como la anchura, el color de fondo y la alineación. Sin embargo, no se puede definir el TIPO de una tabla porque la gramática de HTML no lo permite.

XML, al no definir ni las etiquetas ni la gramática, es completamente extensible, tal y como indica su propio nombre. Si se quiere utilizar la etiqueta <TABLE> y posteriormente anidar dentro de esa etiqueta varias etiquetas <SILLA>, se puede hacer. Podemos ver esta característica en el siguiente ejemplo.

Listado 3.1: Un ejemplo simple XML

```
<?xml version="1.0"?>

<dining-room>
        <table tipo="redonda" madera="haya">
                <fabricante>Maderera S.A. </fabricante>
                <precio>100€</precio>
        <table>
        <silla madera="haya">
                <cantidad>2</cantidad>
                <color>azul</color>
        </silla>
        <silla madera="pino">
                <cantidad>3</cantidad>
                <color>amarillo</color>
        </silla>
</dining-room>
```

En este ejemplo, las etiquetas y la gramática son completamente inventadas, pero con el analizador correcto es completamente válido.

XML define el lenguaje en sí mismo y proporciona un marco de trabajo de tipo metadatos debido a que marca con etiquetas la información contenida en sus archivos.

Hay dos conceptos que hay que tener muy en cuenta cuando tratemos con documentos XML:

- El documento XML debe estar bien formado para que pueda ser analizado de forma correcta. Un documento bien formado es aquel que tiene cerrada cada etiqueta que se haya abierto, no tiene etiquetas anidadas fuera de lugar y es sintácticamente correcto de acuerdo con la especificación.
- El documento debería ser válido (aunque no es estrictamente necesario). Un documento válido es aquel conforme a su definición de tipo de documento (DTD) o su esquema XML. Tanto el DTD como el esquema XML definen la gramática y el conjunto de etiquetas para un formato XML específico, aunque el esquema XML proporciona más posibilidades que la DTD. Otro esquema de validación posible y muy utilizado en Lenya es el *Relax NG Schema.* Se puede encontrar más información sobre este en [RELAXNG2001].

Debido a su versatilidad, XML ha servido de base para la aparición de otras tecnologías como pueden ser:

- XSL: Lenguaje Extensible de Hojas de Estilo, cuyo objetivo principal es mostrar cómo debería estar estructurado el contenido, cómo debería ser diseñado el contenido de origen y cómo debería ser paginado en un medio de presentación como puede ser una ventana de un navegador Web o un dispositivo de mano, o un conjunto de páginas de un catálogo, informe o libro.
- XPath: Lenguaje de Rutas XML, es un lenguaje para acceder a partes de un documento XML.
- Xlink: Lenguaje de Enlace XML, es un lenguaje que permite insertar elementos en documentos XML para crear enlaces entre recursos XML.
- Xpointer: Lenguaje de Direccionamiento XML, es un lenguaje que permite el acceso a la estructura interna de un documento XML, esto es, a sus elementos, atributos y contenido.
- XQL: Lenguaje de Consulta XML, es un lenguaje que facilita la extracción de datos desde documentos XML. Ofrece la posibilidad de realizar consultas flexibles para extraer datos de documentos XML en la Web.

Apache Lenya, como veremos en un apartado posterior, proporciona a Apache Cocoon las cualidades necesarias para convertirlo en un Sistema de Gestión de Contenidos. Como veremos en el mismo apartado, tanto Lenya como Cocoon hacen un uso extensivo de XML. Sus archivos de configuración, sus hojas de estilo (hojas XSLT) e incluso los archivos en los que se almacena la información, son archivos XML (aunque de diferentes tipos y sujetos a diferentes espacios de nombres y validación).

3.1.1.Espacios de nombres.

Un espacio de nombres es una correspondencia entre un elemento prefijo y una URI (*Uniform Resource Identifier*). Esta correspondencia se utiliza para manejar las colisiones en el espacio de nombres de los elementos y definir las estructuras de datos que posibilitan a los analizadores la gestión de las mismas. Por ejemplo, si IBM sólo usa las URIs en el dominio ibm.com y Sun sólo usa las que estén en el dominio sun.com, entonces no existirá ninguna confusión entre un mismo elemento XML que se llame de la misma forma en los dos espacios de nombres.

Las URIs son cadenas de caracteres que sirven a modo de identificadores. Incluso si una

URI es una URL (*Uniform Resource Locator*), el analizador no se conecta al servidor e intenta descargar el documento que se encuentra indicado por ésta. Es más, probablemente el documento ni siquiera existe. Como ejemplo, podemos ver que las siguientes URLs identifican la misma página (o documento), pero sin embargo también identifican tres espacios de nombres distintos:

- http://ns.dominio.com/prueba
- http://ns.dominio.com/prueba/
- http://ns.dominio.com/prueba/index.html

Como puede darse el caso de que una URI contenga caracteres ilegales en los nombres de los elementos, así como que sea excesivamente larga, se pueden utilizar prefijos cortos en lugar de las URIs. Estos prefijos se encontrarán separados de los nombres locales por dos puntos. Por ejemplo, en lugar de la URI *http://www.w3c.org/2001/XInclude* se podría utilizar el prefijo *xinclude* o bien *xi*. De esta forma, un elemento XML llamado *incluir* que pertenezca a ese espacio de nombres podría ser escrito de la forma *xi:incluir*. Este elemento tendría por prefijo *xi*, por nombre local *incluir*, su nombre cualificado sería *xi:incluir* y su espacio de nombres la URI *http://www.w3c.org/2001/XInclude* .

Cada prefijo usado en un elemento o un nombre de atributo debe estar unido a una URI. Si no lo está se provoca un error de espacio de nombre mal formado. Una URI, además, puede tener varios prefijos distintos para identificar el mismo espacio de nombres.

También puede definirse un espacio de nombres que será el aplicado por defecto a todos los elementos de un archivo XML que no tengan un prefijo. Ese espacio de nombres también le será aplicado a los descendientes del elemento en cuestión.

A los atributos de un elemento también se les pueden asignar espacios de nombres, aunque si el atributo en cuestión no tiene un prefijo, se considera que no pertenece a ningún espacio. Por tanto, los atributos no estarán en el mismo espacio de nombres del elemento al que acompañan a no ser que se especifique.

3.2.XSLT.

XSLT son las siglas de *eXtensible Stylesheet Language for Transformations* y es una recomendación oficial del *World Wide Web Consortium* (W3C) [XSLTRECOM]. XSLT es un lenguaje flexible y poderoso que permite transformar documentos XML en otro tipo de documentos. Estos son algunos de los tipos de documentos en los que permite transformar un documento XML dado:

- HTML
- Otro documento XML
- PDF
- archivo SVG (archivo de gráficos vectoriales escalable)
- VRML (lenguaje de modelado de realidad virtual)
- Código Java
- Archivo de texto plano
- Imagen JPEG

Una hoja de estilo (también llamada hoja lógica) XSLT define las reglas de transformación para el documento XML y un procesador de instrucciones XSLT es el que realiza el trabajo de transformación. Se pensó en XSLT como un lenguaje flexible que permitiera transformar documentos de forma independiente a un lenguaje de programación. Aunque los motores o

procesadores de hojas de estilo XSLT pueden estar escritos en lenguajes de programación distintos, no es necesario conocerlos para conseguir transformar un documento XML. Sólo es necesario conocer las reglas de XSLT y definir la transformación. Este comportamiento incluye una capa extra de abstracción que permite pasar por alto las peculiaridades de procesado de un lenguaje de programación en particular.

Las transformaciones XSLT pueden tener lugar tanto en el cliente como en el servidor, aunque las transformaciones en el servidor son las que predominan en la actualidad dado que la mayoría de los buscadores de la actualidad no son compatibles con XSLT.

Lenya utiliza la tecnología de las hojas de estilo XSLT para transformar los ficheros fuentes en su viaje a lo largo de las *pipelines,* desde el inicio de la petición por parte del cliente hasta la devolución del resultado. Pueden aplicarse una o varias hojas en este proceso de transformación, y cada una de ellas puede dar como resultado documentos con un formato distinto, e, incluso, otra hoja XSLT que pueda ser aplicada también en la transformación. En el Apéndice A se puede encontrar un ejemplo completo sobre XSLT.

3.2.1.Algunas características de XSLT.

Algunas de las principales líneas de diseño de XSLT fueron las siguientes:

- Una hoja de estilo XSLT debería ser un documento XML. Esto significa que se puede escribir una hoja de estilo que transforme a otra hoja de estilo. Este comportamiento recursivo es muy común en la tecnología XSLT.
- El lenguaje XSLT debería estar basado en búsqueda de patrones. La mayoría de las hojas de estilo constan de una seria de reglas (llamadas *template* en XSLT) que se utilizan para la transformación de un documento. Cada una de estas reglas se encarga de encontrar un patrón en un archivo y de sustituirlo por el contenido indicado en la regla.
- Las plantillas XSLT deberían estar libres de efectos laterales o condiciones de carrera. Esto es, XSLT se ha diseñado de forma que se pueden aplicar varias reglas de la hoja de estilo a la vez. El mayor impacto de esta característica es que las variables de las hojas de estilo no se pueden modificar. Una vez que se ha inicializado el valor de una variable, éste no puede ser cambiado.
- En XSLT no se pueden utilizar bucles del modo en que se haría en otro lenguaje de programación. Puesto que los valores de las variables no se pueden cambiar, sería imposible realizar un bucle `while` o un bucle `for`. Por esto, XSLT usa dos métodos alternativos: la iteración y la recursividad. Mediante la iteración, se puede conseguir hacer que se busquen en un archivo todas las apariciones de un patrón dado, consiguiendo un comportamiento parecido a un bucle `do-while` (sigue iterando mientras queden elementos que procesar). Mediante la recursividad conseguimos comportamientos similares a los de los bucles `for`.

3.2.2.XPath.

XPath forma parte de otra recomendación del W3C [RECOMXPATH1999] y está diseñado para ser utilizado por XSLT y Xpointer. El principal objetivo de XPath consiste en definir un mecanismo para tratar las partes de un documento XML, lo cual significa que se utiliza para apuntar a nodos de elementos, nodos de atributos, nodos de texto y cualquier otra cosa que pueda aparecer en un documento XML. XPath trata dichos nodos como parte de una estructura en árbol en lugar de tratar el XML como una cadena de texto. XSLT confía también en la estructura en árbol que define XPath. Además de esto, XPath contiene un conjunto de funciones para formatear texto, convertir cifras y tratar con datos booleanos.

A diferencia de XSLT, XPath no se expresa utilizando la sintaxis XML. Una sintaxis

simplificada resulta razonable si tenemos en cuenta que XPath se utiliza normalmente dentro de valores atributivos que se hayan dentro de otros documentos XML. XPath está formado por una sintaxis compleja y un conjunto de abreviaturas, cuyo resultado se parece mucho a los nombres de ruta de un sistema de archivos o de un sitio web [JAVAXSLT2002].

3.3.CSS.

Las Hojas de Estilo en Cascada (*Cascading Style Sheets*), son un mecanismo simple que describe cómo se va a mostrar un documento en la pantalla, o cómo se va a imprimir, o incluso cómo va a ser pronunciada la información presente en ese documento a través de un dispositivo de lectura. Esta forma de descripción de estilos ofrece a los desarrolladores un amplio control sobre estilo y formato de sus documentos.

CSS se utiliza para dar estilo a documentos HTML y XML, separando el contenido de la presentación. Los *Estilos* definen la forma de mostrar los elementos HTML y XML. CSS permite a los desarrolladores Web controlar el estilo y el formato de múltiples páginas Web al mismo tiempo. Cualquier cambio en el estilo marcado para un elemento en la CSS afectará a todas las páginas vinculadas a esa CSS en las que aparezca ese elemento.

CSS funciona a base de reglas, es decir, declaraciones sobre el estilo de uno o más elementos. Las hojas de estilo están compuestas por una o más de esas reglas aplicadas a un documento HTML o XML. La regla tiene dos partes: un selector y la declaración. A su vez la declaración está compuesta por una propiedad y el valor que se le asigne.

```
h1 {color: red;}
```

`h1` es el selector

`{color: red;}` es la declaración

El selector funciona como enlace entre el documento y el estilo, especificando los elementos que se van a ver afectados por esa declaración. La declaración es la parte de la regla que establece cuál será el efecto. En el ejemplo anterior, el selector `h1` indica que todos los elementos `h1` se verán afectados por la declaración donde se establece que la propiedad `color` va a tener el valor `red` (rojo) para todos los elementos `h1` del documento o documentos que estén vinculados a esa hoja de estilos.

Las tres formas más conocidas de dar estilo a un documento son las siguientes:

- Utilizando una hoja de estilo externa que estará vinculada a un documento a través del elemento <link>, el cual debe ir situado en la sección <head>.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN">
<html>
  <head>
    <title>Título</title>
    <link rel="stylesheet" type="text/css"
      href="http://www.w3.org/css/officeFloats.css" />
  </head>
  <body>
  .
  .
  </body>
</html>
```

- Utilizando el elemento <style>, en el interior del documento al que se le quiere dar estilo, y que generalmente se situaría en la sección <head>. De esta forma los estilos serán reconocidos antes de que la página se cargue por completo.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN">
```

```
<html>
  <head>
    <title>hoja de estilo interna</title>
    <style type="text/css">

      body {
        padding-left: 11em;
        font-family: Georgia, "Times New Roman", serif;
        color: red;
        background-color: #d8da3d;
      }

      h1 {
        font-family: Helvetica, Geneva, Arial, sans-serif;
      }

    </style>
  </head>
  <body>
    <h1>Aquí se aplicará el estilo de letra para el Título</h1>
  </body>
</html>
```

- Utilizando estilos directamente sobre aquellos elementos que lo permiten a través del atributo `<style>` dentro de `<body>`. Pero este tipo de estilo pierde las ventajas que ofrecen las hojas de estilo al mezclarse el contenido con la presentación.

Algunas normas básicas a la hora de crear una CSS son las siguientes:

- En el siguiente ejemplo, `h1{color: red;}`, el *selector*, `<h1>`, le dice al navegador la parte del documento que se verá afectada por esa regla. Los selectores pueden aparecer individualmente o agrupados, separándolos con comas:

```
h1, h2, h3 {
  color: red;
}
o lo que es lo mismo
h1 {color: red;}
h2 {color: red;}
h3 {color: red;}
```

- La *propiedad*, que en este caso sería `color`, especifica qué aspecto se va a cambiar. En este ejemplo la propiedad cambiada será el color. Las propiedades que se desean modificar en una CSS para un mismo selector pueden agruparse, pero será necesario separar cada una de ellas con un punto y coma.

```
p {text-align:center;color:red}
```

Normalmente se describe una propiedad por línea, de la siguiente manera:

```
h1 {
  padding-left: 11em;
  font-family: Georgia, "Times New Roman",Times, serif;
  color: red;
  background-color: #d8da3d;
}
```

- El *valor*, en este caso `red`, establece el valor de la propiedad. Es importante recordar que si el valor está formado por más de una palabra, hay que ponerlo entre comillas.

```
p {font-family: "sans serif";}
```

3.4.Arquitectura de Aplicaciones J2EE.

Java 2 Enterprise Edition (J2EE) es una arquitectura multicapa para implementar aplicaciones de tipo empresarial y aplicaciones basadas en la Web. Esta tecnología soporta una gran variedad de tipos de aplicaciones desde aplicaciones Web de gran escala a pequeñas aplicaciones cliente-servidor. El objetivo principal de la tecnología J2EE es crear un simple modelo de desarrollo para aplicaciones empresariales utilizando componentes basados en el modelo de aplicación. En este modelo dichos componentes utilizan servicios proporcionados por el contenedor, que de otro modo tendrían que estar incorporados en el código de la aplicación. Observa que esto podría no ser lo ideal para todos los escenarios: por ejemplo, una pequeña aplicación se cubriría mejor utilizando una solución de la tecnología Java de peso ligero (por ejemplo Servlets, JSPs, etc.).

3.4.1. Componentes J2EE

Las aplicaciones J2EE están compuestas de diferentes componentes. Un componente J2EE es una unidad de software funcional auto-contenido que se ensambla dentro de una aplicación J2EE con sus clases de ayuda y ficheros y que se comunica con otros componentes de la aplicación. La especificiación J2EE define los siguientes componentes J2EE:

- Las **Aplicaciones Clientes** y los **Applets** son componentes que se ejecutan en el lado del cliente.
- Los componentes **Java Servlet** la tecnología **JavaServer Pages** son componentes Web que se ejecutan en el lado del servidor.
- Los **Enterprise JavaBeans** (*beans enterprise*) son componentes de negocio que se ejecutan en el servidor de aplicación.

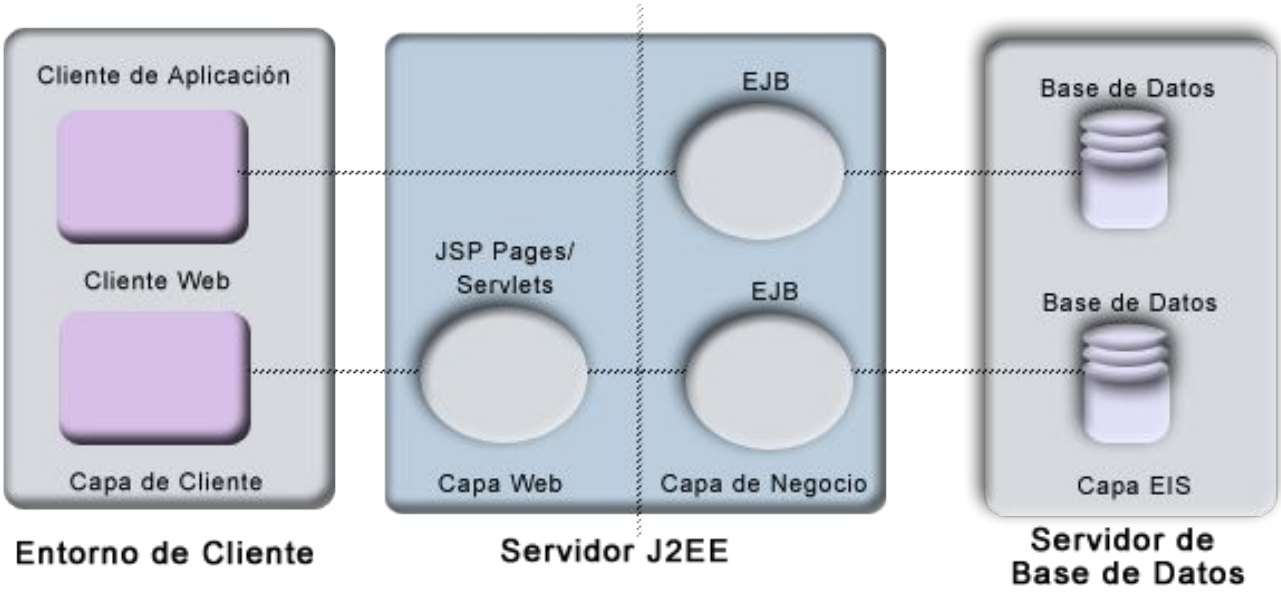

Figura 6: Arquitectura de N capas de una aplicación J2EE

Todos esos componentes se ensamblan en una aplicación J2EE, se verifica que están bien formados y que cumplen la especificación J2EE, y se despliegan en el entorno de producción, donde se ejecutan y son controlados por el servidor de aplicaciones J2EE.

Además de estos componentes principales, J2EE incluye servicios estándar y tecnologías de soporte como:

- **Java Database Connectivity** (JDBC) tecnología que proporciona acceso a sistemas de bases de datos relacionales.
- **Java Transaction API (JTA)** o **Java Transaction Service (JTS)** proporciona soporte para transacciones a los componentes J2EE.
- **Java Messaging Service (JMS)** para comunicación asíncrona entre componentes J2EE.
- **Java Naming** y **Directory Interface (JNDI)** proporcionan accesos a nombres y directorios.

Hay que indicar que todos los componentes J2EE están escritos en lenguaje Java.

3.4.2. Clientes J2EE

Normalmente hay dos tipos de clientes J2EE: clientes Web y aplicaciones cliente como vimos en la figura anterior.

Un cliente Web consta de dos partes, páginas Web dinámicas que contienen distintos tipos de lenguajes de marcas (HTML, XML, y otros), que son generados por los componentes Web que se ejecutan en la capa Web, y un navegador Web, que dibuja las páginas recibidas desde el servidor. Otra categoría de clientes web son los conocidos como clientes *thin* (pequeños). Este tipo de pequeños clientes normalmente no hacen cosas como consultas a bases de datos o ejecutar complejas reglas de negocio. Cuando se utilizan clientes pequeños, las operaciones de peso pesado como estás las manejan los *beans enterprise* que se ejecutan en el servidor J2EE donde pueden tratar con la seguridad, los servicios y el rendimiento de las tecnologías del lado del servidor J2EE.

Una página Web recibida desde la capa del cliente puede incluir un applet embebido. Un applet es una pequeña aplicación cliente escrita en Java que se ejecuta en la máquina virtual Java instalada en el navegador Web. Sin embargo, los sistemas cliente necesitarán el Plug-in Java y posiblemente un fichero de política de seguridad para poder ejecutar con éxito los applets en el navegador Web. Normalmente los componentes Web son el API preferido para crear programas clientes Web porque no necesitan plug-ins ni ficheros de política de seguridad en los sistemas clientes. Además esto permite un diseño más claro y modular de la aplicación porque proporciona un significado a la separación de la lógica de la aplicación del diseño de la página Web.

Una aplicación cliente se ejecuta sobre una máquina cliente y proporciona una forma para que los usuarios puedan manejar tareas que requieren un interfaz de usuario más rico que el que puede proporcionar un lenguaje de marcas. Normalmente tienen un interfaz gráfico de usuario (GUI) creado con los APIs **Swing** o **Abstract Window Toolkit (AWT)**. Las aplicaciones cliente acceden directamente a los beans enterprise que se ejecutan en la capa de negocio. Pero si se necesita un cliente Web pueden abrir una conexión HTTP para establecer comunicación con un servlet que se ejecute en la capa Web. También es posible utilizar un interfaz por línea de comandos.

3.4.3.Componentes Web

Los componentes Web de J2EE pueden ser servlets o páginas JSP. Los servlets son clases Java que procesan dinámicamente las peticiones y construyen las respuestas. Las páginas JSP son documentos basados-en-texto que se ejecutan como servlets pero permiten una aproximación más natural para crear contenido estático. Las páginas HTML y los applets se juntan con los componentes Web durante el ensamble de la aplicación, pero la especificación J2EE no los considera como componentes J2EE. De forma similar, las clases de utilidades del

lado del servidor también se unen a los componentes Web como páginas HTML, pero tampoco se consideran como componentes J2EE. En la Figura 7 podemos ver la comunicación entre componentes Web.

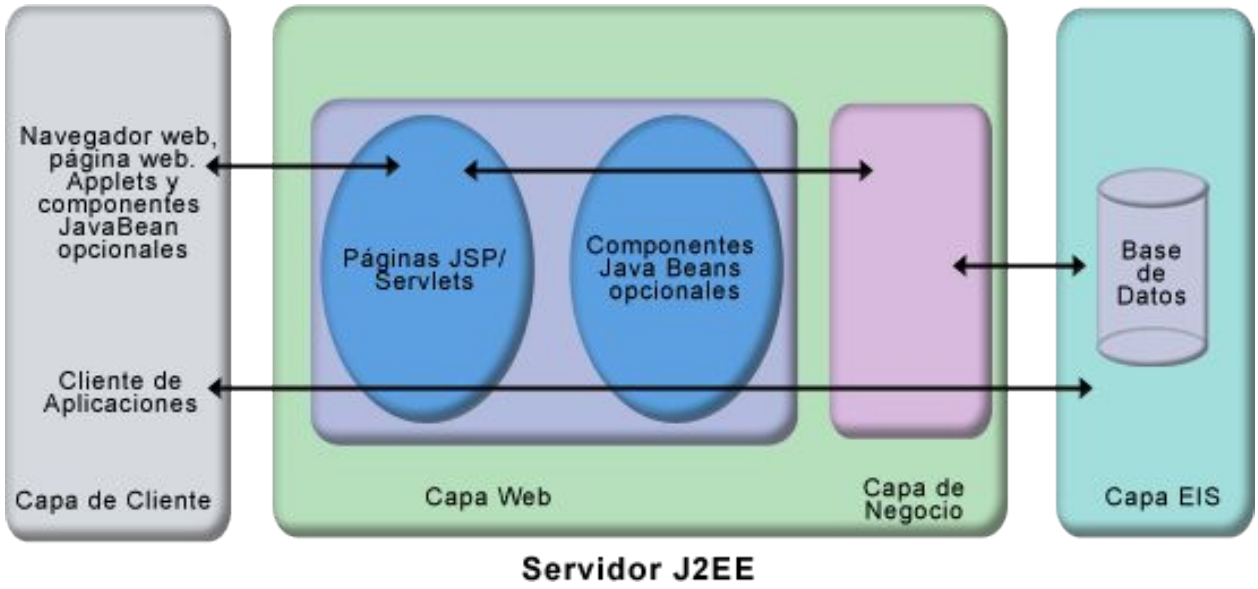

Figura 7: Comunicación entre los componentes web

La capa Web podría incluir componentes JavaBeans para manejar la entrar del usuario y enviar esta entrada a los *beans enterprise* que se ejecutan en la capa de negocio para su procesamiento según hemos visto en la figura anterior.

3.4.4. Componentes de Negocio

El código de negocio, que es la lógica que resuelve o cumple las necesidades de un negocio particular, como la banca, la venta, o la financiación, se maneja mediante *beans enterprise* que se ejecutan en la capa de negocio.

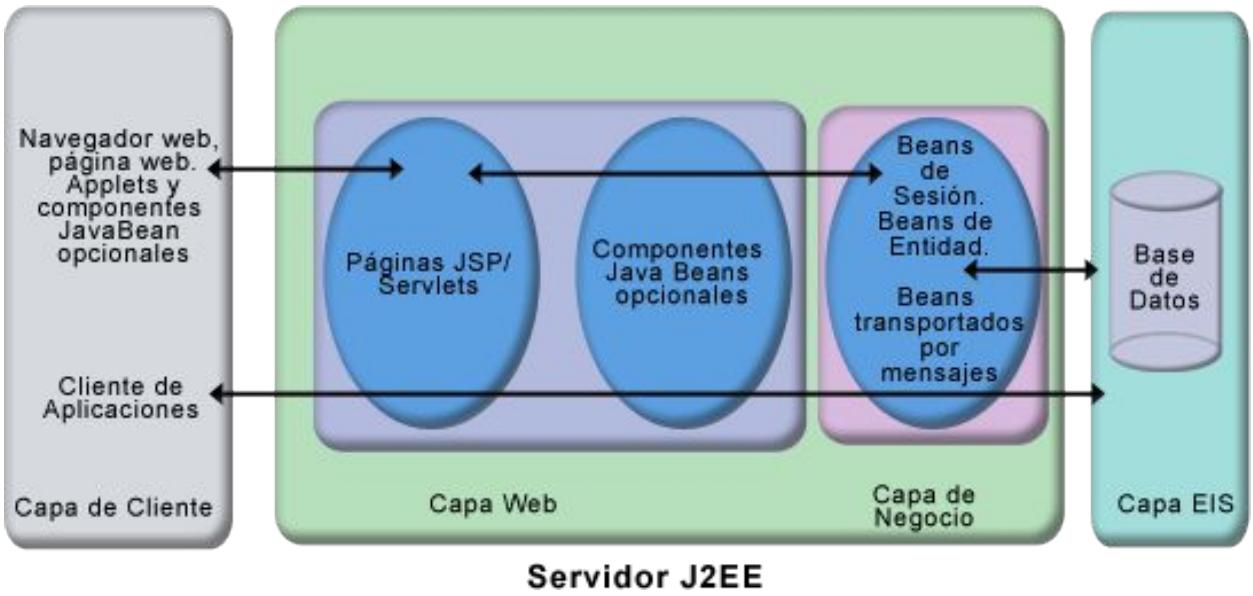

Figura 8: Comunicación entre los componentes de negocio.

La Figura 8 muestra la comunicación entre los componentes de negocio, donde un *bean enterprise* recibe datos de los programas clientes, los procesa (si es necesario), y los envía a la capa del sistema de información empresarial para su almacenamiento. Un bean enteprise también recupera datos desde el almacenamiento, los procesa (si es necesario), y los envía de vuelta al programa cliente.

Hay tres tipos de *beans enterprise*: beans de sesión (con o sin estado), beans de entidad (manejados por el bean o por el contenedor) y beans dirigidos a mensaje. Un bean de sesión representa una conversación temporal con un cliente. Cuando el cliente finaliza su ejecución, el bean de sesión y sus datos desaparecen. Por el contrario, un bean de entidad representa datos persistentes almacenados en una fila de una tabla/relación de una base de datos. Si el cliente se termina o si se apaga el servidor, los servicios subyacentes se aseguran de grabar el bean. Un bean dirigido-a-mensaje combina las características de un bean de sesión y de un oyente de *Java Message Service* (JMS), permitiendo que un componente de negocio reciba asíncronamente mensajes JMS.

Nota:
La especificación J2EE no considera como componentes J2EE a los Java Beans ya que son diferentes de los Beans Enterprise. La arquitectura de componentes JavaBeans se pueden utilizar tanto en la capa de cliente como de servidor para manejar la comunicación entre una aplicación cliente o un applet y los componentes que se ejecutan en el servidor J2EE o entre los componentes del servidor y una base de datos, mientras que los componentes Enterprise JavaBeans sólo se utilizan en la capa de negocio como parte de una capa de servidor. Los JavaBeans tienen variables de ejemplar y métodos accesores y mutadores para acceder a las propiedades del bean o digamos, acceso a los datos en las variables de ejemplar lo que simplifica el diseño y la implementación de los componentes JavaBeans.

3.4.4.1. La Capa del Sistema de Información Empresarial

La capa del sistema de información empresarial maneja el software del sistema de información empresarial e incluye la infraestructura del sistema así como los *plannings* de recursos (ERP), procesamiento de transacciones a *mainframes*, sistemas de bases de datos y otros sistemas de información legales (personalizados). Los componentes de aplicaciones J2EE podrían necesitar acceder a estos sistemas de información empresariales para conectividad con sus bases de datos.

3.4.5. Contenedores J2EE

Los contenedores J2EE proporcionan acceso a los servicios subyancentes del entorno del Servidor J2EE mediante contenedores para diferentes tipos de componentes. Tradicionalmente, los desarrolladores de aplicaciones tenían que escribir código para el manejo de transacciones, manejo del estado, multi-threads, almacenamiento de recursos, etc. Ahora el contenedor J2EE proporciona estos servicios permitiendo que te puedas concentrar en resolver los problemas de negocio.

Los contenedores son el interfaz entre un componente y la funcionalidad de bajo nivel específica-de-la-plataforma que soporta el componente. Por ejemplo, antes de poder ejecutar un componente Web, un *bean enterprise* o un componente de una aplicación cliente, debe ensamblarse dentro de una aplicación J2EE y desplegarse dentro de su contenedor.

El proceso de ensamble implica especificar las configuraciones del servidor para cada componente de la aplicación J2EE y para la propia aplicación J2EE. Estas configuraciones personalizan el soporte subyacente proporcionado por el servidor J2EE, que incluye servicios como JNI, JNDI, seguridad, control de transacciones, etc.

El servidor J2EE proporciona contenedores para **Enterprise JavaBeans** (EJB) y para componentes **Web**. El contenedor EJB maneja la ejecución de los *beans enterprise* de las

aplicaciones J2EE, mientras que el contenedor Web maneja la ejecución de las páginas JSP y los componentes servlets de la aplicación J2EE. Otros contenedores distintos a estos son el contenedor de aplicaciones clientes y el contenedor de applets, que no son parte del servidor J2EE porque residen en la máquina del cliente.

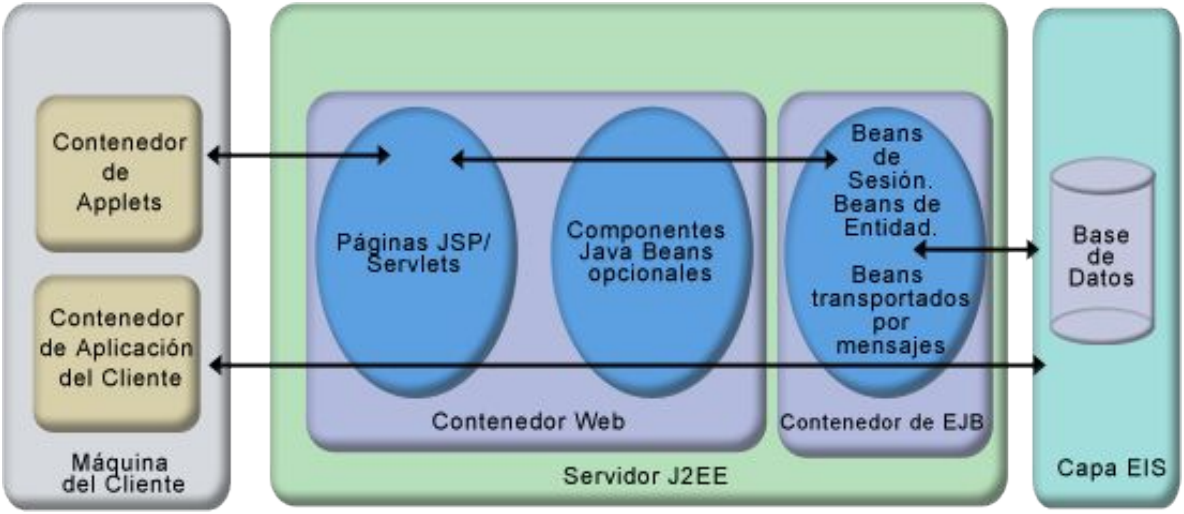

Figura 9: Contenedores J2EE

Una contenedor de aplicaciones cliente maneja la ejecución de los componentes de la aplicación cliente mientras que un contenedor de Applets maneja la ejecución de los applets. Normalmente están en el JRE (*Java Runtime Environment*) y el navegador Web compatible con Java, respectivamente.

3.4.6. Empaquetado

Para poder desplegar una aplicación J2EE, después de desarrollar los diferentes componentes, se empaqueta en ficheros de archivo especiales que contienen los ficheros de las clases relevantes y los descriptores de despliegue XML. Estos descriptores de despliegue contienen información específica de capa componente empaquetado y son un mecanismo para configurar el comportamiento de la aplicación en el momento del ensamble o del despliegue. Estos se empaquetan en diferentes tipos de archivos según los distintos componentes.

Los componentes Web se empaquetan en un archivo Web (`.war`) que contiene los servlets, las páginas JSP y los componentes estáticos como las páginas HTML y las imágenes. El fichero `.war` contiene clases y ficheros utilizados en la capa Web junto con un descriptor de despliegue de componentes Web.

Los componentes de negocio se empaquetan en un archivo Java (`.jar`) que contiene los descriptores de despliegue EJB, los ficheros del interface remoto y del objeto junto con ficheros de ayuda requeridos por el componente EJB.

Los ficheros de clases del lado del cliente y los descriptores de despliegue se empaquetan en un fichero Java (`.jar`) que crea la aplicación cliente.

Una aplicación J2EE se empaqueta en un archivo *enterprise* (`.ear`) que contiene toda la aplicación junto con el descriptor de despliegue que proporciona información sobre la aplicación y sus componentes.

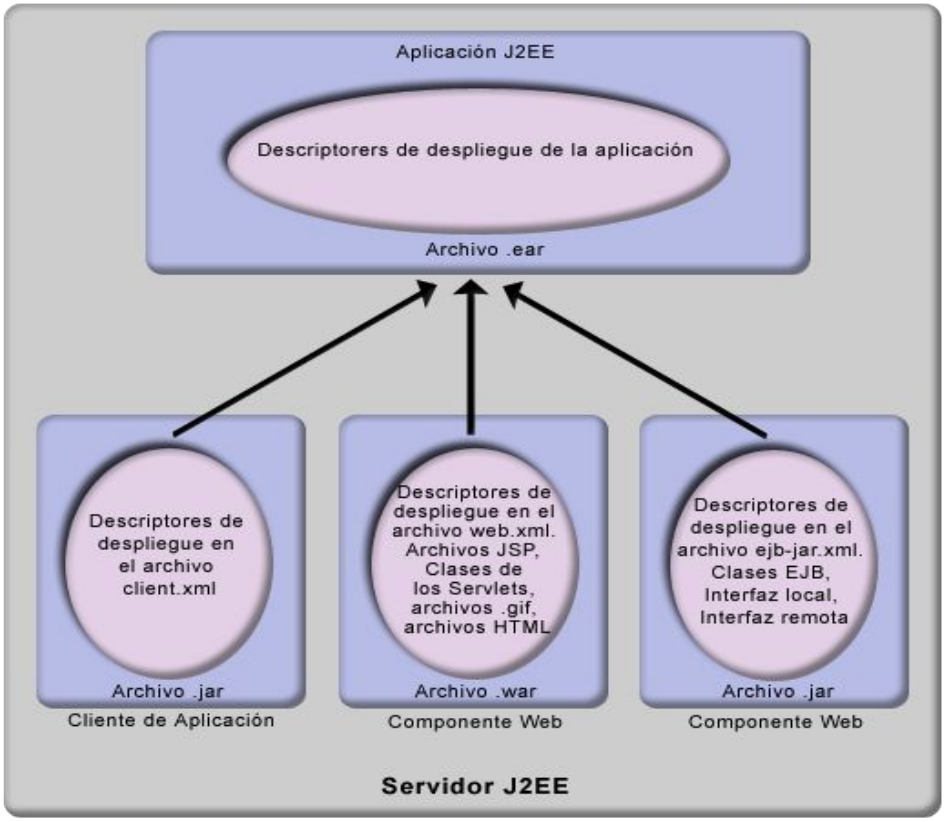

Figura 10: Empaquetado en las aplicaciones J2EE

3.4.7. Roles de la Plataforma J2EE

La construcción de los diferentes componentes de una aplicación J2EE implica a varios roles en el desarrollo, despliegue y control de una aplicación empresarial.

- El proveedor de componentes de aplicación desarrolla componentes J2EE reutilizables, que pueden ser componentes Web, beans enterprise, applets, o aplicaciones clientes para utilizar en aplicaciones J2EE.
- El ensamblador de aplicaciones toma todos los bloques de los diferentes proveedores de componentes y los combina en aplicaciones J2EE.
- El desarrollador es el responsable de la instalación/despliegue de los componentes en un entorno o servidor J2EE.
- El administrador del sistema es el responsable de configurar y administrar los sistemas informáticos en una empresa.
- El proveedor de herramientas es un vendedor utilizado para desplegar, empaquetar y desplegar aplicaciones J2EE.

Nota:
Todos los roles mencionados se pueden asignar a personas u organizaciones.

3.4.8. La Arquitectura Distribuida en J2EE

Todas las aplicaciones J2EE implementan una arquitectura distribuida. En ésta un objeto está asociado con un nombre, donde los nombres los proporciona un servicio de nombres,

notificando a distintos componentes y resolviendo las referencias de clientes para estos componentes de servicio como se muestra en la siguiente figura:

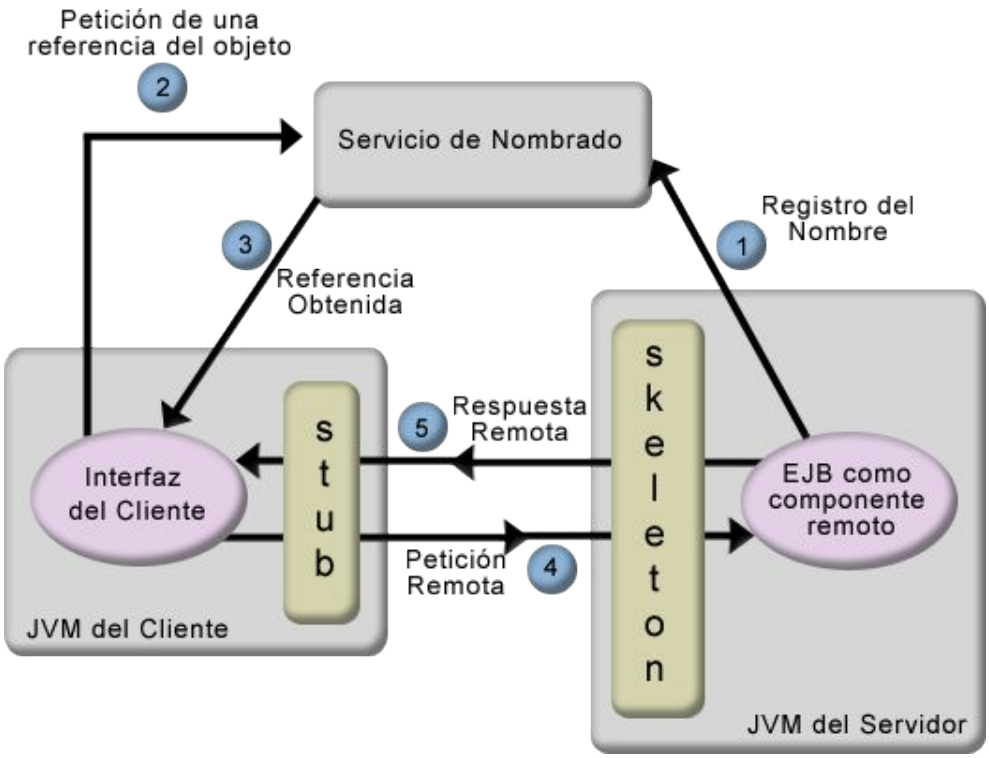

Figura 11: Esquema de la arquitectura distribuida J2EE

Como resultado de esto, las referencias de objetos se obtienen buscando un objeto por su nombre notificado, una vez encontrado, se obtiene la referencia, y se llevan a cabo las operaciones necesarias sobre ese objeto utilizando los servicios del host. Un objeto remoto notifica su disponibilidad en el servicio de nombres utilizando un nombre lógico y el servicio de nombres lo traduce a la localización física del objeto en el entorno J2EE. Una vez que la petición del cliente obtiene una referencia a un componente remoto, puede enviarle peticiones. El sistema de ejecución maneja la comunicación distribuida entre objetos remotos, lo que incluye la serialización y des-serialización de parámetros.

Nota:
*Algunos de los sistemas de nombres utilizados en los sistemas distribuidos son **RMI** (sólo para implementaciones Java), CORBA, LDAP, DNS, NIS.*
El servidor de aplicaciones JBOSS utiliza RMI como su servicio de nombres.

3.4.9. La Arquitectura Java Naming Directory Interface (JNDI)

J2EE utiliza el API JNDI para acceder genéricamente a servicios de nombrado y directorio utilizando la tecnología Java. El API JNDI reside entre la aplicación y el servicio de nombres y hace que el servicio de nombres subyacente sea transparente para los componentes de la aplicación.

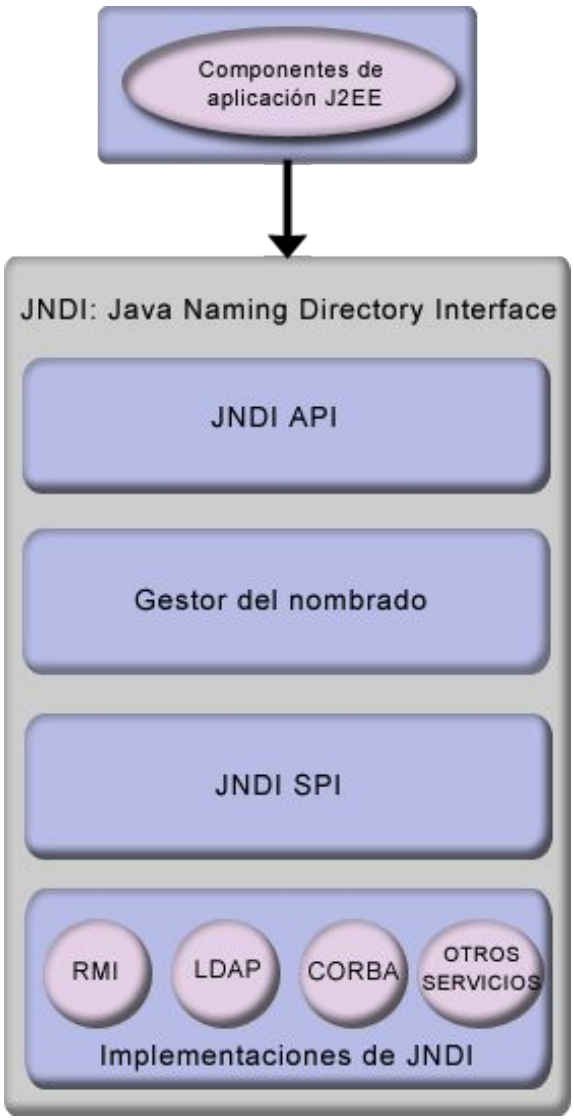

Figura 12: Java Naming Directory Interface

Un cliente puede buscar referencias a componentes EJB u otros recursos en un servicio de nombres como el mencionado arriba. El código del cliente no se modifica, sin importar el servicio de nombres que se esté utilizando o en qué tecnología esté basado, y esto no crea ninguna diferencia en el modo en que los clientes localizan los objetos remotos mediante el API JNDI.

3.5.Apache Cocoon.

Cocoon es un marco de desarrollo en el lado del servidor basado en Java que permite la publicación dinámica de contenido XML usando XSLT (*XML Stylesheet Language Transformation*). Basándose en XML para describir el contenido y en XSLT como medio para transformar ese contenido en múltiples formatos, Cocoon provee una plataforma para construir aplicaciones con una gran separación entre la lógica, el contenido y la presentación.

Cocoon usa el concepto de *pipeline* para describir el proceso de la publicación del contenido hacia el mundo web. Se incluye una gran variedad de componentes reusables, que

pueden ser configurados para producir comportamientos complejos sin necesidad de un gran desarrollo de clases de Java, es decir, sin la necesidad de programación en Java tal y como pudiera suceder en tecnologías como las JSP. Como ejemplo, usando XML y XSLT en solitario, se puede utilizar a Cocoon para:

- Servir páginas estáticas y respuestas generadas dinámicamente.
- Realizar transformaciones XSLT tanto simples como en varias etapas.
- Pasar parámetros de forma dinámica a las transformaciones XSLT.
- Generar una gran cantidad de formatos de salida tales como XML, HTML, PNG, JPEG, SVG y PDF.

Todo esto muestra el potencial que poseen los elementos ya implementados de Cocoon en conjunción con XML y XSLT.

3.5.1.Cocoon 1 y Cocoon 2

Cocoon es un proyecto de fuentes abiertas que ha sido desarrollado como parte del esfuerzo de la Apache Software Foundation. Cocoon 2 es una revisión completa de la aplicación Cocoon original.

La intención de Cocoon 2 fue la de aprender de los problemas y errores cometidos con Cocoon 1 y usar este conocimiento para crear una plataforma más eficiente y escalable. En particular, Cocoon 1 se basaba en la API de DOM (Document Object Model) para pasar los datos XML entre los componentes. El modelo DOM es ineficiente en ese aspecto debido a que un árbol DOM típico puede consumir varias veces la memoria que ocuparía el documento XML original. Esto limitaba seriamente la escalabilidad de Cocoon. Cocoon 2 se construyó en torno a la API de SAX, que es más ligera en términos de la manipulación de los datos XML.

Otra diferencia fundamental entre las dos versiones de Cocoon se centra en la gestión de la aplicación. En Cocoon 1, cada documento XML declaraba cómo debía ser procesado incluyendo instrucciones de proceso de Cocoon. Esto ataba a los documentos a un procesamiento específico, lo que redundaba en una restricción de la flexibilidad para reutilizar el contenido de formas diferentes. Cocoon 2 realiza la gestión del procesamiento en un archivo de configuración conocido como el *sitemap*. Esto separa la lógica del procesamiento del contenido mismo, lo que se convierte en la separación de conceptos entre contenido, lógica y presentación anteriormente citada.

3.5.2.La arquitectura de Cocoon 2

Esta sección introduce los principios de la arquitectura de Cocoon 2. Se cubrirán varios conceptos clave:

1) El procesamiento de un documento XML se puede dividir en varios pasos discretos. La combinación entre estos pasos describe una *pipeline*. Una *pipeline* consiste en una entrada, un procesamiento de los datos y una salida. Cocoon 2 usa eventos SAX para la comunicación entre cada paso del procesamiento.
2) Cada estado de esta *pipeline* se puede modelar con una clase particular de componente. Por ejemplo, un generador para producir una entrada y un serializador para producir la salida. Cocoon define varios componentes diferentes y provee varias implementaciones para cada uno de ellos. Estos tipos de componentes y sus implementaciones más útiles se tratarán posteriormente en la sección Componentes de las *Pipelines*.

3) La respuesta a las peticiones del usuarios incluyen la identificación de la *pipeline* correcta para servir la petición y la construcción de la *pipeline* para producir la respuesta para el usuario.

En la siguiente figura podemos observar un esquema de la arquitectura de publicación de de Cocoon. En ella se muestra el flujo que sigue una petición desde que llega al servlet hasta que devuelve la respuesta solicitada por el cliente.

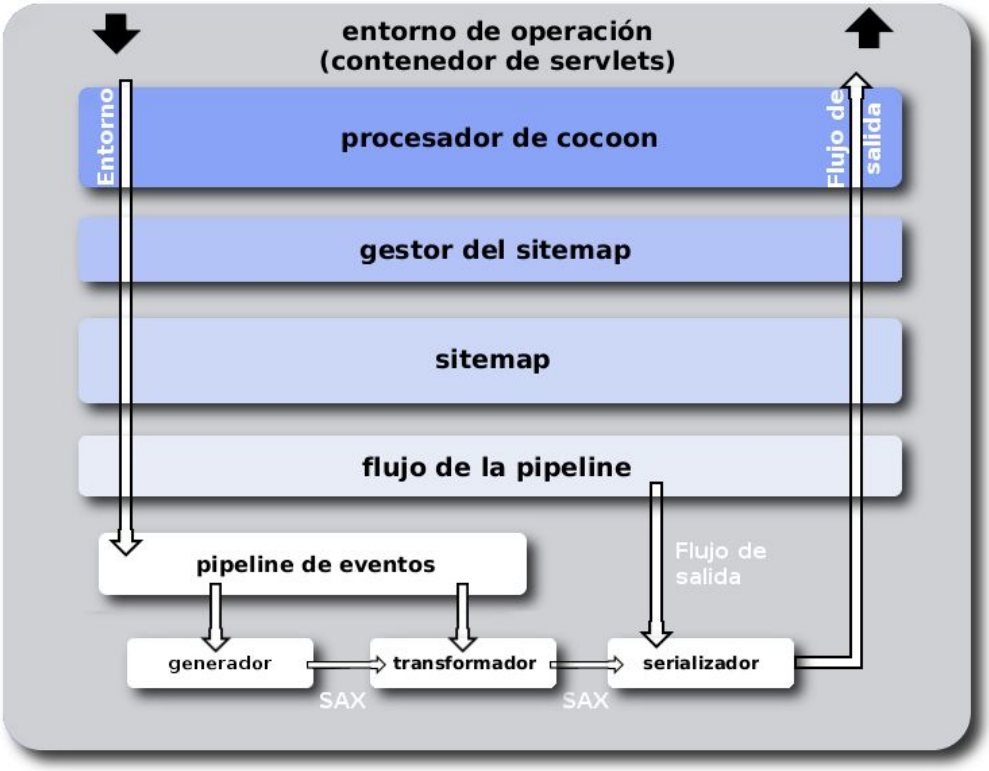

Figura 13: Arquitectura de publicación de Cocoon

Aunque este esquema refleja todo el marco de publicación de Cocoon, no entraremos con todo detalle en su explicación, aunque sí en lo necesario para tener una mejor comprensión de la arquitectura y funcionamiento del Gestor de Contenidos Lenya. En posteriores secciones de este capítulo se hablará de los componentes que se observan en la figura y se introducirá el concepto de *sitemap*, que permite conectar los componentes dentro de las *pipelines* para realizar todo el procesamiento de la información.

3.5.2.1.El modelo de las *pipelines.*

Hay un concepto simple que conforma la clave de la arquitectura de Cocoon 2: las *pipelines*. Una *pipeline* consiste en una serie de datos de entrada que se siguen de una serie de etapas de procesamiento. Cada etapa del procesamiento acepta la salida de la etapa anterior como entrada hasta que se alcanza el final de la *pipeline* y se produce la salida.

Las *pipelines* son un concepto simple. Es fácil descomponer cualquier tarea compleja de procesamiento en etapas más pequeñas que se pueden organizar dentro de una *pipeline*. Este concepto es bastante familiar a los usuarios de UNIX que están habituados a combinar aplicaciones pequeñas y de propósito general (por ejemplo `find`, `grep` y `sort`) para realizar tareas más específicas.

Esto puede ilustrar otro beneficio potencial de las *pipelines*, debido a que cada etapa de procesamiento tiene una conducta bien definida respecto a unas entradas y salidas fijadas, se pueden crear componentes generales y reusables para ellas. Esta reutilización permite la construcción de aplicaciones con una pequeña sobrecarga de programación.

3.5.2.2.Componentes de las *Pipelines.*

Como hemos mencionado anteriormente, Cocoon incluye varios componentes generales que se pueden interconectar para la construcción de las aplicaciones. Estos componentes se pueden agrupar en varios grupos, dependiendo del papel que jueguen en una *pipeline*:

- Entrada a las *pipelines*: Generadores y *Readers*.
- Etapas de procesamiento: Transformadores y Acciones.
- Salida de las *pipelines*: Serializadores.
- Procesamiento condicional: *Matchers* y Selectores.

Una *pipeline* en Cocoon generalmente consiste, al menos, de un generador y un serializador, pero puede consistir en cualquier número de pasos de procesamiento. Los datos se pasan a través de una *pipeline* como eventos SAX.

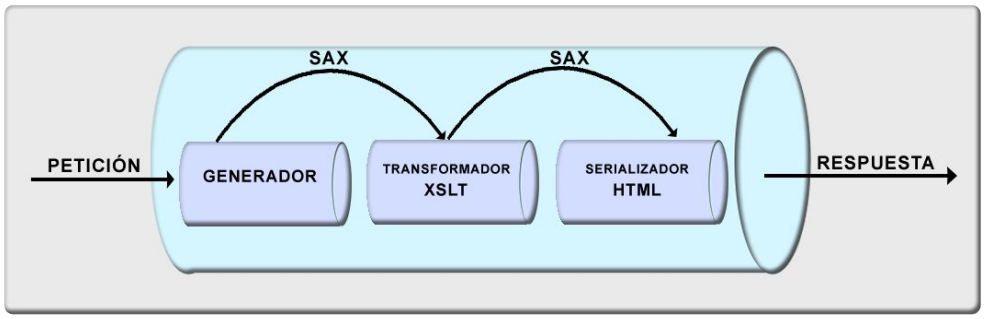

Figura 14: Las pipelines generan y consumen eventos SAX

Los cuatros siguientes puntos muestran una visión algo más detallada de estos componentes.

3.5.2.2.1.Entradas: Generadores y Readers.

Los Generadores son los responsables de leer la fuente de datos (por ejemplo, un archivo) y pasar esos datos a la *pipeline* como una serie de eventos SAX. El generador más simple, por lo tanto, es un analizador SAX. Generalmente, cualquier fuente de datos que se pueda mapear en una serie de eventos SAX se puede convertir en la base para un generador.

Hay varios generadores disponibles en Cocoon. Los más habituales son:

- FileGenerator: lee archivos XML del sistema de archivos o de internet.
- HTMLGenerator: lee archivos HTML del sistema de archivos o de internet.
- DirectoryGenerator: lee el sistema de archivos para proveer listados de directorios.

Los *Readers* son un caso especial del modelo de *pipeline* de Cocoon ya que no se trata de componentes que acceden a datos XML. Los *Readers* simplemente acceden a un recurso externo y lo copian directamente en la respuesta. Se suelen usar para servir archivos estáticos

tales como imágenes u hojas de estilo CSS. Se pueden ver como *pipelines* autocontenidas: generan los datos de entrada y los serializan hacia la respuesta.

3.5.2.2.2.Procesamiento: transformadores y acciones

Los transformadores son las principales etapas de procesamiento en una *pipeline* de Cocoon. Aceptan eventos SAX como entrada, los procesan y pasan los resultados hacia el resto de la *pipeline* como eventos SAX. Una forma de ver el transformador es como un componente que modifica el flujo de los eventos SAX cuando pasan a través de él. En este punto, son parecidos a los filtros SAX.

El transformador más ampliamente usado es el XSLT. Pasa su entrada a un procesador XSLT que realiza la transformación. El resultado de la transformación se devuelve hacia la *pipeline* en forma de eventos SAX.

Las acciones son la forma de añadir un comportamiento dinámico adicional a la *pipeline* y, en general, se hacen a medida para aplicaciones particulares. Se ejecutan durante la configuración de la *pipeline*. Su propósito es el de ejecutar el código necesario para que la *pipeline* pueda funcionar. Por ejemplo, la acción puede sacar información de una base de datos para poblar el documento que consume el generador. Su ejecución puede tener éxito o no. Si la acción falla, el segmento de *pipeline* definido dentro de la acción no se ejecuta. De cualquier forma, Cocoon viene con algunas acciones genéricas de serie, por ejemplo, para manejar interacciones con bases de datos, validación de *forms*, envío de correo electrónico, etc.

3.5.2.2.3.Salida: serializadores

Los serializadores son los puntos finales de las *pipelines* de Cocoon. Son los responsables de coger un flujo de eventos SAX producidos directamente desde un generador (en la *pipeline* más pequeña posible) o de una etapa previa del procesamiento (como pudiera ser un transformador) y renderizarlos en el formato adecuado para la respuesta. El formato específico depende del serializador que se esté utilizando.

El serializador más simple es el XML, que devuelve los eventos SAX en forma de un documento XML. Otros serializadores pueden producir HTML, texto plano, PDFs e incluso imágenes. Esos serializadores esperan que el flujo de eventos SAX se ajuste a un vocabulario XML particular:

- Serializador HTML: convierte XHTML en HTML válido.
- Serializador SVG: convierte SVG en imágenes JPEG o PNG.
- Serializador PDF: convierte XSL-FO en un documento PDF.

Esta habilidad para tomar el contenido XML, procesarlo y servirlo en varios formatos es el poder real de Cocoon.

3.5.2.2.4.Condiciones: matchers y selectores

Cualquier *pipeline* no trivial abarca, generalmente, algunas secciones condicionales. Por ejemplo, el número exacto de etapas de procesamiento puede depender de factores tales como los parámetros de la petición, el navegador del usuario, etc.

Los *matchers* son los componentes condicionales más simples y son equivalentes a las cláusulas `if`. Si se cumple una condición, entonces se evalúa una *pipeline* en particular o una sección de ella.

El segundo tipo de componente condicional es el selector, que es similar a una cláusula `if-then-else`. Los Selectores se usan cuando se dispone de una de varias entre varias

opciones y se suelen utilizar para crear secciones condicionales dentro de una *pipeline*, mientras que los *matchers* se utilizan para comprobar si se debería entrar en una *pipeline* en particular.

Hay varias implementaciones de cada uno de esos componentes. Todas siguen un patrón común al comprobar algún aspecto de la petición (como el nombre de la máquina, el agente usuario, la URL, etc.) o de la sesión del usuario. La comprobación se puede realizar contra expresiones regulares o contra una *wildcard* (que permite aceptar como válido un patrón genérico), mientras que los selectores generalmente enumeran todos los valores posibles.

3.5.2.3.Haciendo funcionar a las *pipelines*

Ahora que se han introducido los componentes que se usan generalmente para construir las *pipelines* de Cocoon, es importante poner a las *pipelines* dentro de su contexto para ver cómo trabajan. Una descripción del ciclo lógico de la recepción de peticiones y de la devolución de las respuestas se puede resumir como sigue.

1) Aceptar una petición del usuario.
2) Determinar la *pipeline* que se debería usar para interpretar esta petición (usando un matcher) y producir una respuesta.
3) Construir la *pipeline* con los componentes disponibles.
4) Introducir los datos en la *pipeline* para servir la petición.
5) Devolver la respuesta generada por la *pipeline* al usuario, posiblemente guardando en caché los resultados para un uso posterior.

Este es el ciclo básico petición-respuesta que Cocoon usa para publicar los datos XML hacia el mundo Web. Para gestionar este ciclo, Cocoon provee un archivo de configuración XML llamado *sitemap*, que se detallará en las siguientes secciones.

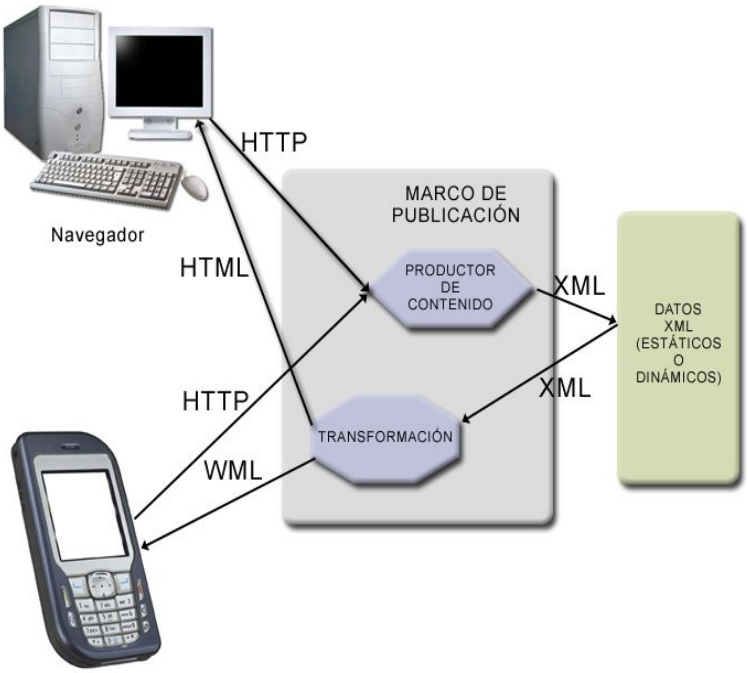

Figura 15: Esquema simplificado de la gestión de una petición

Esta sección sólo ha visitado los componentes más usados para construir las *pipelines* de Cocoon. Estos componentes reusables se benefician directamente del modelo de *pipeline* y su uso permite la creación de un procesamiento sofisticado con muy poca o ninguna programación aparte de la creación de las hojas de estilo XSLT.

3.5.3.Conceptos básicos del *sitemap*

3.5.3.1.Responsabilidades del *sitemap*

El *sitemap* es el punto central a partir del cual se gestiona un sitio Web basado en Cocoon. Se encarga de dos funciones:

- Es el lugar en el que se declaran los componentes antes de ser usados en las *pipelines*.
- Es el lugar en el que se definen las *pipelines* usando los componentes declarados.

Esta sección trata de la primera de esas responsabilidades e introduce la estructura básica del *sitemap*. La configuración de las *pipelines*, junto a algunos ejemplos, se cubrirá en la siguiente sección, “El *sitemap*: definición de las *pipelines*”.

3.5.3.2.Estructura del *sitemap*

El *sitemap*, como ya hemos mencionado, es un archivo de configuración XML y pro consiguiente tiene una estructura bien definida. El *sitemap* por defecto en Cocoon, `sitemap.xmap`, se puede encontrar en el directorio de la aplicación web de Cocoon, `$CATALINA_HOME/webapps/cocoon/sitemap.xmap` (suponiendo una instalación sobre Tomcat).

Un *sitemap* está estructurado de acuerdo a la forma en que aparece en el siguiente fragmento XML. Hay que notar que hay un espacio de nombres específico para el *sitemap*, `http://apache.org/cocoon/sitemap/1.0`, que se usa para identificar los elementos del mismo. También hay que comentar que el *sitemap* está dividido en dos grandes secciones, `map:components` y `map:pipelines`, lo cual refleja las dos responsabilidades del archivo.

Listado 3.2: Estructura básica de un archivo xmap.

```
<map:sitemap
   xmlns:map="http://apache.org/cocoon/sitemap/1.0">

   <map:components>
     <!-- component declarations -->
     <map:generators/>
     <map:readers/>
     <map:transformers/>
     <map:actions/>
     <map:serializers/>
     <map:actions/>
     <map:matchers/>
     <map:selectors/>
   </map:components>

   <map:pipelines>
     <!-- pipeline definitions -->
   </map:pipelines>

</map:sitemap>
```

Las declaraciones para cada tipo de componente se agrupan juntas dentro de un elemento específico. Por ejemplo, todas las declaraciones de generadores se pueden encontrar dentro del elemento `map:generators`.

3.5.3.3.Declarando los componentes del *sitemap*

Los componentes se declaran dentro del *sitemap* de la forma general que se muestra en el siguiente ejemplo. Hay algunos puntos que se deben tener en cuenta:

- El *Component-type* es el nombre de un tipo específico de componente, por ejemplo, el elemento generador contiene declaraciones de generadores.
- Cada componente debe tener un único atributo *name*. Los nombres se usan para referenciar a los componentes en cualquier parte del *sitemap*.
- Cada componente debe identificar su implementación. Es posible tener más de una instancia de un componente compartiendo la misma implementación, pero declarada con diferentes nombres.
- Se debe identificar un componente por defecto. Se usará cuando no se nombre específicamente a una instancia del componente.
- El *sitemap* puede pasar parámetros a los componentes. Es posible, por lo tanto, tener varias instancias del mismo componente pero con diferentes parámetros. Los elementos *parameter* son específicos a cada componente.

Listado 3.3: *Component-types*

```
<map:component-types default="component-name">

   <map:component-type
     name="component-name"
     src="implementation">

     <!-- component-type specific parameters -->

   </map:component-type>

</map:component-types>
```

Para ejemplos específicos de declaraciones de componentes, se remite a la primera sección del *sitemap* por defecto de Cocoon `$CATALINA_HOME/webapps/cocoon/sitemap.xmap`.

3.5.3.4.Capacidad de ampliación

El marco de desarrollo de Cocoon carga los componentes basándose en las declaraciones del *sitemap* usando las capacidades de carga de clases dinámica de Java. Para ser cargado de forma dinámica y conectado a las pipelines, cada componente debe implementar una interfaz Java específica dependiendo de su tipo. Por ejemplo, todos los generadores deben implementar la interfaz `org.apache.cocoon.generation.Generator`.

El mecanismo de las interfaces para la descripción de cada tipo de componente implica que las capacidades de Cocoon pueden ser extendidas de forma simple creando nuevas implementaciones de esas interfaces y añadiendo las declaraciones apropiadas al *sitemap*.

Consideremos el siguiente escenario: hay una gran cantidad de datos en formato CSV (*Comma Separated Values*) que debe estar disponible tanto en el formato original como en HTML. Una aproximación podría ser la de escribir una aplicación simple que lea los datos CSV y que los convierta en XML. Esto impondría una sobrecarga extra de gestión debido a la

necesidad de convertir los nuevos datos conforme van llegando y almacenarlos tanto en el formato original como en XML.

Una integración más acorde a Cocoon podría ser la de escribir un generador personalizado del estilo `com.mycompany.CSVGenerator`, que sea capaz de analizar los archivos CVS para generar directamente eventos SAX. Esta clase se podría conectar luego en Cocoon declarándola dentro del *sitemap* como sigue:

```
<map:generator name="csv" src="com.mycompany.CSVGenerator"/>
```

Este componente puede usarse ahora como un medio para alimentar de datos CVS a las *pipelines* de Cocoon, donde estos datos pueden ser manipulados, transformados y serializados a múltiples formatos. Los archivos CVS originales también se pueden aprovechar usando un componente *Reader*. Debido a que sólo hay un archivo fuente de datos y a que toda la transformación dinámica se lleva a cabo por Cocoon, hay una menor carga de gestión. Obviamente, este ejemplo se podría aplicar a diferentes tipos de formatos de documentos, y se podría extender para obtener documentos de Sistemas de Gestión de Contenidos, etc.

3.5.3.5.Montado de sub-sitemaps.

Los puntos de montaje de los *sub-sitemaps* permiten paralelizar la carga de gestión del sitio y simplificar el despliegue del portal mediante su construcción a partir de una cascada de *sitemaps*. De esta forma se puede crear una especie de árbol de *sitemaps* que tenga el principal como raíz y la posibilidad de encontrar varios *sub-sitemaps* como nodos y hojas del mismo.

Los *sub-sitemaps* sirven para dos objetivos muy importantes: escalabilidad y simplificación del mantenimiento. Cada uno de estos *sub-sitemaps* es independiente y no afecta a los demás.

El mecanismo para el montaje de los *sub-sitemaps* está basado en la etiqueta *<map:mount ...>*.

```
<map:match pattern="faq/*">
      <map:mount uri-prefix="faq" check-reload="no"
                 src="faq/sitemap.xmap"/>
</map:match>
```

El atributo *src* indica donde se halla el *sub-sitemap*. Si termina en una barra "/", se le añade al final "sitemap.xmap" para localizarlo. En otro caso, se utiliza el valor indicado por el atributo *src*. El atributo *check-reload* se puede usar para determinar si se debe comprobar la fecha del *sub-sitemap* para su recarga. El atributo *uri-prefix* es la parte que debería ser eliminada de la URI de la petición. En el ejemplo anterior, si se pidiera "faq/cocoon", se eliminaría "faq/" de la URI y se le pasaría "cocoon" al *sub-sitemap* que se debe cargar, "faq/sitemap.xmap".

Los componentes de un archivo *sitemap* (generadores, transformadores, etc.), son accesibles desde un *sub-sitemap* por sus nombres. Esto es debido al hecho de que cada *sitemap* tiene su propio *SitemapComponentManager* y a que se estructura en la misma forma de árbol en la que se disponen todos los archivos de *sitemap* de la aplicación. De esta forma, un nodo del árbol siempre sabe cuál es su *SitemapComponentManager* padre y puede preguntarle por cualquier *SitemapComponent* que no conozca.

Hay que tener en cuenta que un *sub-sitemap* siempre debe o bien terminar la petición que ha recibido (mediante la llamada a algún serializador) o bien pasarla hacia un *sitemap* que se encuentre un nivel más abajo. Si la petición se pasa a un *sub-sitemap* que no contiene una *pipeline* que acabe en un serializador, se lanzará una excepción diciendo que ninguna *pipeline* capturó la misma. La única posibilidad dentro de un *sub-sitemap* para realizar una llamada a una *pipeline* de un nivel superior es usar el subprotocolo *cocoon:*, de tal forma que cada subpetición (esto es, una petición a un generador, transformador, etc, realizada desde el *sub-sitemap*) que comienze por *cocoon://* se comenzará a procesar en el *sitemap* raíz

(sitemap.xmap) como si fuera una petición completamente nueva.

3.5.4.El *sitemap*: definiendo las *pipelines*

3.5.4.1.Configuraciones del sitemap

Esta sección detalla algunos ejemplos de configuraciones del *sitemap* que demuestran cómo se pueden combinar los componentes introducidos en las secciones anteriores para realizar el trabajo. El foco estará en la configuración del *sitemap* de Cocoon más que en otros detalles acerca de las transformaciones individuales y los formatos XML.

Los ejemplos asumen que se ha desplegado Cocoon en Tomcat y que se ha creado la siguiente estructura de directorios bajo `$CATALINA_HOME/webapps/cocoon` (a la que nos referiremos como `$COCOON_HOME` a partir de ahora):

`/static`	`Documentos HTML estáticos`
`/content`	`Contenido XML`
`/styles`	`Hojas de estilo CSS`
`/transforms`	`Hojas de estilo XSLT`

3.5.4.2.Declaraciones de componentes

Los siguientes ejemplos de configuración asumen que se han realizado las siguientes declaraciones en el *sitemap*:

Listado 3.4: Declaración de componentes en Cocoon.

```
<map:generators default="file">
   <map:generator name="file"
   src="org.apache.cocoon.generation.FileGenerator"/>
</map:generators>

<map:transformers default="xslt">
   <map:transformer name="xslt"
   src="org.apache.cocoon.transformation.TraxTransformer"/>
</map:transformers>

<map:readers default="resource">
   <map:reader name="resource"
   src="org.apache.cocoon.reading.ResourceReader"/>
</map:readers>

<map:serializers default="html">
   <map:serializer name="xml" mime-type="text/xml"
   src="org.apache.cocoon.serialization.XMLSerializer/>
   <map:serializer name="html" mime-type="text/html"
   src="org.apache.cocoon.serialization.HTMLSerializer/>
   <map:serializer name="svg2png" mime-type=”image/svg”
   src="org.apache.cocoon.serialization.SVGSerializer"/>
   <map:serializer name="fo2pdf" mime-type=”application/pdf”
   src="org.apache.cocoon.serialization.FOPSerializer"/>
</map:serializers>

<map:matchers default="wildcard">
   <map:matcher name="wildcard"
   src="org.apache.cocoon.matching.WildcardURIMatcher"/>
</map:matchers>
```

3.5.4.3.Servir un documento estático

Usar Cocoon para servir documentos estáticos es un buen ejemplo para empezar. La siguiente descripción de una *pipeline* demuestra cómo se consigue esto usando un *Reader*. Obviamente es preferible permitir a un servidor Web ya existente manejar los archivos estáticos, pero esto provee un ejemplo simple para ilustrar el trabajo con las *pipelines*.

Listado 3.5: *Match* para un documento HTML estático.

```
<map:pipelines>

   <map:pipeline>
      <map:match pattern="index.html">
         <map:read src="static/index.html" mime-type="text/html"/>
      </map:match>
   </map:pipeline>

<map:pipelines>
```

En primer lugar, notar que la *pipeline* se define mediante el hijo `map:pipeline` del elemento `map:pipelines`, que debería contener todas las definiciones de éstas.

El *Matcher* se usa para asociar la *pipeline* con una petición usando un patrón de asignación. En este caso, una petición del documento "index.html" (por ejemplo, `http://localhost:8080/cocoon/index.html`) lanzaría esta *pipeline*.

Después se define el procesamiento que se debe realizar cuando se entra en la *pipeline*. Aquí, un *Reader* se utiliza para mandar el archivo `$COCOON_HOME/static/index.html` a el usuario con el tipo mime `text/html`. La localización del archivo es completamente independiente de la ruta URL usada para pedirlo.

3.5.4.4.Uso de *wildcards*

El ejemplo inicial se pude expandir para demostrar el uso de las *wildcards* para capturar fragmentos de la URL pedida. La *pipeline* también se ha extendido un poco para permitir tanto documentos HTML como la aplicación de hojas de estilo CSS. Hacer esto sólo implica añadir un nuevo *matcher* a la *pipeline* existente. Esto funciona igual que si declaramos una nueva *pipeline*, porque Cocoon comprobará ambos patrones.

Listado 3.6: Captura de un patrón.

```
<map:pipeline>

   <map:match pattern="*.css">
      <map:read src="styles/{1}.css" mime-type="text/css"/>
   </map:match>

   <map:match pattern="**.html">
      <map:read src="static/{1}.html" mime-type="text/html"/>
    </map:match>

</map:pipeline>
```

Las *wildcards* permiten dos clases de sustituciones y ambas están ilustradas en el ejemplo anterior. La primera, un asterisco, equivale a cualquier número de caracteres excepto a la barra '/'. Un doble asterisco equivale a cualquier número de caracteres incluyendo a la barra. El texto sustituido por estos patrones está disponible a los demás componentes del *sitemap* y puede

referenciarse como {1}, {2}, {3}, y de ahí en adelante, dependiendo de cuantas wildcards se hayan utilizado.

En el ejemplo anterior, una petición como http://localhost:8080/mysite.css encajará en el patrón CSS y el valor "mysite" será asignado a {1}. El componente *Reader* devolverá $COCOON_HOME/styles/mysite.css al usuario con el tipo mime correcto. Una petición como http://localhost:8080/styles/mysite.css no encajaría en el patrón porque sólo especifica un asterisco.

El patrón HTML revisado de este ejemplo usa un doble asterisco como *wildcard*. Así, una petición de la página http://localhost:8080/help/help.html encajará correctamente en el patrón y se le asignará el valor help/help. El componente *Reader* devolverá $COCOON_HOME/static/help/help.html al usuario.

Deberíamos recalcar que Cocoon procesa los patrones secuencialmente en el orden en que se han definido en el *sitemap*. Cocoon procesa la petición de acuerdo al primer patrón que encaja satisfactoriamente. Por consiguiente, el orden en el *sitemap* es importante y se debe tener cuidado poniendo los patrones más específicos al principio del *sitemap*. Por ejemplo, si queremos definir un patrón de captura para index.html se debería declarar antes que el patrón *.html, o de lo contrario nunca sería utilizado.

3.5.4.5.Realizando una transformación

Al menos se necesitan tres componentes para realizar una transformación: un generador para leer el documento XML, un transformador para realizar la transformación y un serializador para enviar los resultados. Aquí se mostrará como se unen estos elementos para realizar la transformación.

En primer lugar, hay que declarar la *pipeline* y el patrón de captura utilizado para lanzarla:

```
<map:pipe>

  <map:match pattern="content/*.html">
```

Después, se añade un generador para leer los documentos XML del directorio content:

```
      <map:generate src="content/{1}.xml"/>
```

Después, se añade un transformador para transformar el documento XML usando una hoja de estilo específica:

```
      <map:transform src="transforms/content2html.xsl"/>
```

Por último se usa un serializador para convertir los resultados de la transformación en un documento HTML que se devuelve al usuario.

```
      <map:serialize type="html"/>

   </map:match>

</map:pipe>
```

La petición de la URL http://localhost:8080/content/document.html lanza esta *pipeline* y causa que Cocoon analice en primer lugar document.xml, y después lo transforme usando $COCOON_HOME/transforms/content2html.xsl, antes de enviar los resultados de vuelta al navegador.

Se pueden conseguir transformaciones más complejas añadiendo más transformadores a la *pipeline*. De nuevo, las *wildcards* se han usado para evitar tener que defini las entradas reales a la *pipeline*, usando los resultados del proceso de captura de las URLs.

3.5.4.6.Generar otros formatos

Ahora que hemos visto cómo generar dinámicamente HTML a partir de XML, vamos a considerar cómo tratar otros formatos de documentos. Cocoon soporta la generación de otros formatos con el soporte de serializadores personalizados.

Cuando se envía XML en lugar de HTML como resultado de una transformación, se debe usar el serializador XML. Si está disponible una transformación para generar un archivo para compartir contenidos RSS, entonces podemos escribir una *pipeline* que incluya el siguiente fragmento:

```
<map:transform src="transforms/content2rss.xsl"/>

<map:serialize type="xml"/>
```

Se debe notar que se ha seleccionado un serializador específico mediante el atributo `type`. El valor del atributo encaja con el nombre de uno de los serializadores de las declaraciones de componentes realizadas anteriormente en esta sección.

SVG (Scalable Vector Graphics) es un formato para describir diagramas de líneas. Normalmente, se necesita un plugin en el navegador para ver documentos SVG. De cualquier forma, Cocoon incluye un serializador que es capaz de crear imágenes JPEG o PNG directamente desde un documento SVG. Este serializador se podría invocar de la siguiente forma:

```
<map:transform src="transforms/content2svg.xsl"/>

<map:serialize type="svg2png"/>
```

Cocoon también soporta la creación de archivos PDF directamente desde documentos XSL-FO. De nuevo, si nos aseguramos de que la hoja de estilo produce el formato de documento correcto para la entrada al serializador especializado, sólo serán necesarias las siguientes líneas:

```
<map:transform src="transforms/content2fo.xsl"/>

<map:serialize type="fo2pdf"/>
```

3.5.4.7.Paso de parámetros

A menudo es útil pasar parámetros a una transformación XSLT. Cocoon soporta el paso de parámetros desde el *sitemap*. La primera forma de conseguirlo es con el elemento `map:parameter`. Aquí mostramos un ejemplo:

Listado 3.7: Paso de parámetros a una hoja xsl.

```
<map:transform src="transforms/content2html.xsl">

   <map:parameter name="myFixedParam" value="fixed-value"/>
   <map:parameter name="myDynamicParam" value="{1}"/>

</map:transform
```

Tanto el nombre como el valor del parámetro se especifican como atributos del elemento `map:parameter`. Es posible pasar a las hojas de estilo parámetros tanto fijos como dinámicos con este método. El segundo parámetro que se pasa en este ejemplo tendrá como valor el indicado por la captura del patrón referenciado como {1} por la *wildcard*. Si la hoja de estilo incluye un elemento `xsl:param`, el parámetro será correctamente pasado a la transformación.

La forma alternativa para el paso de parámetros a una hoja de estilo permite el paso de

todos los parámetros de una petición en una URL. Por ejemplo, si la petición de la página `http://localhost:8080/content.html?param1=value1&param2=value2` provoca que se lance la siguiente *pipeline*, entonces se pasarán dos parámetros (`param1` y `param2`) a la hoja de estilo.

Listado 3.8: Forma alternativa para el paso de parámetros a una hoja xsl.

```
<map:transform src="transforms/content2html.xsl">
   <map:parameter name="use-request-parameters" value="true"/>
</map:transform>
```

Este método es útil cuando hay un número variable de parámetros que se pueden pasar en una petición. Sin embargo, esto provoca un coste en el rendimiento, porque Cocoon maneja con más dificultad la caché de resultados de transformaciones que usan este método que la de los que usan parámetros fijos. Si cualquiera de los parámetros de la URL cambia, aunque no sean directamente utilizados por la hoja de estilo, los resultados almacenados en la caché no se usarán.

capítulo 4:
apache lenya

Apache Lenya [LENYA2005] es un Sistema de Gestión de Contenidos escrito en Java y basado en estándares abiertos tales como XML o XSLT. Lenya se construye sobre Apache Cocoon y otros componentes de la pila de software de la Fundación Apache. Su arquitectura basada en XML permite el desarrollo de contenidos enfocado hacia las capacidades de varios dispositivos (por ejemplo, hacia navegadores web o hacia navegadores wml).

Las líneas de diseño generales de Lenya siempre han observado una interacción muy profunda con Cocoon. Lenya es, realmente, una aplicación desarrollada y desplegada sobre Cocoon. Esto tiene ciertas ventajas como puede ser la facilidad de inserción de características y bloques propios de Cocoon dentro de Lenya, aún cuando no estuviera planeado en un principio por parte de los desarrolladores.

Lenya abarca a un sitio basado en Cocoon. Le otorga las funcionalidades clásicas de un Gestor de Contenidos, entre las que se encuentran:

- La posibilidad de tener interfaces de edición de los contenidos del sitio (en el caso de Lenya estos interfaces de edición son interfaces web accesibles a través de un navegador).
- Herramientas para la gestión del sitio web. Necesarias para la realización de tareas tales como crear, mover, borrar... documentos.
- Planificador de tareas. Permite la automatización de trabajos tales como la publicación o desactivación de páginas en fechas determinadas.
- Notificación vía e-mail de sucesos (generalmente lanzada por transiciones en el workflow) como pueden ser la presentación de una página para revisión o la publicación de un contenido.
- Control de Acceso. Para que sólo los usuarios capacitados por el administrador del sistema puedan editar los contenidos del portal.
- Motor de búsqueda. Lenya utiliza Lucene que a su vez viene incluído en la distribución de Cocoon.

Además, otro de los principales puntos de referencia a la hora del diseño de Lenya fue el de la reutilización de componentes *OpenSource* probados y funcionales, entre los que podríamos destacar *Ant* y *Quartz Scheduler* (utilizado como base para el planificador de tareas).

También desde un primer momento se consideró necesario que se tratara de un gestor de contenidos descentralizado, esto es, que incluyera mecanismos de bloqueo de archivos para que se pudiera realizar un trabajo colaborativo a través de la red sin interferencias. Si un archivo está siendo usado por alguien, automáticamente se encuentra bloqueado para los demás hasta que el usuario actual lo desocupe.

Las tareas propias del gestor comentadas hace un momento se implementan a través de componentes propios desarrollados para Lenya. Estos componentes están incluidos en el núcleo y vienen con implementaciones por defecto para su funcionamiento desde un primer momento. Esto también supone una línea de diseño predefinida. Lenya separa los conceptos “núcleo del gestor” y “publicaciones”.

- Núcleo. Tiene la tarea de encargarse de las funcionalidades básicas y deja para la publicación las características propias de cada desarrollo particular. El núcleo se corresponde con una especie de elemento monolítico preparado para funcionar siempre de la misma manera, evitando así el cambio de las clases que lo implementan y problemas de desarrollo entre versiones y personalización. Si el núcleo se queda inalterable y todos los cambios referentes a personalización se incluyen en las publicaciones, siempre será posible actualizar la versión del núcleo evitando muchos de los problemas de compatibilidad.

- Publicación. Es, en realidad, el portal o *website* que Lenya se encarga de gestionar. Cada una de las publicaciones se puede personalizar de la forma que se desee. Así podremos, por ejemplo, cambiar los elementos de navegación o la forma en la que se mapean los documentos a una URL en una publicación basada en Lenya, mientras que otra publicación distinta que se encuentre cohabitando con la anterior en nuestro motor de servlets mantendrá intacta la funcionalidad por defecto.

La particularización de las funcionalidades para una publicación se consigue por medio del llamado mecanismo de *fallback*. Éste busca archivos o clases personalizados en la estructura de directorios de la publicación, y si no los encuentra, pasa a utilizar los genéricos incluidos en el núcleo de Lenya. En un apartado posterior comentaremos la estructura de archivos de Lenya.

Por último, aparte de la búsqueda de una migración simple de sitios basados en Cocoon, una de las características más aplaudidas de Lenya es la intención de desarrollarse conforme a estándares como XML, XSL, WebDAV, XHTML, etc.

4.1.Proyectos relacionados.

Como ya hemos comentado, Lenya es una aplicación basada en Cocoon. Por lo comentado, y debido a la interacción propia existente entre los proyectos de la Fundación Apache y a los esfuerzos de la ésta para la coordinación y el desarrollo conjunto de sus iniciativas, Lenya se relaciona con los siguientes proyectos:

- Cocoon. Es un marco de desarrollo y publicación web basado en XML y XSLT. Está diseñado en torno a un procesado de eventos SAX y ofrece un entorno flexible basado en la separación entre los contenidos, la lógica y el estilo. [COCOON2006]
- Slide. El módulo principal del proyecto Slide es un Sistema de Gestión de Contenidos e Integración, que puede ser visto como un marco de desarrollo para gestión de contenidos de bajo nivel. De una forma conceptual, provee una organización jerárquica de los contenidos binarios que se pueden guardar en almacenes de datos arbitrarios, distribuidos y heterogéneos. Además, Slide integra herramientas de seguridad, bloqueo y control de versión. [SLIDE2006]
- Jackrabbit. Este proyecto se ha creado para desarrollar una implementación de fuente abierta del JCR (Content Repository for Java Technology API), que se especifica en la JSR-170. No está implementado aún en las versiones 1.2.x aunque se planea introducirlo en la rama 1.4. [JACKR2006]
- Ant. Es una herramienta basada en Java para la compilación y construcción de programas (una especie de *make* mejorado). Los archivos de configuración están basados en XML. Dentro de ellos hay un árbol de objetivos a partir del cual se ejecutan las tareas. [ANT2006]
- Forrest. Forrest es un framework de documentación XML basado en Cocoon. Provee hojas de estilo XSLT, esquemas, imágenes y otros recursos. Forrest utiliza todos estos elementos para renderizar el código fuente XML dentro de un sitio web mediante órdenes de línea de comandos, aplicaciones automatizadas o aplicaciones web dinámicas. [FORREST2006]
- Excalibur. Excalibur es proyecto basado en Avalon (proyecto clausurado por la Apache Software Foundation en 2004) cuyo principal producto es un contenedor ligero y empotrable llamado Fortress. Está basado en la Inversión de control y se encuentra escrito en Java. Intenta crear un pequeño bloque de software (llamado contenedor) que se encarga de indicarles a sus componentes como tienen que interactuar entre ellos. Excalibur se encuentra integrado en el núcleo de Cocoon. [EXCALIB2006]

- WebDAV. WebDAV toma su nombre de *Web-based Distributed Authoring and Versioning*. Es un conjunto de extensiones para el protocolo HTTP que permite a los usuarios editar y gestionar archivos que se encuentran en servidores remotos de forma colaborativa. [WEBDAV2006]

4.2.Conceptos.

Lenya extiende el framework de Cocoon añadiéndole acciones y *matchers* personalizados. Además, también define dos nuevos esquemas propietarios:

- fallback: (para las versiones 1.2 y 1.4)
- lenya: (para la versión 1.4)

Estos esquemas están unidos a un módulo de entrada personalizado para Lenya, el *PageEnvelope*.

4.2.1.Área de Edición, área de producción, interfaz de usuario y publicaciones.

La instalación de Lenya agrega varias partes dentro de una única aplicación Cocoon. Por defecto, una instancia de Lenya puede ser usada para editar y renderizar un número arbitrario de publicaciones que son completamente independientes entre sí.

Hay varias formas de ver lo que significa el concepto publicación, pero por ahora asumiremos que cada publicación representa un sitio web independiente.

Dentro de cada publicación nos podemos encontrar con:

- El área de edición
- El área de producción
- Los componentes de la interfaz del CMS (menús desplegables, editores, etc.)

El espacio de URIs se utiliza como medio de organización y para controlar el acceso a estos lugares.

4.2.2.El espacio de URIs de Lenya

En el caso de desplegar Lenya dentro del contexto raíz de un contenedor de servlets, se puede acceder a la aplicación a través de http://localhost:8080/ , de forma que se pedirá directamente el *sitemap* raíz de Lenya. En el caso de que no se haya desplegado en el contexto raíz del contenedor, la primera parte de la URI se utilizará por éste para decidir qué aplicación es la responsable de gestionar la petición. En este caso, sería necesario pedir *http://localhost:8080/lenya* para alcanzar el *sitemap* raíz de Lenya.

En el resto de este apartado consideraremos que Lenya ha sido desplegado en el contexto raíz del contenedor. Intentaremos observar qué hace Lenya para renderizar el documento que se obtiene cuando se solicita la URL *http://localhost:8080/default/authoring/tutorial/new_doctype.html*

4.2.2.1.Parte 1: El identificador de la publicación.

La primera parte de la URL solicitada es el identificador de la publicación. En este caso se trata de *default* lo que le indica a Lenya que debe seleccionar la publicación por defecto. El nombre de la publicación y su identificador no tienen por qué ser iguales. El identificador debería ser compatible tanto con el sistema de archivos en el que se almacena como con el esquema de

codificación de la URI, ya que formará parte tanto del nombre del directorio que guardará la publicación como de la URL. Por tanto, es una buena práctica ajustarse a un nombre basado en ASCII de 7 bit sin espacios o caracteres especiales.

Por el contrario, el nombre de la publicación (que se mostrará en la lista de publicaciones en la página de inicio de Lenya) puede ser más largo y contener tanto espacio como caracteres Unicode.

El identificador de la publicación se utiliza para montar el *sitemap.xmap* específico de la misma en $LENYA_HOME/pubs/{pub-id}. La operación *map:mount* separará el identificador de la publicación del resto de la URL, de forma que el *sitemap* de la misma sólo observará el trozo *authoring/tutorial/new_doctype.html*.

Sin embargo, una publicación tiene su identificador siempre disponible a través del módulo *PageEnvelope*.

4.2.2.2.Parte 2: El área.

En Lenya se definen dos posibles áreas:

- Edición (Authoring)
- Producción (Live)

Se podría pensar en ellas como si nos encontráramos dentro de dos modos distintos. El "modo de producción" y el "modo de edición". El de producción permite ver la publicación como se mostraría a un visitante que solicitara la página. El de edición se utiliza para que los editores y los revisores puedan modificar el contenido de las publicaciones.

La mayor diferencia entre ambos modos es que en el de edición los menús del CMS no se muestran. Siguiendo los principios WYSIWYG de Lenya, el contenido de la publicación se muestra de la misma forma tanto en el modo de edición como en el de producción.

Aparte de que se muestren o no los menús, hay copias distintas del documento en el repositorio para cada una de las dos áreas. Esto permite a los editores la modificación de una copia de trabajo sin que esto afecte al sitio en producción. Cuando un documento se publica después de su revisión, simplemente se copia en el lugar correcto del repositorio.

De la misma forma que el identificador de la publicación, el área en el que nos encontramos también se encuentra almacenado en el módulo *PageEnvelope*. Esto hace posible que el área actual se encuentre disponible tanto para el *sitemap* como para los componentes de la capa de Java de Lenya a través del módulo de entrada *PageEnvelope*.

En este punto, Lenya ha procesado la URL de forma que ya se conoce:

- a qué publicación pertenece la petición
- qué repositorio debe usarse (edición o producción)
- si debe o no mostrar los menús del CMS

4.2.2.2.1.Menús del CMS, Casos de Uso y pantallas del CMS.

Antes de llegar al contenido real de la publicación, vamos a realizar algunas observaciones sobre los menús del CMS y las pantallas del CMS.

Lenya utiliza los denominados casos de uso para realizar acciones. Casos de uso pueden ser:

- enviar (un documento)
- publicar (un documento)
- ...

Los menús del CMS no son más que un mecanismo que permite lanzar casos de uso al

usuario del CMS. Como alternativa a la elección en el menú de “Workflow” --> “Enviar”, se podría también añadir *?lenya.usecase=submit* a la URL del documento. En el momento en que Lenya encontrara un parámetro *lenya.usecase* en la petición, sabría que se está solicitando el lanzamiento de un caso de uso y lo procesaría.

4.2.2.3.Parte 3: La URL del documento.

En este punto, el *sitemap* de la publicación se encarga de tomar la URL del documento de la porción de la URL original (en este caso, *tutorial/new_doctuype.html*) y de generar y renderizar el contenido de la página. Esta tarea aparentemente simple se puede volver bastante complicada. Este trozo de URL es el que marca las diferencias más grandes entre los distintos CMS en cuestión de características y configurabilidad.

La forma más simple de tratarlo sería:

- Elegir el repositorio de contenido apropiado (producción o edición).
- Usar la URL del documento para encontrar un archivo por su nombre.
- Si es necesario, aplicar una hoja XSLT y serializar el resultado.

Lenya podría realizar más acciones en este punto:

- Usar una clase específica de la publicación para mapear la URL del documento a un repositorio interno. Esto permitiría ocultar la estructura real del repositorio a un visitante de la web.
- Decidir qué versión de idioma usar para servir el documento. Por ejemplo, si no se especifica un idioma, Lenya podría decidir usar el idioma definido por defecto. Si se solicita un idioma específico, Lenya lo buscaría en el repositorio y lo generaría si estuviera disponible. Si no existiera una versión para el idioma solicitado, Lenya volvería a optar por mostrar la versión en el idioma por defecto.
- Elegir entre una de las *pipelines* de presentación basándose en el tipo de recurso asociado a la fuente del documento. Un tipo de recurso puede ser XHTML, pero también podría ser cualquier otro formato XML tal como RSS, SVG, SlideML, etc.

La mayor parte de esta lógica se implementa en la capa Java. Los componentes de esta lógica son componentes Avalon, que se configuran para cada publicación. En otras palabra, se podrían implementar e introducir versiones personalizadas de estos componentes.

Sólo el mapeado de la URL a la ruta utilizada para acceder al repositorio está implementado en Java como un componente Avalon. El renderizado del documento se realiza por medio de los *sitemaps* de Cocoon. Se puede encontrar un ejemplo de cómo definir un mapeado completamente distinto al de la publicación por defecto en la publicación Weblog de Lenya, que viene también incluida en la distribución básica.

4.2.2.4.Parte 4: Parámetros de la URL.

Estos parámetros son opcionales y entre ellos podemos encontrar los parámetros específicos de Lenya, como pueden ser los referidos a los casos de uso y al workflow, y los específicos de la publicación, introducidos por el desarrollador.

Los parámetros para los casos de uso pueden ser:

- lenya.usecase
- lenya.step

Mientras que para el workflow podemos encontrarnos con:

- wf.event

4.3.Arquitectura funcional.

Lenya abarca a un sitio basado en Cocoon dotándolo de las funcionalidades necesarias para realizar las tareas asociadas a la gestión de contenidos mediante la introducción de componentes específicamente diseñados. Por tanto, le es de aplicación el análisis realizado anteriormente para la arquitectura de Cocoon.

Lenya también basa su funcionamiento en un archivo de configuración principal: el *sitemap*. Además, al igual que Cocoon, procesa las diferentes peticiones mediante las *pipelines*, haciendo uso extensivo de todas las características ya mencionadas en el apartado Arquitectura de Cocoon 2, esto es, generadores, transformadores, serializadores, etc. Un esquema de la arquitectura de Lenya podría ser el mostrado en la Figura 16.

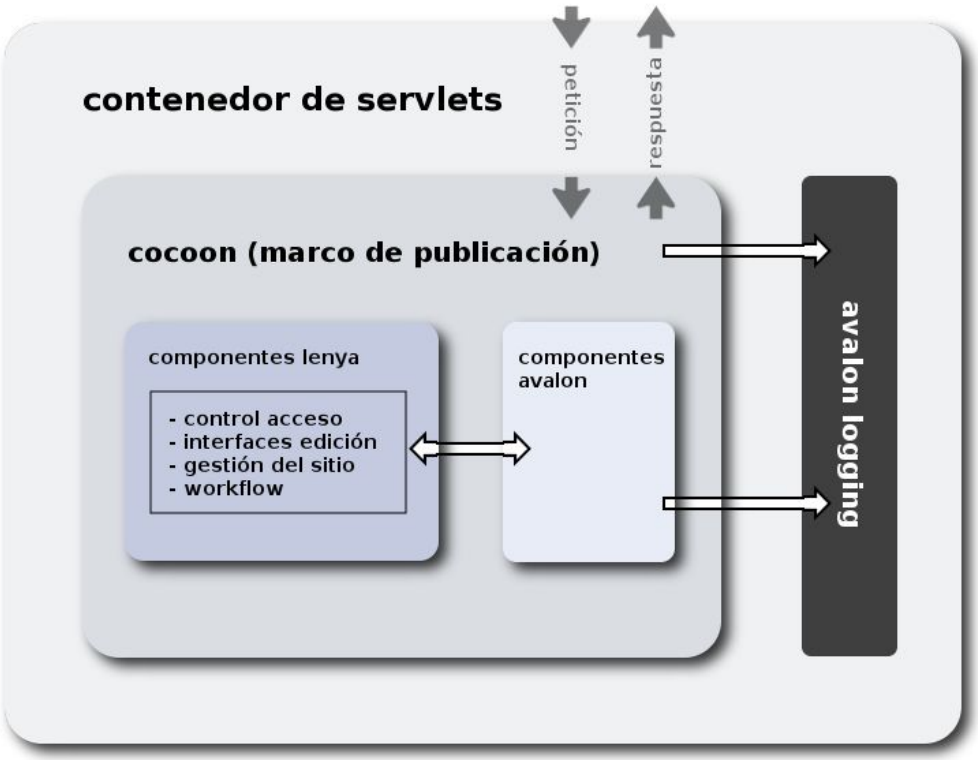

Figura 16: Arquitectura de Lenya con sus componentes particulares.

Recordando en este punto el esquema de la arquitectura de Cocoon (Figura 13), dentro del bloque del *Sitemap* es donde encajaría el que hemos denominado *Componentes Lenya* en la Figura 16. En este bloque se encuentran todos los componentes específicos diseñados, así como los heredados de Cocoon.

Si bien es cierto, como hemos dicho, que la estructura básica de Lenya se corresponde con la de Cocoon, también lo es que el sistema de gestión de *sitemaps* de Lenya es algo más complicado que el de una aplicación Cocoon común. Lenya necesita montar y desmontar varios *sub-sitemaps* para su correcto funcionamiento. El control de acceso, la gestión de los casos de uso, la gestión de tareas, etc. hacen un uso extensivo de estos mecanismos.

El punto de entrada principal a Lenya se encuentra en el archivo *sitemap.xmap* que se halla en la raíz del contenedor del servlet. A través del procesamiento de este archivo se llega a la definición de los componentes presentes en Lenya, tanto los heredados de Cocoon como los realizados expresamente para él. Se definen los generadores, transformadores, serializadores,

readers, matchers, selectores, acciones y vistas que estarán disponibles para todas las publicaciones y para el núcleo de Lenya. Esta primera sección es extensa y en ella se realiza la configuración principal de la aplicación.

Una vez pasada esta zona de configuración y declaración de componentes, nos encontramos con la *pipeline* principal de Lenya. Ella será la encargada de resolver la acción correcta teniendo en cuenta la petición que haya recibido el servlet. En esta *pipeline* nos encontramos desde la aplicación de la transformación de idioma (mediante el montado del *sub-sitemap* correspondiente a i18n que es el medio de internacionalización adoptado por Cocoon), hasta el servicio de imágenes, hojas de estilo o funciones javascript, pasando por el montado del *global-sitemap.xmap.*

Este archivo es el encargado de capturar los enlaces correspondientes a las páginas de inicio de Lenya, así como de el envío de la petición hacia un nuevo *sub-sitemap* en un nivel inferior. Así, se encarga, por ejemplo, de mostrar la página de inicio, lanzar el *sitemap* de la búsqueda mediante Lucene, mostrar la interfaz de usuario de la administración de Lenya o comprobar el caso de uso en el que nos encontramos. Todo esto lo realiza en tiempo de ejecución buscando qué patrón de la petición que se le ha pasado encaja en la *pipeline* que se está procesando. Se puede encontrar el listado del *sitemap* principal de Lenya en el Apéndice 6.2: Apéndice B: El sitemap de Lenya.

4.3.1.Proceso de una petición.

Todas las peticiones que un cliente (sea un navegador web, una pda, etc.) realiza a Lenya siguen un esquema similar de procesamiento, pasando por cada uno de los siguientes pasos:

A) La petición se recibe por el Sistema Operativo.

B) El Sistema Operativo pasa la petición al contenedor de servlets.

C) El Contenedor decide qué aplicación de las que contiene es la receptora de la petición (en este caso Lenya).

D) Lenya recibe la petición y busca en su *sitemap.xmap* principal la instrucción de proceso que encaje en la petición.

E) A continuación (y en general) pasa la petición al *global-sitemap.xmap* que se encarga de decidir si hay que hacer uso del *usecase.xmap* o si bien hay que pasarla a la publicación para que sea el *publication-sitemap.xmap* específico de la misma el que se encargue del procesamiento.

F) La petición se procesa y se devuelve al cliente.

Como podemos comprobar, entre los puntos D y E se abre un espectro bastante amplio de posibilidades de procesamiento de la petición. El lugar donde acabe de procesarse dependerá exclusivamente de los valores de los parámetros de la URL. Si la petición llega a la publicación, se continuarán montando *sitemaps* para gestionar los posibles casos de uso que se presenten o para formatear la información que se devolverá como resultado de la petición. El número de *sub-sitemaps* que se montan puede ser muy variable, dependiendo, por ejemplo, de si lanzamos una petición de edición de la página que estamos visitando.

4.3.2.Aproximación a la Estructura del *sitemap* de Lenya.

En una aplicación Cocoon simple, sólo existe un archivo *sitemap.xmap*. Pero Lenya no es lo que podríamos considerar una aplicación simple. Por consiguiente, el *sitemap* de Lenya está distribuido en varios archivos dentro del directorio de la aplicación web [WIKI-LEN-SIT2005].

Lenya está formado por un conjunto de sitemaps que se utilizan para procesar la petición y devolver la respuesta. Se podría pensar que alguno de los motivos para hacerlo sería el que se consiguiera una depuración más simple de nuestras aplicaciones web y un aprendizaje más

rápido del funcionamiento del gestor, pero esto no es lo que sucede. Parecería más fácil concentrarlo todo en un sólo *sitemap*, pero hay varias razones para tener varios *sitemaps* aun a riesgo de ganar complejidad:

- Separación del contenido de la publicación y de la interfaz del CMS.
- Separación del núcleo de Lenya y las partes específicas de la publicación.
- Implementación de los mecanismos de *fallback*.
- Montaje automático de los *sub-sitemaps* dependiendo de los valores de las variables del *sitemap* padre en tiempo de ejecución para una petición en particular (por ejemplo, el tipo de documento).

Estos *sitemaps* se montan y desmontan automáticamente dependiendo de la petición que estén procesando. Los mecanismos de montado y desmontado de *sitemaps* hacen que no sea simple comprender el funcionamiento de Lenya, tal y como se puede comprobar en el Apéndice 6.2: Apéndice C: Generación de una página en Lenya..

*4.3.2.1.1.*Jerarquía de *sitemaps*.

Como hemos comentado en el apartado anterior, los *sitemaps* de Lenya se seleccionan en tiempo de ejecución, dirigiendo el flujo de la misma hacia un *sitemap* de un nivel inferior si fuera necesario. En la Figura 17 podemos ver un esquema simplificado de este proceso.

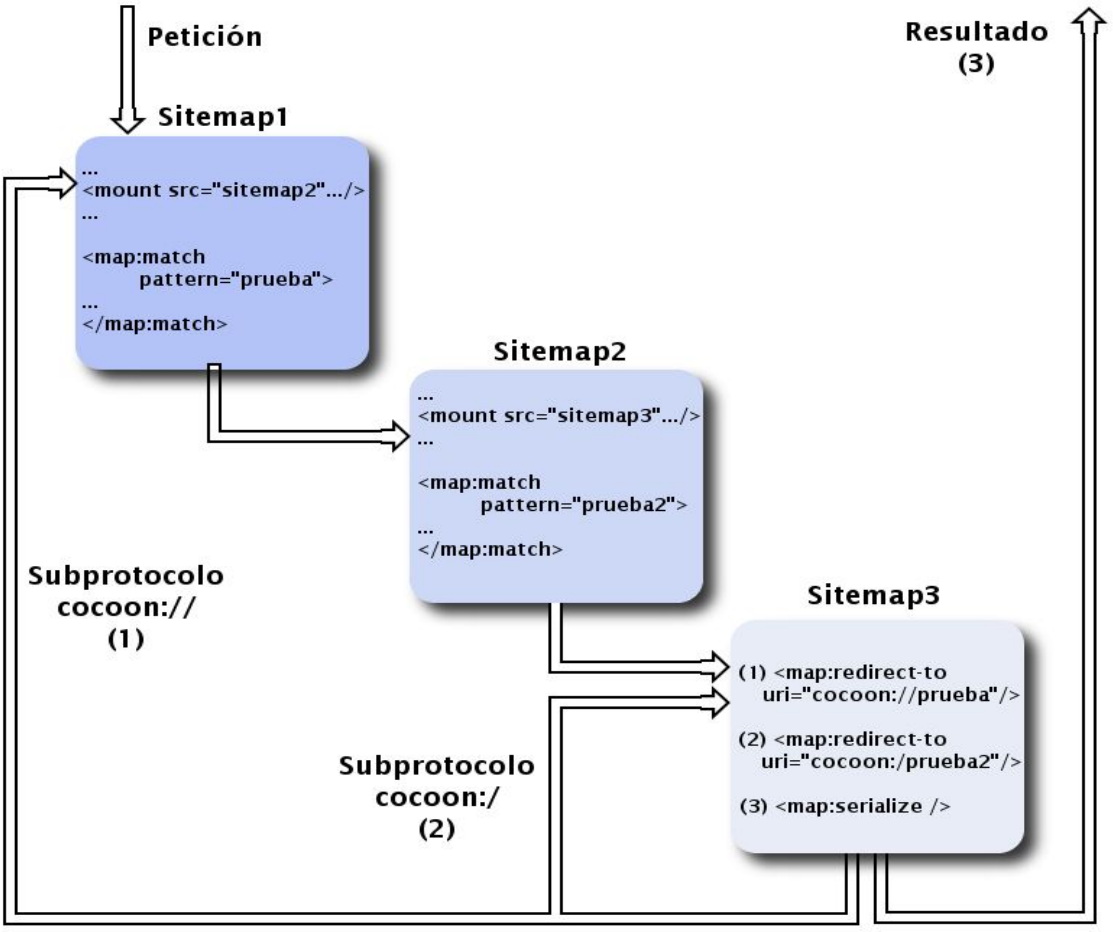

Figura 17: Esquema simplificado del montado de los sitemaps.

En esta figura podemos observar un flujo normal de procesamiento de una petición de una URL. La petición entra en el Sitemap1 y tras ser procesada dentro de él se decide que es necesario proceder al montado del Sitemap2 para seguir atendiéndola. Dentro de éste, se observa que para seguir resolviéndola es necesario montar el Sitemap3, por lo que se procede al montaje del mismo y se le pasa el control. Una vez en este *sitemap* hemos distinguido tres casos:

- Caso 1: Se decide que para seguir resolviendo la petición es necesario ejecutar una pipeline del Sitemap1, por lo que a través del protocolo *cocoon://* se redirige la petición hacía la URI *cocoon://prueba* que será capturada y procesada en el Sitemap1.
- Caso 2: Se decide que para seguir resolviendo la petición es necesario ejecutar una pipeline distinta del mismo *sitemap* en el que se encuentra, por lo que mediante el protocolo *cocoon:/* se pasa el control otra vez al inicio del Sitemap3 para que se capture la URI *cocoon:/prueba2*.
- Caso 3: Se decide que la petición ha sido completamente resuelta y se serializa el resultado, permitiendo la devolución del mismo al navegador del cliente que la originó.

Este ejemplo, aunque simple, nos puede ayudar a tener una idea de cómo se realiza en realidad el procesamiento de las peticiones. Decimos que es simple, porque, por ejemplo, en el caso 1 en lugar de capturar la URI en el Sitemap1 se podría haber capturado en el Sitemap2 y una vez procesada haberla devuelto al cliente. O porque podía haberse dado el caso de que una vez devuelto el control al Sitemap1, éste hubiera decidido procesarla y volverla a mandar al Sitemap3.

Como podemos comprobar, la casuística en un ejemplo tan simple es muy grande. En Lenya hay muchos más *sitemaps* que en este pequeño ejemplo. Hay algunos que se montan para los servicios de edición, otros que se montan para la gestión de los casos de uso, otros que se montan para tareas de internacionalización, etc., motivos todos ellos que hacen difícil en un primer momento hacerse con su funcionamiento.

En la Tabla 1 podemos encontrar una descripción de los *sitemaps* más importantes de Lenya. Se incluye para cada uno de ellos su localización, su nombre, sus funciones y el ámbito en el que es aplicable. La variable `$RAIZ` hace referencia a la raíz de la aplicación web dentro del contenedor. Dependiendo de la forma en la que se haya desplegado dentro de Tomcat, ésta puede ser o bien `$TOMCAT_HOME/webapps/ROOT` o bien `$TOMCAT_HOME/webapps/lenya`.

Nombre	Localización	Funciones	Ámbito

`sitemap.xmap`	`$RAIZ`	`- Definición de los componentes generales de la aplicación.`
		`- Captura de las peticiones para la internacionalización y gestión del caso de uso de login.`
		`- Montaje del global-sitemap.xmap`
			`General`

Nombre	Localización	Funciones	Ámbito
global-sitemap.xmap	$RAIZ	- Sirve la página de inicio del gestor de contenidos. - Captura los eventos de la búsqueda de mediante Lucene. - Captura las llamadas a los editores y monta los sitemaps correspondientes. - Monta el sitemap de los casos de uso. - Para una publicación hace las llamadas nesarias a los sitemaps de las publicaciones para montar la zona de administración, la zona de producción, etc.	General
usecase.xmap	$RAIZ/lenya	- Monta los sitemaps de los casos de uso específicos de la publicación si estos existen. - Captura los casos de uso y ejecuta las acciones necesarias para llevarlos a cabo si no hay alguno específicamente creado para la publicación.	General
lenya.xmap	$RAIZ/lenya	- Captura y sirve todos los recursos (imágenes, hojas css, archivos js, etc.) generales a lenya.	General
resources.xmap	$RAIZ/lenya	- Sirve todos los recursos específicos de la aplicación con su tipo mime correspondiente.	General
navigation.xmap	$RAIZ/lenya	- Crea los componentes de navegación por defecto. - Comprueba si hay alguno definido específicamente para una publicación y si no lo hay aplica el general.	General
admin.xmap	$RAIZ/lenya	- Sitemap que genera todos los elementos de la administración. Hace uso del Flujo de Cocoon.	General
import.xmap	$RAIZ/lenya	- Importa elementos externos en el gestor, como archivos OpenOffice o Microsoft Word.	General
lucene.xmap	$RAIZ/lenya	- Ejecuta el servicio de búsqueda mediante Lucene.	General
i18n.xmap	$RAIZ/lenya	- Sirve los archivos y las transformaciones necesarias para los procesos de internacionalización.	General
scheduler.xmap	$RAIZ/lenya	- Se encarga de la captura y ejecución de las tareas relacionadas con el planificador.	General

Nombre	Localización	Funciones	Ámbito
sitemap.xmap	$RAIZ/lenya/pubs /$NOMBREPUB	- Testimonial. Pasa la llamada directamente al publication-sitemap.xmap	Publicación
publication-sitemap.xmap	$RAIZ/lenya/pubs /$NOMBREPUB	- Encargado de generar las páginas de una publicación agregando navegación, contenido, etc. - Utiliza el UriParametrizer para determinar el tipo de documento. - Agrega el menú de Lenya (dependiendo del área) y el cuerpo.	Publicación
doctypes.xmap	$RAIZ/lenya/pubs /$NOMBREPUB	- Encargado de la transformación de un tipo de documento específico mediante la hoja xsl correspondiente. - Encargado de mostrar las versiones recuperadas de un archivo.	Publicación
parameter-doctype.xmap	$RAIZ/lenya/pubs /$NOMBREPUB	- Devuelve el tipo de documento como un parámetro de la URI	Publicación
menus.xmap	$RAIZ/lenya/pubs /$NOMBREPUB	- Se encarga de la generación de los distintos tipos de menus dependiendo del tipo de documento o del área.	Publicación

Tabla 1: Sitemaps de Lenya

Existen aún más archivos *xmap* dentro de las publicaciones que vienen incluidas en Lenya, pero son muy dependientes de la publicación y pueden cambiar mucho su funcionamiento de unas a otras. Además, estos *sitemaps* se montan dentro de alguno de los listados en la tabla anterior, por lo que su funcionalidad estaría bastante clara.

4.4.Arquitectura física.

Apache Cocoon es framework que permite el desarrollo de aplicaciones web j2ee, y al ser Lenya una aplicación basada en Cocoon, también hereda esta característica. En el apartado 3.4 hemos comentado el esquema genérico de este tipo de aplicaciones.

En este apartado mostraremos la estructura de directorios de Lenya una vez desplegado en el contexto del contenedor de servlets, especificando la función de cada uno de los directorios que lo componen.

En la Figura 18, observamos el esquema de la aplicación una vez desplegado en el contexto raíz de Tomcat. Este contexto (indicado en Tomcat como ROOT) nos permite acceder a Lenya a través de una URL como *http://localhost:8080/* sin tener que especificar la aplicación que gestionará la petición después de indicar la dirección y el puerto.

En los nodos identificados del (1) al (5) tenemos la estructura básica de los directorios e la aplicación.

El nodo (1) identifica la carpeta que almacena todas las funcionalidades específicas de

Lenya, la configuración de sus publicaciones y el contenido de las mismas. El *sitemap* raíz de la aplicación monta los que se encuentran en este directorio para realizar el procesamiento de la URL.

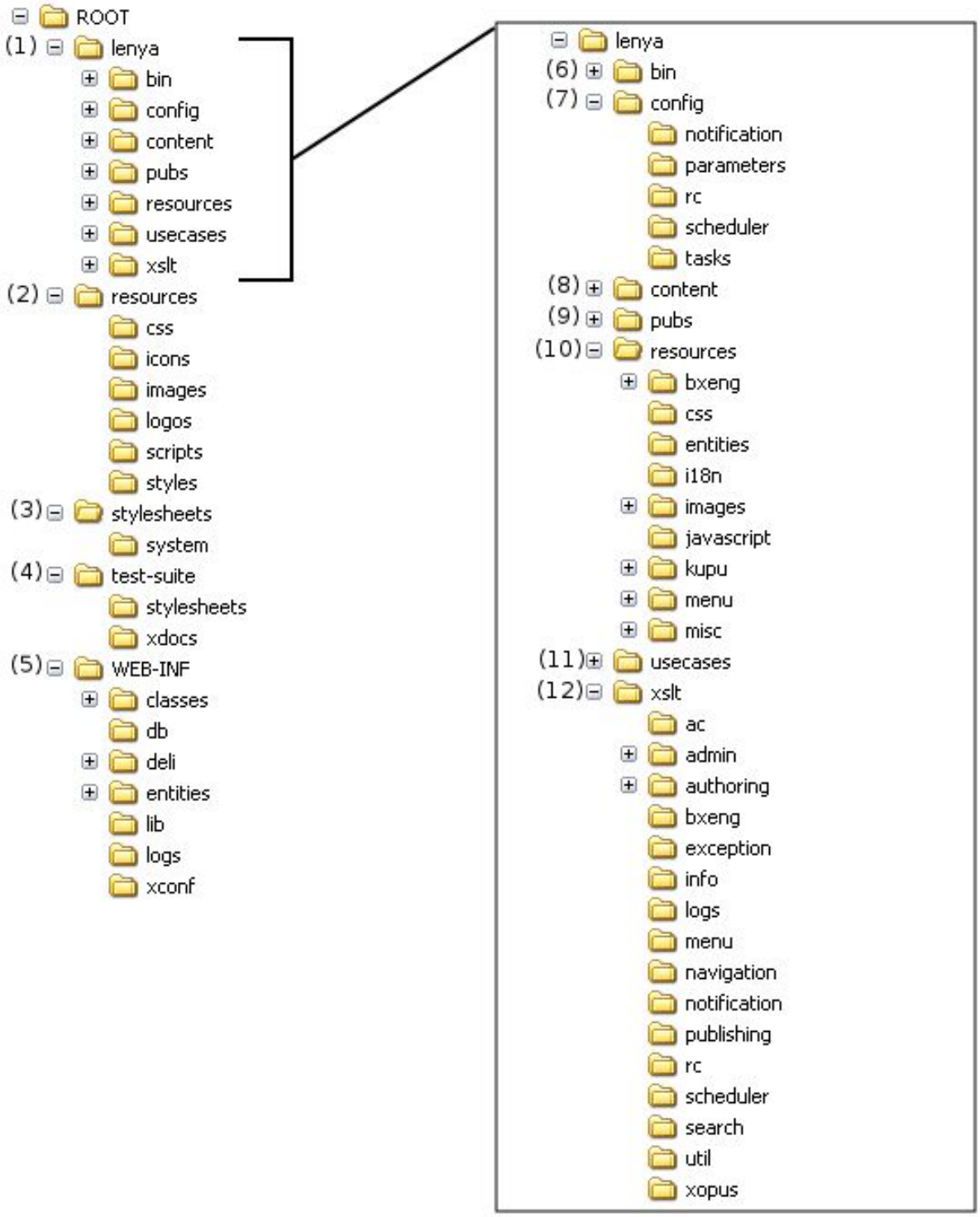

Figura 18: Estructura de directorios de Lenya.

Los nodos etiquetados como (2), (3) y (4), son carpetas heredadas de Cocoon.

El nodo etiquetado como (5), *WEB-INF*, contiene todos los elementos necesarios para la configuración y el funcionamiento de la aplicación web. En él se encuentra el archivo *web.xml* en el que se definen los valores necesarios para que funcione el servlet de Cocoon y los archivos *xconf* necesarios para la configuración del sistema de *loging* (*log4j.xconf*) y de la aplicación (*cocoon.xconf*).

Dentro de este nodo tendremos las clases compiladas y las librerías necesarias para el funcionamiento de Lenya bajo los directorios *classes* y *lib* respectivamente. En la carpeta *logs* se almacenarán los resultados de los *loggers* definidos para la aplicación.

La carpeta en la que realmente se hallan los archivos de configuración y los recursos del gestor de contenidos está etiquetada con el número (1) como ya hemos comentado. En la raíz de la misma se hallan todos los *sitemaps* necesarios para el funcionamiento de la aplicación. Sobre estos podemos ver una explicación más detallada en el apartado 4.3.2.1.1.Jerarquía de sitemaps.

En la carpeta, *bin* identificada con la etiqueta (6), encontramos utilidades para la replica del portal en un servidor remoto y los scripts necesarios para la indexación de la publicación para el funcionamiento del motor de búsqueda.

En la carpeta *config*, identificada con la etiqueta (7), podemos encontrar los archivos *xconf* de configuración de los servicios de notificación, control de revisión, planificador y las tareas (un test para *Ant*).

En la carpeta *content*, identificada con la etiqueta (8), podemos encontrar los archivos de contenido de las páginas de inicio y bienvenida de Lenya, en la que se nos muestran una lista de las publicaciones que se hallan en esta instancia de Lenya. Además, contiene los generadores *xsp* necesarios para la creación de las páginas de administración, edición, los menús, etc. Estos generadores son los tomados por defecto cuando no se han definido componentes específicos para la publicación.

En la carpeta *pubs*, identificada con la etiqueta (9), podemos encontrar todas las publicaciones que se hallan dentro de la instancia de Lenya. En esta carpeta se almacena el contenido de las publicaciones y se pueden encontrar también todos los componentes específicos de la publicación, que se utilizarán en lugar de los del núcleo.

En la carpeta *resources*, identificada con la etiqueta (10), podemos encontrar todos los recursos comunes a Lenya. Las hojas de estilo de la aplicación (para la generación de los menús de administración, los árboles de los contenidos, etc.), las imágenes e iconos de la misma, los archivos de javascript necesarios, etc. se encuentran almacenados en esta carpeta. También se hallan en ella los dos editores WYSIWYG de Lenya: BXE y Kupu. También se pueden encontrar en esta carpeta los catálogos necesarios para la internacionalización, almacenados en el directorio *i18n*.

En la carpeta *usecases*, identificada con la etiqueta (11), podemos encontrar los *sitemaps* de algunos casos de uso que no son específicos de la publicación, como el de edición, o el de edición mediante el formulario.

En la carpeta *xslt*, identificada con la etiqueta (12), podemos encontrar todas las hojas de estilo *xslt* que se aplicarán para la generación del contenido, de los menús y de la estructura de la aplicación en el caso de que no se hayan definido hojas personalizadas en la publicación.

4.4.1.Estructura de archivos de una publicación.

La estructura de archivos de Lenya para una publicación se muestra en la Figura 19. El directorio raíz del que colgarían sería la carpeta que guarda la publicación en su interior.

Observando esta estructura podemos atender a los mecanismos que se han usado en el diseño para conseguir tener una separación entre presentación, contenido y lógica no sólo en el funcionamiento del gestor sino también en el modo en el que se estructura.

Como hemos comentado con anterioridad, Lenya ofrece un núcleo con características básicas que se pueden personalizar dentro de cada publicación. Bien, pues cada uno de los archivos, plantillas o clases java que guardemos dentro de la publicación sobreescribirá el funcionamiento del núcleo. Si hay alguna funcionalidad que ya esté implementada y que no deseamos alterar, simplemente no debe declararse. Entonces Lenya sabrá que tiene que usar la definida por defecto. Estos mecanismos se denominan *Fallback* dentro del funcionamiento de Lenya.

En la Figura 19, se explica el propósito de cada uno de los directorios que conforman la estructura de Lenya. Sólo sería interesante explicar la forma en la que se organiza el directorio *content*. Lenya utiliza como repositorio en las versiones 1.2.x el propio sistema de archivos, por lo que en este directorio es donde se encontrará almacenada toda la información correspondiente a la edición y publicación de contenidos añadida por editores. Pasemos a hablar de cada una de las carpetas:

- *archive.* Guarda en su interior todos los documentos que los editores han decidido almacenar para su posterior uso.
- *authoring.* Guarda en su interior todos los archivos que se encuentran en el estado de edición o de revisión.
- *live.* Guarda en su interior todos los archivos que se encuentran publicados y disponibles para ser visitados vía web.
- *trash.* Guarda en su interior todos los archivos que han sido eliminados de la web. Permite la restauración de un archivo eliminado
- *workflow.* Guarda la historia del workflow de todos los documentos que se han modificado tanto en el área de edición como en el archivo como en la papelera.
- *rcml* y *rcbak*. Directorios para el almacenamiento del histórico de control de revisiones y de las copias del control. Aquí es donde se almacenan los documentos que han sido sustituidos por versiones más nuevas.

Hay que señalar también que las carpetas *archive, authoring, live y trash* contienen un archivo llamado *sitetree.xml* que almacena el árbol de nodos que esquematiza la distribución de los archivos dentro de esa carpeta. Por ejemplo, para la siguiente estructura de archivos:

```
|--> conceptos
            |-->index_es.xml
|--> proyectos
            |-->index_es.xml
            |-->index_en.xml
|--> contactar
            |-->index_es.xml
```

Tendríamos un archivo *sitetree.xml* de la siguiente forma:

Listado

```
<?xml version="1.0" encoding="UTF-8"?>

<site xmlns="http://apache.org/cocoon/lenya/sitetree/1.0"
        xmlns:i18n="http://apache.org/cocoon/i18n/2.1"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        i18n:attr="label" label="Trash">


<node id="conceptos" visibleinnav="true">
  <label xml:lang="es">Conceptos sobre Lenya</label>
</node>
```

```
<node id="proyectos" visibleinnav="true">
  <label xml:lang="es">Proyectos relacionados</label>
  <label xml:lang="en">Related Projects</label>
</node>
<node id="contactar" visibleinnav="true">
  <label xml:lang="es">Información de Contacto</label>
</node>
</site>
```

Como podemos observar, este archivo es meramente descriptivo, intentando reflejar de forma fiel la estructura de la carpeta en forma de árbol, de ahí el nombre de *sitetree.xml*. Cada uno de los archivos *index_*.xml* son los archivos en los que se almacena la información. Son archivos xml en los que se guarda la metainformación de la página (editor, título, descripción, etc.) y el contenido añadido por los editores.

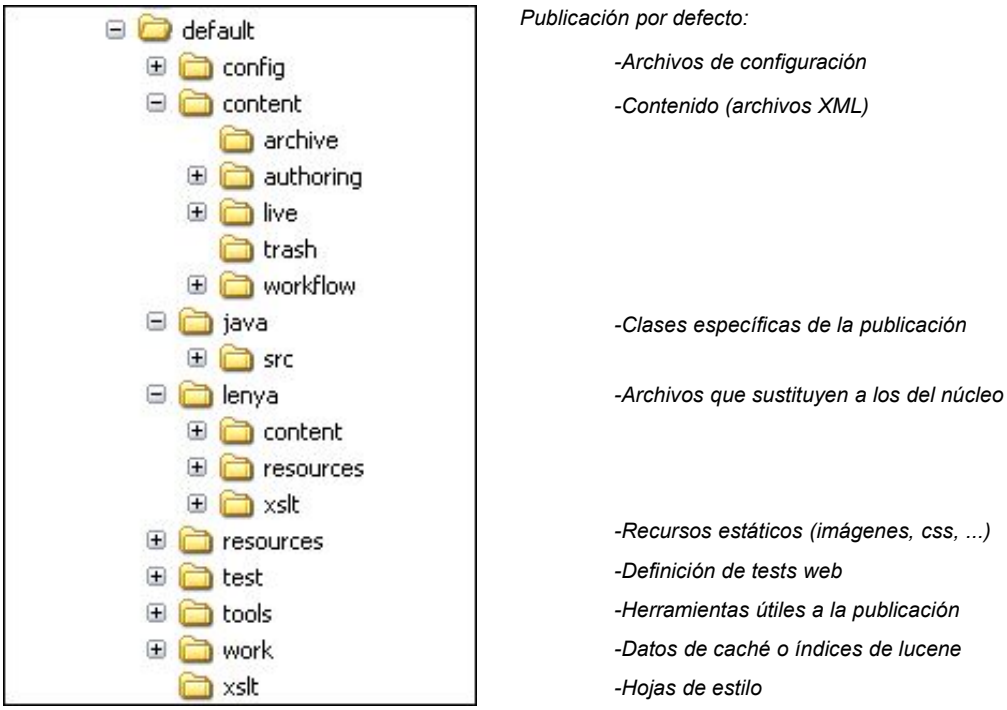

Figura 19: Estructura de directorios de una publicación de Apache Lenya.

También deberíamos comentar algo sobre la carpeta *config*. En ella se definen las características básica de configuración de la publicación. Las políticas de control de acceso, definición de los usuarios con permisos en el sitio, permisos de los usuarios, passwords y todas las tareas relacionadas se almacenan dentro de la carpeta *ac*. Por defecto, la configuración de las notificaciones y del control de revisión son las predefinidas para Lenya, por lo que si se desea personalizarlas, se deberán incluir en los directorios *notification* y *rc* respectivamente.

En la carpeta *config* también podemos encontrar las carpetas *doctypes, menus, publishing, search, tasks* y *workflow.* En cada una de ellas se podrán configurar para nuestra publicación (respectivamente) los esquemas de los tipos de documentos soportados, las páginas *xsp* necesarias para la generación de los menús, la configuración de los mecanismos de publicación, la configuración del motor de búsqueda (lucene), la configuración de las tareas y la configuración predefinida del workflow (roles o grupos necesarios para ejecutar una transición). Todos estos archivos de configuración son archivos xml.

4.5.Servicios de Lenya como CMS.

Lenya es un Sistema de Gestión de Contenidos bastante avanzado, por lo que implementa la mayoría de los servicios generales de un CMS indicados en la primera parte de esta memoria. Estos servicios se gestionan y se controlan a través del esquema de workflow de la publicación y del lanzamiento de casos de uso diferentes. La interrelación entre los casos de uso, el esquema de workflow y los módulos que implementan los servicios de una publicación se muestran en la Figura 20.

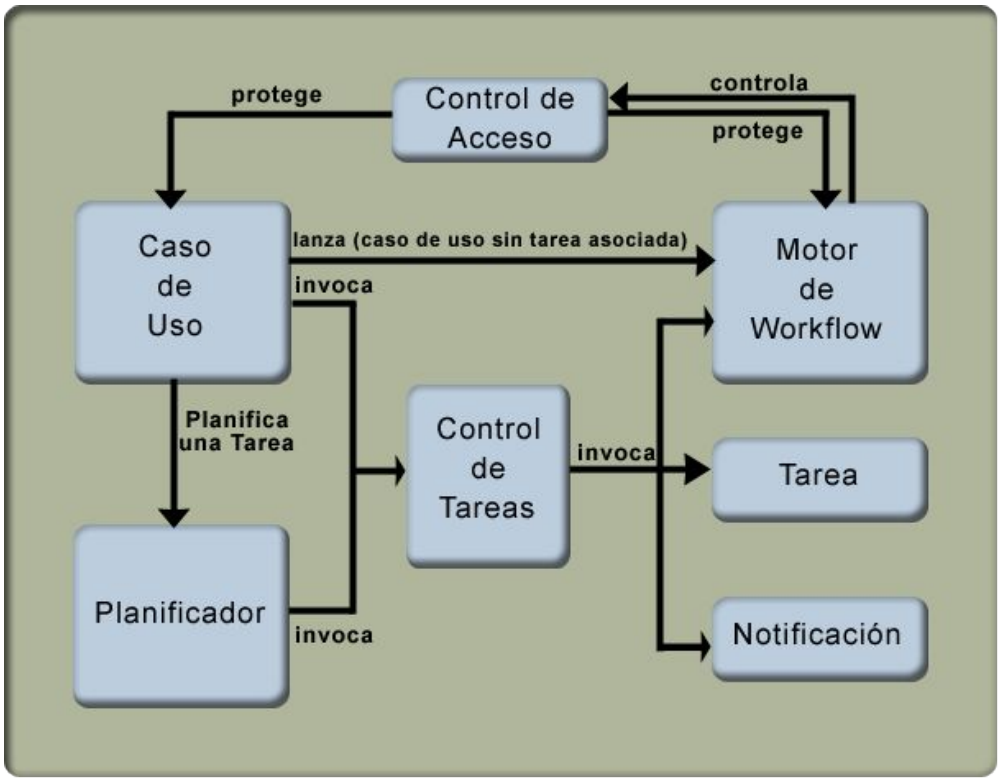

Figura 20: Interrelación de los componentes de Lenya

En los siguientes apartados mostraremos los servicios que vienen ya implementados en Lenya, así como una breve descripción de los mismos.

4.5.1.Gestión del flujo de Trabajo (Workflow).

Lenya viene con un esquema de workflow general que puede modificarse para cada publicación, convirtiéndose en un esquema personalizado que sólo será de aplicación a la publicación para la que esté definido. El esquema de Workflow que trae Lenya por defecto se puede ver en la Figura 24.

En ese esquema, además, se definen varios roles por defecto: administrador, editor, revisor, visitante (el mundo). El administrador es el encargado de dar de alta y de baja a los usuarios, asignarles roles y gestionar los rangos de ip desde los que pueden acceder, etc. Los editores son los encargados de añadir contenido al sitio web. Los revisores son los encargados de comprobar que el contenido enviado por los editores es correcto antes de que sea publicado en la red. Si no lo es, pueden rechazar el contenido presentado, volviendo el documento al estado de edición.

En el apartado 4.6.2 se puede obtener información acerca de cómo introducir cambios en

el esquema de workflow de la publicación.

4.5.2.Gestión de usuarios, grupos y administradores.

La gestión de los usuarios, grupos y administradores en Lenya se hace a través de una GUI en el navegador. Es necesario identificarse como administrador (en la configuración por defecto de las publicaciones) para poder realizar la gestión de los perfiles de cada uno de estos roles.

Una vez identificados, accedemos a la pantalla mostrada en la Figura 21.

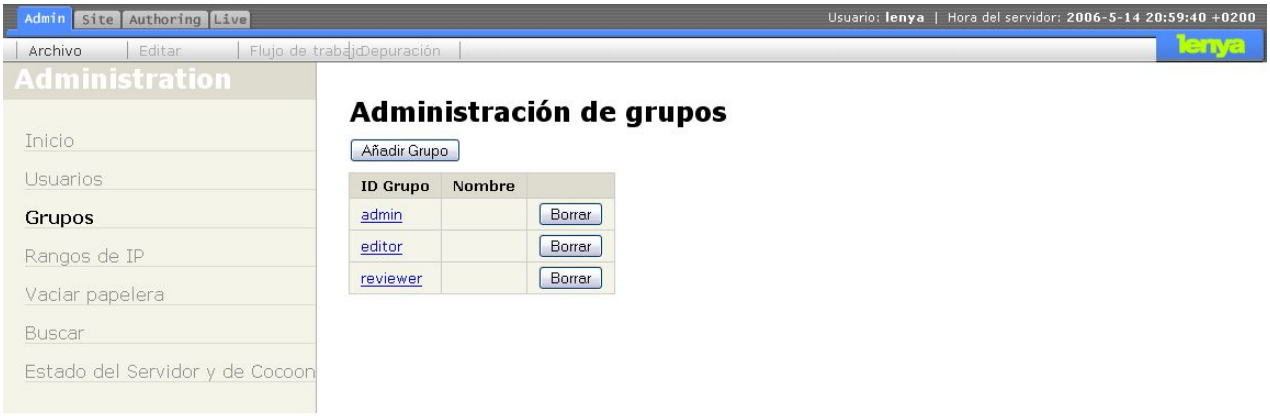

Figura 21: Interfaz de administración de Lenya

Como se puede comprobar, desde aquí se accede a todos los parámetros de gestión de usuarios, grupos, rangos de IP válidos, papelera y se puede lanzar la tarea de indexación de Lucene (para generar los índices para las búsquedas). Si hacemos click en la pestaña *site* accedemos a la pantalla mostrada en la Figura 22.

Figura 22: Interfaz de administración de las páginas de la publicación.

Desde esta interfaz somos capaces de cambiar el acceso para edición y producción de las páginas del sitio. También se pueden generar nuevos documentos o planificar la publicación, borrado, etc. Para obtener una mayor profundidad en el manejo de la interfaz de administración, consultar el Apéndice 6.2: Apéndice F: Manual de usuario.

4.5.3.Servicio de Edición.

El servicio de edición en Lenya tiene cuatro posibilidades distintas, tantas como editores vienen integrados por defecto en la distribución. Entre estos editores tenemos un editor de código fuente, un editor de formularios y dos editores gráficos (accesibles desde el navegador).

El editor de código fuente permite editar directamente el código que se almacenará en el archivo xml de la página. Este archivo implementa un recurso o tipo de documento y, por lo tanto, tiene asociado un esquema con una gramática RelaxNG. El editor de código fuente realiza una validación gramatical contra este esquema antes de proceder al guardado del documento. Si no se pasa la validación provocará un error.

El editor de formularios permite editar la página por bloques, cada uno de ellos delimitado por una etiqueta de apertura y otra de cierre. Por ejemplo, permitiría editar un bloque contenido entre dos etiquetas XHTML `<p>` y `</p>`. Este editor también permite introducir nuevos bloques desde una lista de elementos permitidos y, al igual que el editor de código fuente, realiza una validación del código generado antes de proceder al guardado del documento.

En cuanto a los editores gráficos, existen dos, Bitflux (BXE) y Kupu. Cada uno tiene sus ventajas e inconvenientes, y dependiendo del contenido a editar puede resultar más cómodo el uso de uno u otro. Sin embargo hay que tener en cuenta que Kupu permite sólo la edición de archivos XHTML, mientras que BXE permite más tipos de documentos. También hay que tener en cuenta que a veces, debido a deficiencias en la integración, cuando un documento se ha modificado con uno de estos dos editores da error la modificarlo con el otro.

4.5.4.Gestión y control de versiones.

Lenya posee mecanismos para la gestión y el control de versiones. Estos permiten bloquear un archivo cuando está siendo editado por otro usuario o volver a una versión anterior en caso de que fuera necesario.

Cada vez que un documento es publicado, Lenya guarda una copia de la última versión en `$LENYA_PUB/content/rcbak` y un histórico de modificaciones en `$LENYA_PUB/content/rcml`. Consultando los archivos contenidos en estas carpetas es capaz de conocer qué documento tiene que recuperar en el caso de que sea necesario.

4.5.5.Planificación.

Lenya posee también un servicio de planificación basado en el *QuartzScheduler* de Cocoon. Mediante este servicio se puede planificar la activación y la desactivación de contenidos para que se efectúen en un día y una hora determinadas. En estos momentos en la versión estable de Lenya (versión 1.2.5, recomendada para entornos de producción), los mecanismos de planificación solo se pueden asignar a un documento a la vez en un instante. Si es necesario marcar varios documentos para que se activen al mismo tiempo, es necesario ir haciéndolo uno a uno.

4.5.6.Motor de búsqueda.

El servicio de búsqueda en Lenya está proporcionado por Lucene. A partir de la versión 1.2.4 se ha rehecho y modificado para que la lista de resultados adquiera el estilo de la publicación en la que se encuentra.

Se puede acceder al motor de búsqueda desde cualquier página de la publicación que se encuentre en el entorno de producción, permitiendo la consulta acerca de una palabra o una frase.

Al tratarse de búsquedas realizadas mediante Apache Lucene, es necesario el indexado

de las páginas para poder realizar las consultas. Por el momento, Lenya no dispone de ningún mecanismo para realizar un indexado periódico. Tampoco posee ninguna posibilidad para el lanzamiento del reindexado cuando se publica un documento. Para solucionar el primer problema, se podría realizar una tarea programada (por ejemplo, mediante cron en linux) de forma que todos los días a las 12 de la noche se reindexara el portal para que los cambios realizados en el mismo pudieran contemplarse accesibles para las consultas de los usuarios. Para solucionar el problema del reindexado automático cuando se publica un documento, sería necesario crear un caso de uso nuevo que acometiera esta tarea.

De cualquier modo, Lenya sí que permite una reindexación manual de las páginas de la publicación. A ella se accede desde el menú de administración, haciendo click sobre el enlace a Buscar mostrado en la Figura 23.

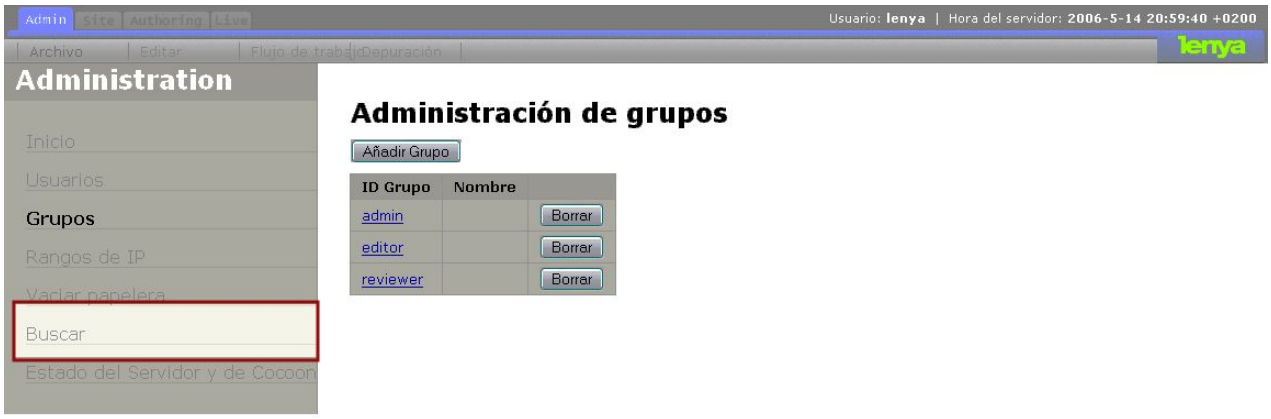

Figura 23: Acceso al servicio de indexación para las búsquedas

4.6.Personalización.

En este apartado trataremos de dar algunas directrices para realizar las tareas de personalización de las publicaciones. Estas tareas de personalización siempre son necesarias cuando se está realizando desarrollo web.

Cuando hablamos de personalización no nos referimos simplemente al aspecto visual de la publicación. Esto sería fácilmente realizable mediante la modificación de la hoja de estilos css asociada a la misma (ya que como hemos comentado, la presentación está separada del contenido). La diferenciación entre portales cuando hablamos de personalización va un poco más allá. Por ejemplo, imaginemos que tenemos un portal (una publicación) de noticias y otro que tiene un funcionamiento similar a un calendario de eventos con una zona de descarga.

Evidentemente, los requisitos de ambas publicaciones son bastante distintos. Puede que necesiten conexión a bases de datos o que los roles de las personas implicadas en el mantenimiento del mismo tengan características diferentes y, por supuesto, los esquemas de los formularios necesarios para completar la información a publicar no tienen nada que ver. En este caso, si queremos que Lenya se encargue de gestionar ambas publicaciones necesitamos moldearlo de forma que se consiga nuestro objetivo. Ése es el sentido que tiene la personalización sobre la que trata este apartado.

Todos los datos sobre personalización que siguen a continuación se comentan para la publicación que viene por defecto con Lenya. Esto es debido a que, al venir integrada en el paquete que se descarga, siempre estará en el mismo estado para todo el mundo que quiera comprobar las modificaciones descritas en los siguientes apartados. Suponiendo que Lenya se

ha desplegado dentro del contexto de Tomcat, la publicación por defecto se encuentra en `$TOMCAT_HOME/webapps/lenya/lenya/pubs/default`, si se ha desplegado como una aplicación normal, o en `$TOMCAT_HOME/webapps/ROOT/lenya/pubs/default` si se ha desplegado dentro del contexto raíz de Tomcat.

4.6.1.Apariencia de la publicación.

Para cambiar la apariencia de la publicación que viene por defecto en Lenya podemos comenzar por dos pasos simples (aplicable a las versiones 1.2 y 1.4):

1. Cambiar el esquema de color y las fuentes de las páginas.
2. Reemplazar la cabecera de la página.

Otros cambios podrían ser:

- Cambiar el menú de lugar.
- Usar un estilo diferente para el menú.
- Definir un esquema de página distinto al actual.

Como un primer paso, nos centraremos en los puntos 1 y 2. Las hojas de estilo que aplican el formato final a las páginas de contenido de Lenya se encuentran en `$LENYA_PUB/xslt` donde `$LENYA_PUB` identifica la carpeta raíz de la publicación. Modificando estas hojas de estilo se consigue modificar el esquema de la página para crear una composición diferente y no sólo la modificación de los colores y tipo de letra de la publicación actual. En estas hojas de estilo es donde se debe incluir el diseño de la publicación.

4.6.1.1.Cambiar las fuentes y los colores.

Las fuentes y los colores se definen mediante una hoja de estilo en cascada (CSS). Esta se encuentra en `$LENYA_PUB/resources/shared/css/page.css`.

Aunque en la ruta nos encontremos “resources/shared” eso no quiere decir que la hoja de estilo se comparta entre diferentes publicaciones, sino que se comparte entre todas las páginas de la misma publicación.

Si se realizan cambios en la hoja de estilo y se recarga la página en el navegador, se pueden comprobar los cambios realizados.

4.6.1.2.Cambiar la cabecera de la página.

Mirando dentro de $LENYA_PUB/xslt/page2xhtml.xsl nos encontramos con el siguiente trozo de código HTML.

Listado 4.1: Cambios en la cabecera de la página.

```
<table width="100%" cellpadding="0" cellspacing="0" border="0">
       <tr>
         <td id="publication-title">Welcome to the Default Publication!</td>

         <td id="project-logo"><img src="{$root}/images/project-logo.png"/></td>
       </tr>
</table>
```

Esta es la cabecera de la página. En ella podemos ver el título de la publicación (`publication-title`) y el logo de la misma (`project-logo`). Estos elementos son los que habría que cambiar para modificar la cabecera de la publicación.

4.6.2.Esquema de Workflow.

La definición del esquema de workflow de una publicación se encuentra en `$LENYA_PUB/config/workflow/`. La implementación del motor de workflow y de los esquemas es la misma para las versiones 1.2 y 1.4.

Un archivo de definición de esquema de workflow tiene el siguiente aspecto:

Listado 4.2: Esquema de workflow de Lenya.

```
<workflow xmlns="http://apache.org/cocoon/lenya/workflow/1.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://apache.org/cocoon/lenya/workflow/1.0
        ../../../resources/entities/workflow/workflow.xsd">

  <state id="authoring" initial="true"/>
  <state id="live"/>
  <state id="trash"/>
  <state id="archive"/>

  <variable name="is_live" value="false"/>

  <transition source="authoring" destination="authoring">
    <event id="edit"/>
    <condition class="org.apache.lenya.cms.workflow.RoleCondition">
      edit, review, organize
    </condition>
  </transition>

  <transition source="authoring" destination="authoring">
    <event id="deactivate"/>
    <condition class="org.apache.lenya.workflow.impl.BooleanVariableCondition">
      is_live = true
    </condition>
    <condition class="org.apache.lenya.cms.workflow.RoleCondition">
      review, organize
    </condition>
    <assign variable="is_live" value="false"/>
  </transition>

  ...

</workflow>
```

Para Lenya, la URI correspondiente al espacio de nombres del esquema de workflow es `http://apache.org/cocoon/lenya/workflow/1.0`

4.6.2.1.Estados

Todos los estados que se usen en el esquema de workflow tienen que ser declarados mediante los elementos `<state>`. El estado inicial se marca con el atributo `initial=true`.

4.6.2.2.Variables

Todas las variables usadas tienen que ser declaradas mediante los elementos `<variable>`. El valor inicial de la variable se asigna mediante el atributo `value`.

4.6.2.3.Transiciones

Una transición se declara mediante el uso de los elementos `<transition>`. Estos elementos requieren dos atributos, `source` y `destination`, que denotan los estados que están conectados por esta transición.

Los elementos `<transition>` deben contener un elemento `<event>` con un atributo `id`. Más allá, pueden contener un número arbitrario de elementos `<condition>` y `<assign>`.

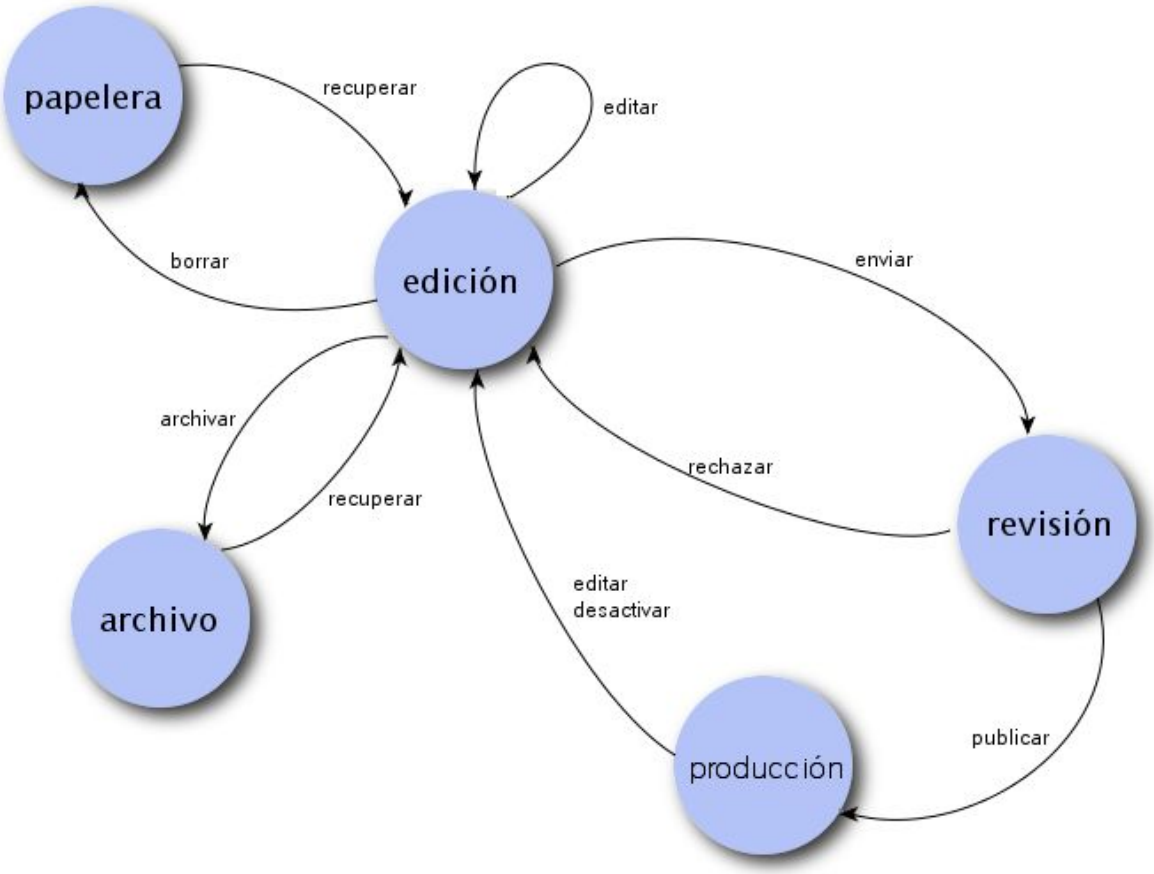

Figura 24: Esquema de workflow por defecto de Lenya.

Un elemento `<transition>` puede tener un atributo `synchronized="true"`. En este caso, si la transición se lanza usando una *SynchronizedWorkflowInstance*, se invoca en todas las instancias.

4.6.2.4.Asignaciones de variables

Una asignación de un valor a una variable vendría dada por una sentencia de la forma `<assign variable="..." value="..."/>`

Esta variable debe haber sido declarada en el esquema de workflow. Debido a que las variables sólo soportan valores booleanos, el valor de la misma debe ser `true` o `false`.

4.6.2.5.Condiciones

Una condición tiene la forma `<condition class="...">...</condition>`

El atributo clase contiene el nombre completo (incluyendo el paquete en el se se haya) de la clase que implementa la condición. Se pueden usar las clases para las condiciones que vienen con Lenya o implementar unas personalizadas. Todas las clases de condiciones deben implementar el interfaz `org.apache.lenya.workflow.Condition`. El texto dentro del elemento es la expresión que debe ser evaluada. Se le pasa como argumento al método `setExpression()`.

4.6.2.5.1.BooleanVariableCondition

La clase `org.apache.lenya.workflow.impl.BooleanVariableCondition` requiere una expresión de la forma `{variable-name} = {value}`, donde `{variable-name}` es el nombre de la variable que se declaró en el esquema de workflow y `{value}` tiene el valor `true` o `false`.

4.6.2.5.2.RoleCondition

La clase `org.apache.lenya.cms.workflow.RoleCondition` requiere una lista de identificadores de roles separada por comas: `{role-id-1}, {role-id-2}, ...`

La condición se completa cuando la identidad actual tiene alguno de esos roles para la URL solicitada.

4.6.3.Elementos de navegación.

La publicación por defecto de Lenya viene con un marco de navegación que se compone de pestañas horizontales, miga de pan y un menú a la izquierda. Estos elementos son muy comunes en los diseños de las páginas de internet, pero dependiendo del diseño del sitio web que se esté construyendo se puede querer cambiar el comportamiento de la navegación o su estructura. Se puede encontrar información sobre el marco de navegación de Lenya en [LENYANAV2006] (aplicable a la versión 1.2).

4.6.3.1.Generación de la navegación en Lenya.

La fuente para todos los elementos de navegación es es *sitetree*, un archivo xml que ya hemos comentado en el apartado 4.4.1. Éste puede encontrarse en `$LENYA_PUB/authoring/sitetree.xml`, `$LENYA_PUB/live/sitetree.xml`, `$LENYA_PUB/trash/sitetree.xml` o en `$LENYA_PUB/archive/sitetree.xml` dependiendo del área en el que nos encontremos.

El documento *sitetree.xml* de la publicación por defecto que usaremos como ejemplo tiene el siguiente formato:

Listado 4.3: Archivo *sitetree.xml*.

```
<?xml version="1.0" encoding="UTF-8"?>
<site i18n:attr="label" label="Authoring"
    xmlns="http://apache.org/cocoon/lenya/sitetree/1.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation=
        "http://apache.org/cocoon/lenya/sitetree/1.0
        ../../../../resources/entities/sitetree.xsd"
    xmlns:i18n="http://apache.org/cocoon/i18n/2.1">

  <node id="index">
    <label xml:lang="en">Home</label>

    <label xml:lang="de">Home</label>
  </node>

  <node id="tutorial">
    <label xml:lang="en">Tutorial</label>

    <label xml:lang="de">Tutorial</label>

    <node id="new_doctype">
      <label xml:lang="en">Create new doctype</label>
    </node>

  </node>

  <node id="concepts">
    <label xml:lang="en">Concepts (english only)</label>
  </node>
```

```
    <node id="features">

      <label xml:lang="en">Features</label>
      <label xml:lang="de">Funktionen</label>
    </node>

    <node id="doctypes">

      <label xml:lang="en">Document Type Examples</label>
      <label xml:lang="de">Dokumenttyp Beispiele</label>
       <node id="xhtml-document">
        <label xml:lang="en">XHTML Doctype</label>

        <label xml:lang="de">XHTML Dokumenttyp</label>
      </node>
   </node>

  </site>
```

En la publicación por defecto se definen tres elementos de navegación:

- Pestañas (horizontal).
- Miga de pan (ruta de la página desde la raíz).
- Menú (izquierda).

Cada uno de ellos se construye a partir de la aplicación de una hoja XSLT específica al archivos *sitetree.xml*. En caso de necesitar elementos de navegación adicionales, se pueden generar a partir de una hoja XSLT adicional integrándola en el *sitemap*. Trataremos este punto más adelante.

4.6.3.2.Pestañas y menús.

Dependiendo de la cantidad de contenido del sitio, se puede optar por diferentes estilos de navegación. Se podría, por ejemplo, prescindir de las pestañas y sólo dejar el menú. En cambio, en la publicación por defecto, estos elementos se usan en paralelo. Si se tiene una gran cantidad de contenido organizada en múltiples niveles, se podría elegir usar las pestañas para el primer nivel de navegación y el menú para un segundo nivel.

Si, por ejemplo, tenemos un *sitetree* con esta estructura:

- Home
- Products
    - Consumer Products
        - Product Line A
        - Product Line B
        - Product Line C
    - Industrial Products
- Services
    - Maintainance Contracts
    - 24-hour Emergency Hotline

un esquema típico de navegación podría ser:

```
+-------------------------------------------------------------------
|                        | *HOME* | Products | Services |
+-------------------------------------------------------------------
```

cuando el usuario hace click en Producs, se muestra es siguiente menú:

```
+-------------------------------------------------------------------------
|                          | Home | *PRODUCTS* | Services |
+-------------------------------------------------------------------------
|
| - Consumer Products
| - Industrial Products
|
```

si se hace click en Services se mostrará un menú diferente:

```
+-------------------------------------------------------------------------
|                          | Home | Products | *SERVICES* |
+-------------------------------------------------------------------------
|
| - Maintainance
|   Contracts
| - 24-hour Emergency
|   Hotline
```

4.6.3.3.Solución simple: esconder el menú usando CSS.

Este comportamiento se puede conseguir mediante la edición del CSS de la publicación por defecto. Habría que editar el archivo `$PUB_HOME/resources/shared/css/page.css` y añadir el siguiente trozo de código, preferiblemente debajo de las declaraciones existentes de *menublock*:

Listado 4.4: Esconder el menú con CSS.

```
.menublock-1 {
    display: none;
}
```

De esta forma, todas las entradas del menú de primer nivel que no estén seleccionadas se encuentran ocultas, lo que comprende todas las entradas del menú que pertenecen a las otras pestañas.

4.6.3.4.Solución avanzada: Excluir los menús usando XSLT.

Mediante esta aproximación, se usa una hoja XSLT personalizada para generar el trozo de código XHTML de la barra de menú. Ese archivo debería estar situado en `$PUB_HOME/lenya/xslt/navigation/menu.xsl`.

Si nos fijamos bien en la ruta nos encontramos conque existe un `/lenya`. Esto es necesario para decirle a Lenya que se está sobreescribiendo un archivo del núcleo. La hoja de estilo podría tener esta forma.

Listado 4.5: Excluir el menú con XSLT.

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:nav="http://apache.org/cocoon/lenya/navigation/1.0"
    xmlns="http://www.w3.org/1999/xhtml"
    exclude-result-prefixes="nav"
    >

  <xsl:import href="../../../../../xslt/navigation/menu.xsl"/>

  <xsl:template match="nav:site/nav:node[not(descendant-or-self::nav:node[@current =
'true'])]"/>

  <xsl:template match="nav:site/nav:node[descendant-or-self::nav:node[@current =
```

```
'true']]">
          <div class="menublock-selected-{count(ancestor-or-self::nav:node)}">

            <xsl:apply-templates select="nav:node"/>
          </div>
        </xsl:template>

      </xsl:stylesheet>
```

- En primer lugar, se importa el archivo XSL del menú del núcleo, para que de esta forma pueda reusarse la mayoría de la funcionalidad del mismo.
- La primera etiqueta <xsl:template> excluye todos los elementos <nav:node> de primer nivel que no contengan el elemento actual.
- La segunta etiqueta <xsl:template> es opcional. Si se incluye, los elementos de primer nivel no serán mostrados en la barra del menú.

4.6.4.Adición de nuevos tipos de documentos.

En este apartado intentaremos explicar cómo añadir un tipo de documento personalizado (también llamado recurso en Lenya) a la publicación por defecto para permitir mezclar la edición libre de las páginas (tal y como en XHTML) y la edición restringida de las mismas (que se consigue con el tipo de documento personalizado). En los siguientes puntos daremos los pasos que se deberían seguir para la adición exitosa de un nuevo tipo (aplicable a la versión 1.2).

4.6.4.1.Decidir si realmente se necesita un nuevo recurso.

La introducción de un nuevo tipo de documento donde no se necesita realmente consume bastante tiempo y, a menudo, es un ejercicio innecesario. Como regla simple, se debería usar un nuevo recurso si se va a intentar renderizar documentos que, en su base, son distintos a los XHTML. Un tipo de recurso es necesario cuando, por ejemplo, se va a convertir un archivo XML arbitrario a XHTML que después será introducido en la página de la publicación.

Tiene sentido, pues, introducir nuevos tipos de documentos para esquemas XML bien conocidos tales como:

- RSS
- DocBook
- NewsML, SportsML, NITF or IPTC
- FOAF

También tiene sentido por lo tanto, introducir un nuevo recurso para esquemas XML personalizados como podrían ser el formato XML del formato de un catálogo o el XML que describe los posibles servicios que ofrece una compañía.

Por el contrario, no es una buena idea introducir un tipo de documento personalizado si se quieren renderizar documentos que son básicamente XHTML pero que contienen algunas etiquetas no-XHTML tales como:

- <xi:include ...>
- <ft:...> (etiquetas de Cocoon Forms)

En este caso, lo más adecuado sería añadir algunos transformadores extra a la *pipeline* que renderizan esas etiquetas a XHTML. También es importante indicar que los tipos de documentos no son plantillas XHTML.

4.6.4.2.Elegir un nombre.

Se debe elegir un nombre para el recurso que no esté siendo utilizado para otro dentro de la misma publicación. Usaremos como ejemplo *myresourcetype* como nombre para nuestro nuevo tipo de documento.

4.6.4.3.Definir la ID del tipo de documento.

Es necesario definir el nombre del tipo de documento en `mypub/config/doctypes/doctypes.xconf`.

Listado 4.6: Definiendo el tipo de documento.

```
      <doctypes>
        <!-- Aquí iría la ID del nuevo recurso -->
        <doc type="myresourcetype">
          <!-- si se permite al documento tener hijos, aquí se definen los tipos de
documento que pueden tener -->
          <children>
            <!-- a este tipo de recurso se le permite tener documentos xhtml y/o
myresourcetype como hijos -->

            <doc type="xhtml"/>
            <doc type="myresourcetype"/>
          </children>
          <!-- Esta es la clase que crea una nueva instancia del este recurso -->
          <creator src="org.apache.lenya.cms.authoring.DefaultBranchCreator">
            <!-- este es el archivo de ejemplo que se usa cuando se crea una nueva instancia
de este tipo de documento. Se puede encontrar en config/doctypes/samples -->

            <sample-name>myresourcetype.xml</sample-name>
          </creator>
          <!-- Las tareas descritas aquí hacen referencia a las tareas ant que se deberían
llamar para este recurso. Por ejemplo, podría necesitarse una tarea específica para la
publicación. -->
          <!-- Las tareas se definen en mypub/config/tasks/tasks.xconf -->
          <tasks>

            <task id="publish"/>
          </tasks>
          <!-- Especificar la definición de workflow para este tipo de documento. Se puede
definir un esquema de workflow separado para cada tipo de recurso, por ejemplo, con uno o
dos niveles para la aprobación. -->
          <!-- el proceso del workflow se define en  mypub/config/workflow/workflow.xml -->
          <workflow src="workflow.xml"/>
        </doc>

      </doctypes>
```

4.6.4.4.Crear el esquema.

Es necesario crear el esquema RelaxNG: `mypub/config/doctypes/schemas/myresourcetype.rng`

Este paso es opcional pero importante si se quieren utilizar los editores de formularios. Si se toman los XML que se quieren renderizar para el recurso personalizado desde otro lugar u editor, se puede omitir este paso. También se puede saltar este paso si se quiere crear un documento de ejemplo de forma manual para comprobar el funcionamiento del resto de proceso del recurso. Una vez comprobado, se puede volver a este punto para completarlo.

Listado 4.7: Esquema RNG

```
      <?xml version="1.0" encoding="UTF-8"?>
      <!--+
          | Tipo de documento: myresourcetype
          +-->

      <grammar ns="http://www.w3.org/1999/xhtml"
```

```
                xmlns="http://relaxng.org/ns/structure/1.0"
                xmlns:lenya="http://apache.org/cocoon/lenya/page-envelope/1.0"
                xmlns:dc="http://purl.org/dc/elements/1.1/"
                xmlns:dcterms="http://purl.org/dc/terms/"
                xmlns:xhtml="http://www.w3.org/1999/xhtml"
                >

      <!-- Incluir el esquema general de Lenya -->
      <!-- Se debe incluir el esquema de lenya si se quieren usar metadatos o activos
(documentos, etc.) -->
      <include href="lenya.rng"/>

      <!-- Incluye el esquema XHTML transicional original -->
      <include href="xhtml/xhtml-basic.rng">

        <define name="html">

          <element name="html">
            <ref name="html.attlist"/>
            <ref name="lenya.meta"/> <!-- este es el envoltorio para los metadatos de
lenya-->
            <ref name="head"/>
            <ref name="body"/>

          </element>
        </define>

        <define name="html.attlist">
          <ref name="XHTML.version.attrib"/>
          <ref name="I18n.attrib"/>
          <ref name="dummy.attlist"/> <!-- esto está deprecated -->

        </define>

      </include>


      <!-- Bloques de elementos adicionales -->
      <define name="Block.class" combine="choice">
        <choice>
          <ref name="lenya.asset"/>

        </choice>
      </define>


      </grammar>
```

Para la creación de los esquemas RelaxNG se puede usar, por ejemplo, las herramientas Trang o Sun RelaxNG converter.

4.6.4.5.Añadir el matcher para el nuevo tipo de documento.

Hay que añadir el *matcher* para el recurso en `mypub/parameter-doctype.xmap`.

Listado 4.8: Añadir el matcher.

```
      <!-- Este archivo se monta mediante map:mounted desde
           mypub/publication-sitemap.xmap -->

      <map:action name="sourcetype"
src="org.apache.cocoon.acting.sourcetype.SourceTypeAction">
        <sourcetype name="myresourcetype">

          <document-element local-name="myroottag"/>
          <!-- this matches the root tag -->
        </sourcetype>
      </map:action>
```

El archivo *parameter-doctype.xmap* permite a Lenya determinar el tipo de recurso de cualquier URI. El mapeo que se provee aquí permite mezclar y capturar los tipos de documentos de forma simple. Hace uso de la *SourceTypeAction* [FORRESTSOURCE2006]. Hay varias formas de identificar un tipo de recurso, como puede ser por su etiqueta raíz o el espacio de nombres de su elemento raíz o de su esquema. Hay que indicar que el nombre del tipo de recurso que se especifica aquí se usa a través de Lenya como una convención de nombre, por ejemplo para la hoja *resourcetype2xhtml.xsl.*

La mayoría de los problemas cuando un tipo de documento no se renderiza de forma correcta resultan porque este *matcher* no captura el tipo de recurso y, por consiguiente, utiliza el definido por defecto.

4.6.4.6.Archivo de ejemplo.

Añadir un archivo de ejemplo: `mypub/config/doctypes/samples/myresourcetype.xml.`

Este ejemplo se usará como una plantilla. Si se crea un nuevo documento con este recurso, se utilizará una copia del archivo de ejemplo como punto de partida. Debido a esto, debería escribirse el contenido de este archivo de forma adecuada, ya que cualquier persona que vaya a editar o crear una nueva página con este tipo de recurso verá el contenido de este archivo de ejemplo.

Listado 4.9: Archivo de ejemplo.

```
      <?xml version="1.0" encoding="UTF-8"?>
      <html xmlns="http://www.w3.org/1999/xhtml"
            xmlns:xhtml="http://www.w3.org/1999/xhtml"
            xmlns:dc="http://purl.org/dc/elements/1.1/"
            xmlns:dcterms="http://purl.org/dc/terms/"
            xmlns:lenya="http://apache.org/cocoon/lenya/page-envelope/1.0"
            xhtml:dummy="FIXME:keepNamespace" dc:dummy="FIXME:keepNamespace"
            lenya:dummy="FIXME:keepNamespace" dcterms:dummy="FIXME:keepNamespace">
          <lenya:meta>
              <dc:title>dctitle</dc:title>

              <dc:creator>dccreator</dc:creator>
              <dc:subject>dcsubject</dc:subject>
              <dc:description>Una página vacía para un nuevo tipo de
recurso</dc:description>
              <dc:publisher/>

              <dc:contributor/>
              <dc:date>2004-4-6</dc:date>
              <dc:type/>
              <dc:format/>
              <dc:identifier/>

              <dc:source/>
              <dc:language>en</dc:language>
              <dc:relation/>
              <dc:coverage/>
              <dc:rights>dcrights</dc:rights>

          </lenya:meta>
        <head>
          <title>Ejemplo de nuevo tipo de documento</title>
        </head>
        <body>

          <h1>Ejemplo </h1>
          <p>Esta página de ejemplo es la base para las páginas definidas por este
recurso</p>
        </body>
      </html>
```

4.6.4.7.Presentación.

Hay que crear las hojas de presentación XSLT y CSS. Si son específicas a la publicación,

deberían guardarse de forma respectiva en `mypub/xslt/myresourcetype2xhtml.xsl` y en `mypub/resources/shared/css/myresourcetype.css`. Hay que tener en cuenta las convenciones de nombre para los archivos XSLT.

Copiar `mypub/xslt/page2xhtml-xhtml.xsl` a `mypub/xslt/page2xhtml-myresourcetype.xsl`. Esto le aplicará el mismo estilo al nuevo tipo que el que tenían los documentos viejos. Si se desea cambiar el aspecto del nuevo documento, se debe crear una hoja xsl en *`mypub/xslt/page2xhtml-myresourcetype.xsl`*.

Si se desea editar los nuevos documentos usando BXE (ya que kupu sólo edita XHTML), se debería colocar un archivo CSS para el estilo de los documentos en `mypub/resources/misc/myresourcetype-bxeng.css`.

Por último, es necesario añadir las *pipelines* de presentación en *mypub/sitemap.xmap* si se necesitan *pipelines* especiales para este tipo de documento.

Es necesario comprender que para un nuevo tipo de documento no es necesario reemplazar el archivo page2xhtml.xsl sino el xhtml2xhtml. La razón es que la transformación de un documento en una página en el navegador siempre es un proceso que se realiza en dos pasos:

1. Usando *{resourcetype}2xhtml.xsl* el documento (que podría ser un documento XML arbitrario como podría ser una lista de enlaces, una fuente RSS, etc.) se convierte para una presentación mediante XHTML.
2. Este XHTML es entonces procesado para añadirle la navegación, los menús y todo lo demás usando *page2xhtml.xsl.*
3. La hoja *{resourcetype}2xhtml.xsl* debe devolver un documento de la forma * ... *.

Si tenemos 5 recursos distintos en la publicación, por ejemplo:

- rss
- linklist
- gallery
- docbook
- xhtml

los pasos indicados anteriormente seguirían el siguiente esquema:

```
documento -> rss2xhtml.xsl       --+
documento -> linklist2xhtml.xsl  --+
documento -> gallery2xhtml.xsl   --+---> page2xhtml.xsl -> Navegador
documento -> docbook2xhtml.xsl   --+
documento -> xhtml2xhtml.xsl     --+
```

4.6.4.8.Configurar los menús.

Hay dos aproximaciones, usar un menú existente o usar un menú personalizado. En el apartado 4.6.5 daremos más información sobre cómo modificar los menús.

4.6.4.8.1.Usar un menú existente.

En este apartado usaremos un menú ya existente y le añadiremos los elementos necesarios. Para ello es necesario editar el archivo `mypub/config/menus/generic.xsp` y añadir una entrada para el nuevo tipo de documento.

Listado 4.10: Añadiendo entradas al menú.

```
<menus>
  <menu i18n:attr="name" name="File" label="File">
    <block>
      <xsp:logic>
        {
            if (Publication.ARCHIVE_AREA.equals(area) ||
                  Publication.TRASH_AREA.equals(area)) {
                        <item><i18n:text>New Document</i18n:text></item>

            }
            else {
                        <item uc:usecase="create" uc:step="showscreen"
                        href="?doctype=xhtml"><i18n:text>New
                        Document</i18n:text></item>
                        <item uc:usecase="create" uc:step="showscreen"
                        href="?doctype=myresourcetype">New "Doc type"
                        Document</item>
            }
        }
      </xsp:logic>

    </block>
...
```

4.6.4.8.2.Usar un menú personalizado.

Para usar o crear un menú personalizado para este tipo de recurso es necesario editar el archivo `mypub/menus.xmap` y crear un nuevo archivo de menú en `mypub/config/menus/myresourcetype.xsp`.

4.6.4.9.Workflow

Este paso también es opcional. Aquí se podría definir un esquema de workflow específico para el nuevo recurso. Para ello es necesario:

- Añadir el esquema de workflow a *`mypub/config/workflow/`*
- Asignar el esquema de workflow al tipo de recurso en `mypub/config/doctypes/doctypes.xconf`, como ya hemos indicado anteriormente.

4.6.5.Modificación de los menús del CMS.

Los menús del CMS son los menús de opciones de Lenya que sólo se encuentran accesibles en el modo de edición (*authoring*) y que se usan por el editor, el revisor o el administrador para lanzar operaciones como pueden ser la publicación, el borrado o la edición de una página. Los menús del CMS son completamente diferentes a los menús de navegación de la publicación. La navegación por el sitio es lo que el visitante podrá ver en la vista de producción, mientras que los menús CMS sólo son visibles en el modo de edición (aplicable a la versión 1.2 y 1.4).

La mayoría de las veces los menús del CMS pueden verse como una parte inamovible de la aplicación que simplemente funciona. Pero en ocasiones es necesario introducir opciones de menú adicionales si la publicación tiene alguna operación especial a la que es necesario acceder mediante una opción del menú, como en el caso de la inclusión del tipo de recurso nuevo comentada anteriormente.

También puede darse el caso de que nos encontremos alguna opción del menú deshabilitada cuando creíamos que no iba a estarlo. En ese caso es necesario también

entender la anatomía de los menús para depurar el problema. Una vez que se ha usado el archivo de definición de los menús, se procesa mediante dos transformadores implementados directamente en la capa de Java de Lenya:

Listado 4.11: Transformadores para el menú.

```
<map:transformer name="workflowmenu" logger="lenya.sitemap.transformer.workflowmenu"
        src="org.apache.lenya.cms.cocoon.transformation.WorkflowMenuTransformer"/>

<map:transformer name="usecasemenu" logger="lenya.sitemap.transformer.usecasemenu"
        src="org.apache.lenya.cms.cocoon.transformation.UsecaseMenuTransformer"/>
```

Estos transformadores tienen la misión de deshabilitar las opciones del menú si no son accesibles debido a que:

- La ejecución del Caso de Uso requiere un rol que no posee el usuario perteneciente a la sesión actual.
- El estado actual del workflow del documento prohibe el caso de uso.

4.6.5.1. Introducir una nueva opción en el menú.

Antes de comenzar a añadir la nueva opción al menú del CMS, es necesario saber qué caso de uso debería unirse a esa opción del menú y si ese caso de uso existe ya. Si se encuentra que se necesita un nuevo caso de uso, es mejor implementarlo primero y probarlo de forma manual mediante su invocación con la URI apropiada. Es importante recalcar que para activar la nueva función para el usuario en el menú es necesario tener tanto el caso de uso como la opción en el mismo.

Los menús del CMS se definen en un archivo XML como el siguiente:

Listado 4.12: Añadir una nueva opción en el menú de la página.

```
<menu
    xmlns="http://apache.org/cocoon/lenya/menubar/1.0"
    xmlns:uc="http://apache.org/cocoon/lenya/usecase/1.0"
    xmlns:wf="http://apache.org/cocoon/lenya/workflow/1.0">

  ...

  <item wf:event="edit"
        uc:usecase="edit-forms"
        uc:step="open" href="?...">Edit with Forms</item>

  ...


</menu>
```

4.6.5.2. Editar el archivo generic.xsp para añadir la opción del menú.

En la práctica, este XML se genera de un XSP que se puede encontrar en `$LENYA_PUB/config/menus/generic.xsp`. Para añadir el nuevo elemento al menú, hay que editar este XSP. Este XSP se usa en la siguiente sección del *global-sitemap.xmap*.

Listado 4.13: Edición de *generic.xsp*.

```
<map:match pattern="lenya-page/*/*/**">
  <map:generate src="cocoon:/menu-xml/{1}/{2}/{3}"/>

  <map:call resource="i18n"/>

  <map:act type="resource-exists" src="lenya/pubs/{1}/config/workflow/">
    <map:transform type="workflowmenu"/>

  </map:act>
```

```
              <map:act type="resource-exists" src="lenya/pubs/{1}/config/ac/usecase-
policies.xml">
                <map:transform type="usecasemenu"/>
              </map:act>

              <map:transform src="lenya/xslt/menu/menu2xhtml.xsl">

                <map:parameter name="contextprefix" value="{request:contextPath}"/>
                <map:parameter name="publicationid" value="{1}"/>
                <map:parameter name="completearea" value="{2}"/>
                <map:parameter name="documentarea" value="{page-envelope:area}"/>
                <map:parameter name="documenturl" value="{page-envelope:document-url}"/>
                <map:parameter name="documentid" value="{page-envelope:document-id}"/>

                <map:parameter name="userid" value="{access-control:user-id}"/>
                <map:parameter name="servertime" value="{date-i18n:currentDate}"/>
                <map:parameter name="workflowstate" value="{workflow:state}"/>
                <map:parameter name="islive" value="{workflow:variable.is_live}"/>
              </map:transform>

              <map:call resource="i18n"/>

              <map:transform src="lenya/xslt/menu/menu2xslt.xsl">
                <map:parameter name="contextprefix" value="{request:contextPath}"/>
                <map:parameter name="publicationid" value="{1}"/>
                <map:parameter name="area" value="{2}"/>
                <map:parameter name="documenturl" value="{page-envelope:document-url}"/>
              </map:transform>

              <map:serialize type="xml"/>
            </map:match>
```

La *pipeline* `<map:generate src="cocoon:/menu-xml/{1}/{2}/{3}"/>` es interna (no se puede acceder directamente a ella mediante una URI) y se puede encontrar en el mismo archivo:

Listado 4.14: *Pipeline* interna.

```
            <!-- menu-xml/{publication-id}/... -->

            <map:match pattern="menu-xml/*/**" internal-only="true">
              <map:mount uri-prefix="menu-xml/{1}/" src="lenya/pubs/{1}/menus.xmap"
                         check-reload="true" reload-method="synchron"/>
            </map:match>
```

Este *match* monta el *sitemap* `$LENYA_PUB/menus.xmap` de la publicación, que es el que decide si estamos en el modo de producción o edición y genera la definición del menú desde el archivo menu.xsp específico de la publicación:

Listado 4.15: Sección de generación del menú en *menus.xmap*.

```
            <map:match pattern="live/**">
              <map:generate type="serverpages" src="../../content/menus/live.xsp"/>
              <map:serialize type="xml"/>
            </map:match>

            <map:match pattern="**">

              <map:generate type="serverpages" src="config/menus/generic.xsp"/>
              <map:serialize type="xml"/>
            </map:match>

          </map:pipeline>
```

4.6.5.3.Revisar la configuración de los casos de uso y el workflow.

Si se observa el *matcher* del *global-sitemap.xmap* que captura la URI *lenya-page/*/*/***, los menús se construyen mediante:

- La generación de la definición del menú a través del XSP
- La aplicación de los transformadores *workflowmenu* y *usecasemenu*
- El uso de `$LENYA_PUB/lenya/xslt/menu/menu2xhtml.xsl` para la generación del XHTML

Esto significa que si se añade una nueva opción al XSP, debería mostrarse en en menú, pero puede que se encuentre deshabilitada.

Si este es el caso, o bien el transformador *workflowmenu* o bien el *usecasemenu* han decidido en base a la configuración o en base a la política de los casos de uso, que esta opción no está disponible para este usuario en este momento. Se debería revisar la configuración para ver si este es el caso. Para ello podemos ver el archivo de configuración de los casos de uso que se encuentra en `$LENYA_PUB/config/ac/usecase-policies.xml`.

4.7.Apache Lenya 1.4.

Apache Lenya 1.4 es la nueva versión del Gestor de Contenidos. Lleva algunos retrasos en su liberación debido a complicaciones surgidas a raíz de la integración del repositorio Jackrabbit (implementación del *Java Content Repository* según la recomendación JSR-170).

Esta versión ha supuesto algunos cambios respecto a la 1.2. Se han añadido las siguientes características:

- posibilidad de creación de plantillas de publicación
- un nuevo *framework* para la creación y gestión de los casos de uso
- inclusión de la interfaz para el repositorio
- compatibilidad (limitada por el momento) con el protocolo WebDAV
- agrupación de funcionalidades dentro de los denominados Módulos

Todos estos cambios persiguen que Lenya siga en un futuro la arquitectura de CMS mostrada en la Figura 25.

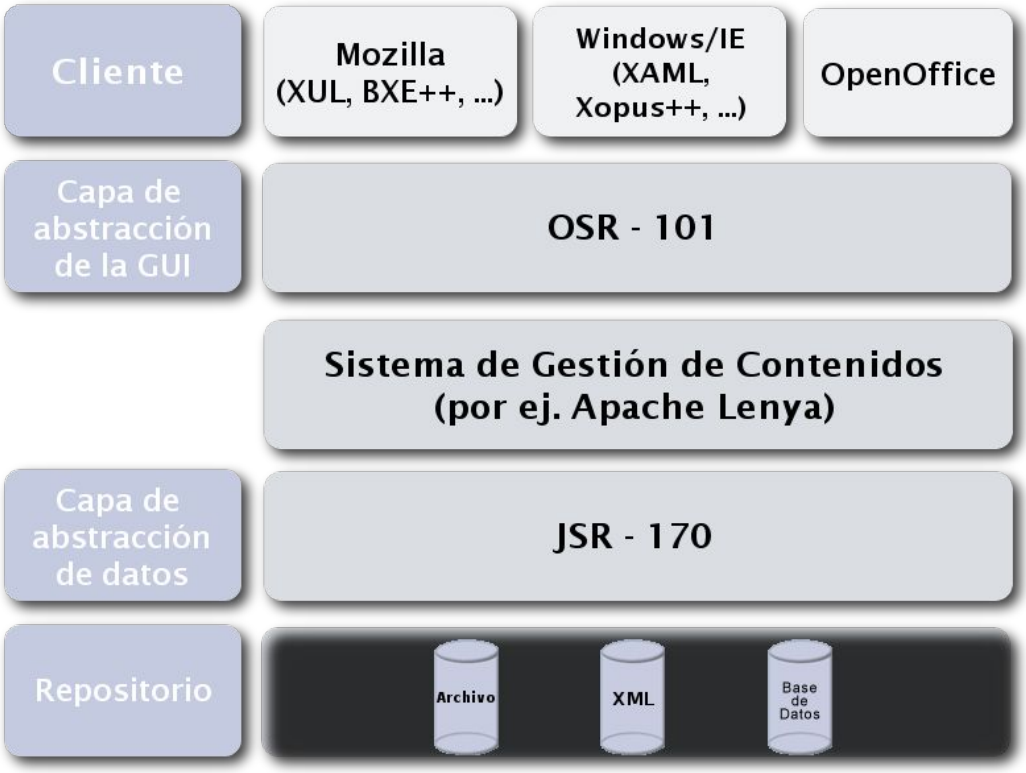

Figura 25: Futura arquitectura de Lenya.

En esta figura se observa una separación clara entre los distintos niveles de funcionamiento del CMS. Desde el punto de vista del cliente, se quiere conseguir una modificación del contenido mediante el uso de aplicaciones externas, en lugar de que se haga a través de la interfaz web aunque ésta seguirá estando disponible. Las peculiaridades de la implementación del lado del servidor se verán ocultas tras la interfaz de abstracción del OSR-101.

Desde el punto de vista del servidor, el motor de Lenya seguirá controlando todas las tareas relacionadas con la gestión del contenido y utilizará la interfaz de abstracción de datos proporcionada por el JSR-170 para evitar las peculiaridades del modo de almacenamiento del contenido.

En los siguientes apartados entraremos en más detalle en los cambios que se han realizado.

4.7.1.Plantillas de publicación.

Proveen un mecanismo para compartir propiedades entre publicaciones. Cada publicación puede ser usada como una plantilla para una nueva publicación, lo que provoca que se produzcan publicaciones hijas. Si se utiliza el caso de uso “Nueva Publicación”, todas las publicaciones se basarán en la que viene por defecto con Lenya.

El mecanismo de plantillas se implementa mediante el protocolo *fallback://*, un protocolo específico creado para Lenya que permite resolver las URIs de una manera especial. Si se ha realizado de forma correcta, las publicaciones pueden compartir cualquier propiedad (archivos xslt, archivos de configuración, usuarios, grupos, etc.) de la plantilla de la que heredan o de la publicación por defecto. El mecanismo de *fallback* opera a nivel de archivo. Por lo tanto sólo puede aplicarse a archivos completos y sólo si se ha hecho referencia a esos archivos mediante URIs en los archivos de configuración. La creación de una nueva publicación hija a partir de una plantilla se denomina instanciación.

Las publicaciones hijas pueden usar características de sus plantillas de dos formas:

- Copiando los archivos de la plantilla durante la instanciación
- Referenciando los archivos

La primera opción provoca que cuando se realice algún cambio en la plantilla, los cambios no afectarán a la publicación hija. La segunda implica que una modificación de la plantilla afectará de forma inmediata a la publicación hija, siempre que ésta use la propiedad de la plantilla que ha sido modificada. [LENYAPUBTEM2006]

4.7.2.Módulos.

Los módulos permiten implementar funcionalidades transversales (pueden utilizarse por varias publicaciones), de forma que pueden ser añadidos o utilizados dentro de una publicación. También se pueden realizar módulos que ofrezcan un conjunto de recursos y las herramientas necesarias para manejarlos.

Una publicación puede usar varios módulos para agregar y construir un sitio web y su contenido.

Un módulo usa la API de Lenya para acceder al workflow y a otros servicios del CMS para poder realizar sus tareas y sus servicios de contenido.

En Lenya 1.4 se encuentran implementados los siguientes módulos:

- Lucene: mecanismos de búsqueda
- Sitetree: manejo de documentos en forma de árbol
- JCR: almacenamiento del contenido en un repositorio
- XHTML: tipo de recurso basado en XHTML
- Links: tipo de recurso para el manejo de listas de enlaces
- Lenyadoc: añade el protocolo *lenyadoc://*
- ODT: tipo de recurso para manejo de archivos OpenOffice 2

4.7.3.Marco de Casos de Uso (*UseCase Framework*).

El Marco de los Casos de Uso en Lenya 1.4 intenta construir una herramienta que permita la creación y tratamiento de un caso de uso de forma simple y estandarizada. La implementación del marco en Lenya 1.4 se basa en un *sitemap* (*usecase.xmap*) que cede el control a un archivo javascript que se ejecuta en el lado del servidor y se encarga del control del flujo. Los resultados se presentan a través de una plantilla *JX Template*. Este esquema se puede ver en la Figura 26.

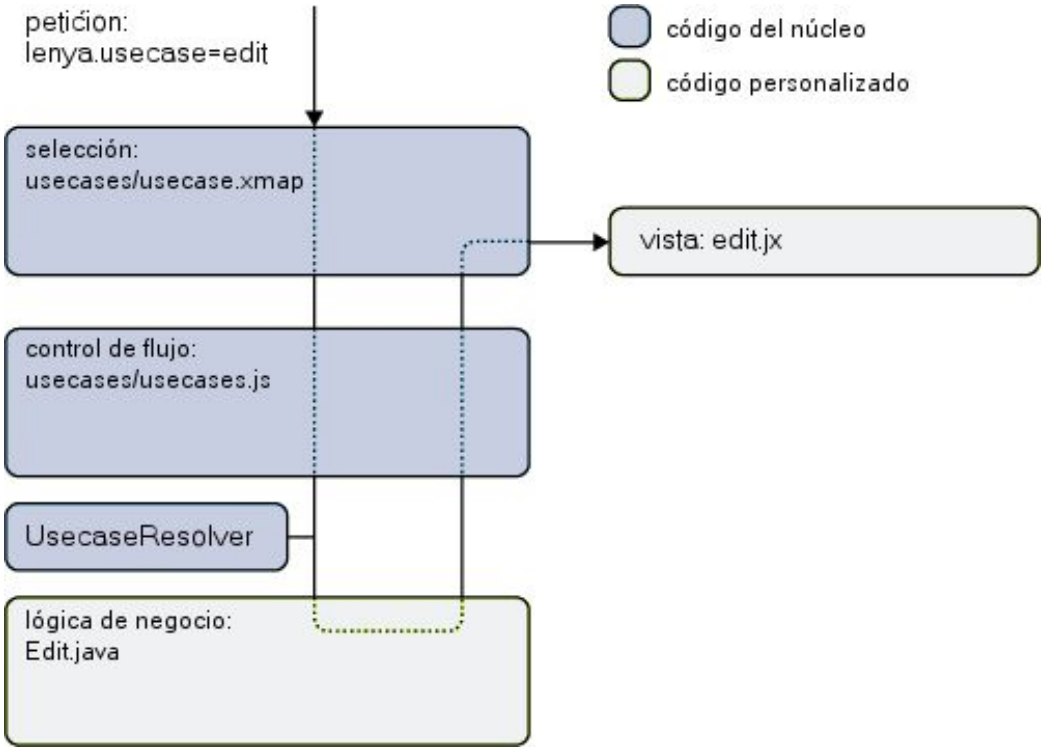

Figura 26: Esquema de la arquitectura del marco de los casos de uso en Lenya 1.4

El marco de los Casos de Uso en Lenya 1.4 es una solución incompleta. Aunque abarca la mayoría de los casos de uso habituales, puede suceder que nos encontremos ante uno que requiera un *flowscript* personalizado. En ese caso, no se podría utilizar este marco.

Para determinar cómo se ejecutará el caso de uso, Lenya intentará encontrar los archivos necesarios para manejarlo de acuerdo al nuevo marco. Si no encuentra estos archivos, tratará el caso de uso de la forma tradicional de Lenya 1.2. [LENYAUSEC2006]

4.7.4.WebDAV.

WebDAV está constituido por un conjunto de extensiones al protocolo HTTP que permiten a los usuarios editar y gestionar ficheros de forma colaborativa en servidores web remotos.

El módulo WebDAV actualmente está en fase de desarrollo, pero ya permite crear y editar documentos con:

- Dreamweaver
- Windows XP Web Folder
- OpenOffice
- Word 2003

aunque aún con limitaciones. [WEBDAV2006][LENYAWEBD2006][LENYAWIKWEB2006]

4.8.Roadmap.

El siguiente *roadmap* de Lenya ha sido obtenido de la página oficial del proyecto [LENYA2006].

4.8.1.Diagrama de posicionamiento.

El *roadmap* de Lenya está determinado por su diagrama de posicionamiento. Éste especifica las prioridades y la orientación del proyecto y muestra las áreas que necesitarán un desarrollo futuro. La versión 1.0 marca el punto en el que Lenya fue donado a la fundación Apache. Las siguientes versiones se han liberado bajo el amparo de la *Apache Software Foundation*. Las características marcadas en negrita indican que se han completado.

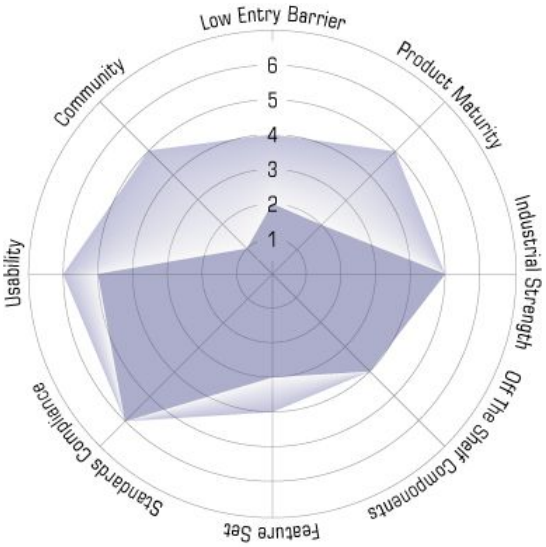

Figura 27: Diagrama de posicionamiento de Lenya

Este diagrama define 8 áreas, con un peso relativo del 1 al 7. El código base actual de Lenya se representa mediante el área de azul más intenso, mientras que el área de azul más luminoso identifica áreas en las que se trabaja o se trabajará en futuras versiones. La distintas áreas se definen como sigue:

1. **Comunidad**
   ¿Es viable que la comunidad soporte el producto? ¿Son contribuyentes independientes? ¿Son activas las listas de correo?
2. **Entrada a bajo nivel**
   ¿Es fácil comenzar a usarlo? ¿Tiene el producto un valor inmediato justo después de su descarga? ¿Es la documentación consistente? ¿Es difícil familiarizarse por si mismo con el código base del producto?
3. **Madurez del producto**
   ¿Es robusto el código? ¿Se siguen prácticas de ingeniería del software? ¿Es la arquitectura clara y consistente? ¿Se han probado los cambios para la regresión del código?
4. **Fuerza en el mercado**
   ¿Hasta qué punto es escalable el producto? ¿Hasta qué punto es flexible y estable? ¿Es seguro? ¿Tiene un buen rendimiento?
5. **Componentes estándar**
   ¿Hasta qué punto se usan componentes estándar? ¿La arquitectura es suficientemente modular como para introducir código externo?
6. **Conjunto de características**
   ¿Tiene el producto un conjunto realista de características o son exageradas?

7. **Ajuste a estándares**
   ¿Cuántos estándares usa el producto y cómo interopera con ellos?
8. **Uso**
   Es la interfaz de usuario comprensible para personal no técnico? ¿Es consistente?

4.8.2.Versiones.

Por el momento se ha planeado la liberación de tres versiones: 1.0, 1.2 y 1.4, con la posibilidad de realizar pequeñas versiones de mantenimiento entre ellas. Los números pares indican versiones estables, los impares indican versiones de desarrollo. Las sub-versiones, por ejemplo la 1.2.2., se utilizan para corrección de *bugs*.

4.8.3.Objetivos a largo plazo.

- Mejorar la comunicación y la colaboración con otros proyectos de Apache.

4.8.4.Versión 1.2

A)Comunidad

- **Añadir responsables al proyecto**
- Definir el alcance y la colaboración con el proyecto Forrest.

B)Entrada a bajo nivel

- **Mejorar y estandarizar los mecanismos de plantillas (XHTML)**
- Permitir la anotación de la documentación (como en php.net)
- **Escribir un HOWTO de una publicación**
- **Mantener un FAQ**
- **Eliminar redundancias entre las áreas de edición y de producción**
- **Escribir tutoriales sobre Lenya**

C)Madurez del producto

- **Completar la JavaDoc**
- **Unificar las URIs del CMS**
- **Estandarizar los sitemaps**
- **Realizar una planificación de actualización para los componentes dependientes (Cocoon etc)**

D)Fuerza en el mercado

- Cobertura sustancial de tests Unit
- Añadir tests web

E)Componentes estándar

- **Migrar el procesador Xinclude de Lenya a Cocoon (o viceversa)**

F)Conjunto de características

- Integrar un repositorio
- **Implementar un motor de workflow**
- **Implementar un marco de navegación de sitetree genérico**
- **Integrar un editor de Forms HTML**

- **Proveer de una GUI para la administración**

G)Ajuste a estándares

- **Proveer autenticación LDAP**

H)Uso

- Mejorar el menú de Lenya para cambiar su colocación en la página
- Mejorar las pantallas del CMS con tooltips de ayuda
- **Mejorar la integración de Bitflux/Xopus**

4.8.5.Versión 1.4

A)Comunidad

- Trabajar con OSCOM.org en estándares para los CMS
- Organizar “Lenya Sprints” (días de trabajo en grupo)
- **Impulsar otros proyectos de Apache**

B)Entrada a bajo nivel

- Mejorar la documentación

C)Madurez del producto

- **Reescribir el Framework para la creación de recursos**
- **Reescribir el Framework de publicación/replicación**
- Completar la cobertura de los tests Unit
- **Completar la JavaDoc**

D)Fuerza en el mercado

- **Mejorar los mecanismos de proxy**
- **Desarrollar estrategias avanzadas de cacheado**
- **Implementar la gestión de las transacciones**

E)Componentes estándar

- Consolidar los componentes de Autenticación y Autorización

F)Conjunto de características

- Proveer integración con OpenOffice
- **Implementar gestión de enlaces**
- **Implementar capacidades multiidioma**

G)Ajuste a estándares

- Refinar y revisar el soporte de estándares

H)Uso

- **Proveer una GUI de administración**

capítulo 5: comparativa

5.1.Gestores a Comparar.

En este capítulo realizaremos una comparación entre los siguientes CMS OpenSource:

- Apache Lenya.
- OpenCms.
- Alfresco.

Sobre Apache Lenya ya hemos tratado a lo largo de la memoria del presente Proyecto Fin de Carrera, así que en los siguientes apartados nos centraremos en comentar algunos aspectos de OpenCms y Alfresco.

5.1.1.OpenCms.

OpenCms [OPENCMS2006] es un sistema de gestión de contenidos web profesional. Posee un editor WYSIWYG con una apariencia similar a la de algunos procesadores de texto bastante conocidos. A través de ella los usuarios pueden añadir o modificar el contenido del sitio. Posee mecanismos de plantillas para conseguir una apariencia consistente dentro del sitio web. Está basado en tecnologías Java y XML. Se distribuye bajo licencia LGPL. Estas son algunas de sus características:

5.1.1.1.Está basado en Java y XML

OpenCms está escrito por completo en Java y usa la tecnología estándar de servlets. Las páginas de contenido se almacenan en archivos XML y toda la lógica se maneja a través de Java o plantillas JSP.

5.1.1.2.Entorno de trabajo a través del Navegador web.

El entorno de trabajo de OpenCms (llamado *Workplace*) está basado en el navegador. El software de OpenCms se instala en un servidor web y el usuario puede acceder a él desde cualquier lugar a través de un navegador web. Se pueden realizar restricciones de acceso para bloquear a usuarios de ciertas redes.

5.1.1.3.Gestión de Activos.

Las imágenes y otros archivos binarios pueden almacenarse en galerías y gestionarse a través de ellas para evitar redundancia de datos y facilitar su manejo. Se pueden gestionar varias galerías y configurar distintos niveles de acceso para cada una de ellas.

5.1.1.4.Gestión de usuarios y sistema de permisos integrado.

El acceso a todos los contenidos controlados por OpenCms se puede restringir mediante el sistema de gestión de permisos. Así se garantiza que el acceso a áreas protegidas está controlado y se faciilta la creación de áreas reservadas. Se puede definir una lista de control de acceso para cada recurso de OpenCms, permitiendo mayor granularidad en el control de acceso a dicho recurso. Las listas de control de acceso de una carpeta son automáticamente heredadas por todos los recursos de esa carpeta.

5.1.1.5.Publicación basada en proyectos.

El mecanismo de gestión del proyecto integrado con OpenCms provee un entorno de trabajo típico en el que el servidor de producción y el servidor de edición se hayan en la misma

máquina. Todo el contenido se organiza en proyectos y el número de estos es ilimitado. Los cambios en los contenidos pueden ser revisados antes de que el contenido sea publicado. También es posible deshacer todos los cambios no publicados y definir reglas para el acceso y la publicación dentro de un proyecto.

5.1.1.6.Workflow y gestión de tareas.

OpenCms provee mecanismos de gestión de workflow para la creación y edición del contenido. Además, se pueden crear tareas en la vista de Workflow dentro del entorno de OpenCms para distintos grupos, niveles de prioridad, usuarios, etc. con diferentes opciones de notificación. Existe un histórico de las acciones del workflow.

5.1.1.7.Editores del contenido.

OpenCms integra tanto un editor WYSIWYG para el contenido sin estructura como un editor basado en formularios para el contenido estructurado (como podría ser el de noticias, eventos, etc.).

5.1.1.8.Control de versiones.

Todos los contenidos de OpenCm están versionados. El histórico del entorno de trabajo nos permite saber qué cambios se hicieron y por quién. Todas las versiones previas se archivan y se pueden recuperar. Posee además una herramienta similar al *diff* que permite comparar los contenidos de dos versiones de una página.

5.1.1.9.Publicación de contenido estática y dinámica.

OpenCms está completamente basado en una base de datos. Por lo general, el sitio gestionado será generado de forma dinámica desde la misma. Para que el rendimiento no se vea afectado, se han introducido mecanismos de caché. También es posible realizar una publicación estática de ciertos contenidos para que sean proporcionados directamente por el servidor web.

5.1.1.10.Mecanismo modular de extensiones.

Se integra un mecanismo modular para la incorporación de extensiones. Para ello se provee la API de OpenCms para poder extender las funcionalidades del CMS sin tener que tocar el núcleo de la distribución.

5.1.1.11.Planificador.

OpenCms integra un sistema de planificación. Con él es posible invocar una acción específica en un momento determinado. Estas acciones pueden incluir publicación automática de páginas, desactivación de contenido obsoleto, etc. La gestión de las acciones planificadas se realiza a través de una herramienta de funcionamiento similar al *cron* de los sistemas UNIX.

5.1.1.12.Importación y exportación de contenido.

Se pueden realizar exportaciones de partes del contenido o de éste en su totalidad. Los recursos exportados se escriben en una base de datos XML y se almacenan en un archivo ZIP. También es posible la exportación de meta información adicional como propiedades o permisos de acceso.

5.1.2.Alfresco.

Alfresco [ALFRESCO2006] nació en Junio de 2005 (tiene sólo un año). Parte de la experiencia de los desarrolladores de Documentum, un gestor de espacios de trabajo colaborativos. En el año de andadura de Alfresco ha sufrido avances espectaculares como CMS debido a un soporte muy activo de la comunidad.

Alfresco ofrece soporte y tiene una versión comercial de pago y otra OpenSource con algunas funcionalidades menos. A medida que la versión comercial evoluciona, se van liberando componentes de ésta a la de código abierto. Por ello, al adoptar Alfresco no es necesario hacer un gran esfuerzo de creación de componentes. Además Alfresco soporta la creación de servicios web. Esto es importante dado que los servicios web parecen ocupar un papel relevante en el futuro de Internet y facilitan la integración con la web semántica.

5.1.2.1.Funciones principales

Ofrece servicios de búsqueda de documentos o páginas (a través de Lucene y multiclasificasificación de los mismos. Ofrece un interfaz a servicios web, permitiendo la integración con cualquier servicio que ofrezca la misma interfaz sin importar el lenguaje de desarrollo. Proporciona una gran integración con Windows y facilidades para el manejo de archivos (viene de un gestor documental). Además el interfaz de usuario es realmente atractivo. Las capacidades que destacan su página web son:

- Orientación al aspecto (incluye la facilidad para añadir nuevos aspectos)
- Estructura de carpetas jerárquica
- Tipos de documentos con comportamientos estándar para un documento.
- Metadatos con tipos extensibles y complejos
- Clasificaciones (navegación y búsqueda basadas en definiciones globales.
- Procesamiento de contenido basado en reglas para la adición o modificación de los datos.
- Procesamiento automatizado del contenido, incluyendo auto-clasificación, auto-indexado y manejo del workflow.
- Diccionario Global basado en espacios de nombres para permitir jerarquías y estandarización de las definiciones.
- Autenticación basada en estándares como LDAP o Microsoft Active Directory™.
- Indexado y rescate de datos completo, usando Lucene 1.4.
- Espacios de colaboración de equipos.
- Control de Versión.
- Bloqueos
- Control de Configuración (un punto central en el que se pueden configurar todas las características de Alfresco).

Los servicios que se pueden programar se realizan a través de estándares Java usando Faces 1.0. Algunos de los servicios o componentes que incluye el producto son:

- Navegación por las jerarquías del contenido con múltiples vistas configurables.
- Asistentes para la creación de espacios de contenido y para la administración.
- Asistentes para el manejo del contenido y la subida de archivos.
- Interfaces para el control de versión y para gestionar las propiedades del contenido.

- Interfaces para los espacios de colaboración.
- Edición en línea del contenido web.
- Integración con el escritorio para el almacenamiento del contenido.
- Soporte de Breadcrumb para la navegación.
- Un portapapeles para guardar el contenido y los componentes.
- Interfaz de búsqueda para permitir a los usuarios buscar y navegar por el contenido.

5.1.2.2.Características

A)Base de Datos

Cualquier base de datos soportada por *Hibernate*, incluyendo MySQL, Oracle, Microsoft SQL Server.

B)Servidor de aplicaciones

JBoss Application Server, Apache Tomcat, J2SE 5.0 (JRE 5.0)

C)Navegadores

Mozilla Firefox, Microsoft Internet Explorer

D)Portal

JBoss Portal, JSR-168

E)Tecnologías usadas

Java, Spring *Aspect-Oriented Framework*, ACEGI – Marco de seguridad *Aspect-Oriented*, MyFaces (implmentación de JSF), Hibernate ORM Persistence, Motor de búsqueda de texto Lucene, JLAN, JBoss Application Server, JBoss Portal, conversión de formatos de archivos POI, PDFBox – PDF Conversion, OpenOffice, MySQL.

F)Interfaces soportadas

- CIFS/SMB – Protocolo para la compartición de archivos.
- JSR-168 – Especificación de Portlets
- JSR-127 Java Server Faces
- FTP
- WebDAV
- Web Services

5.2.Método de comparación.

El método de comparación en el que nos basaremos será el mismo empleado para evaluar los objetivos del proyecto de Lenya para la confección de su *Roadmap.* Recordemos esos puntos:

- **Comunidad**
  ¿Es viable que la comunidad soporte el producto? ¿Son contribuyentes independientes? ¿Son activas las listas de correo?
- **Entrada a bajo nivel**
  ¿Es fácil comenzar a usarlo? ¿Tiene el producto un valor inmediato justo después de su descarga? ¿Es la documentación consistente? ¿Es difícil familiarizarse por si mismo con el código base del producto?
- **Madurez del producto**
  ¿Es robusto el código? ¿Se siguen prácticas de ingeniería del software? ¿Es la arquitectura clara y consistente? ¿Se han probado los cambios para la regresión del código?
- **Fuerza en el mercado**
  ¿Hasta qué punto es escalable el producto? ¿Hasta qué punto es flexible y estable? ¿Es seguro? ¿Tiene un buen rendimiento?
- **Componentes estándar**
  ¿Hasta qué punto se usan componentes estándar? ¿La arquitectura es suficientemente modular como para introducir código externo?
- **Conjunto de características**
  ¿Tiene el producto un conjunto realista de características o son exageradas?
- **Ajuste a estándares**
  ¿Cuántos estándares usa el producto y cómo interopera con ellos?
- **Uso**
  ¿Es la interfaz de usuario comprensible para personal no técnico? ¿Es consistente?

A través de la respuesta a cada una de las preguntas anteriores, podremos tener una base sólida para la evaluación de las herramientas.

5.3.Evaluación.

En este apartado procederemos a la evaluación de las tres herramientas. Se hará una descripción en cada apartado que nos llevará a otorgarle una puntuación en el mismo comprendida entre 1 y 7. Esta puntuación nos ayudará posteriormente a realizar un gráfico que mostrará de forma simple cuáles son los aspectos más positivos y más negativos de cada una de ellas.

5.3.1.Apache Lenya.

5.3.1.1.Comunidad

La comunidad de usuarios de Apache Lenya es moderadamente numerosa. Los principales integrantes del proyecto se dedican de forma exclusiva al desarrollo de la herramienta y a las mejoras que forman parte de los objetivos del mismo. Las implementaciones

particulares y la incorporación de funcionalidades específicas de las publicaciones (por ejemplo, un calendario de eventos) se dejan a los usuarios.

La lista de correo es bastante activa, aunque algunas preguntas realizadas por los usuarios son bastante simples. Bien es cierto que algunas de las preguntas realizadas y contestadas en la misma ayudan a una mejor comprensión de la herramienta. Si las preguntas planteadas en la lista son genéricas, normalmente reciben respuestas concisas y poco concretas. Si, por el contrario, se refieren a algún problema muy puntual y se incluye el código que no funciona es posible, aunque no siempre, que la cuestión sea resuelta de forma completa.

Sí es cierto que aunque no se resuelva completamente la pregunta planteada, las respuestas suelen orientar hacia la solución correcta. También dentro de esta lista de correo hay algunos usuarios de mucho nivel que no sólo contestan a la cuestión sino que además la analizan y proponen alternativas de implementación.

El tiempo de resolución de una cuestión realizada a la lista de correo no suele ser demasiado elevado, estando la primera respuesta disponible casi siempre en una media de 4 o 5 horas.

Los contribuyentes y consultantes de la lista de correo suelen ser programadores de aplicaciones web que están desarrollando algún proyecto basado en Apache Lenya y necesitan ayuda para resolver alguna peculiaridad del mismo. Los desarrolladores de Lenya son *Commiters* de la Fundación Apache y trabajan en empresas independientes. No existe ninguna gran empresa que respalde Lenya, como puede suceder con OpenCms, por lo que encontrar especialistas desarrolladores o soporte puede resultar algo complicado. Wyona, la empresa de desarrollo web que inicialmente ideó Lenya, se dedica en la actualidad a la creación de portales web basados en esta herramienta, aunque algunos de sus trabajadores siguen desarrollando el proyecto.

La Universidad de Zurich ha creado su propio gestor de contenidos basado en Apache Lenya y lo utiliza para el mantenimiento de su sitio web. El código fuente de este proyecto está accesible y se distribuye bajo la misma licencia que el código original de Apache Lenya.

5.3.1.2.Entrada a bajo nivel

Lenya se distribuye en dos versiones, una compilada y otra con el código fuente disponible para la compilación por parte del usuario. Estas versiones están disponibles tanto para sistemas basados en Windows como para sistemas basados en Linux.

La versión compilada está lista para su despliegue en el contenedor del servlet y su puesta en marcha inmediata.

El paquete del código fuente, por el contrario, necesita además la descarga del paquete de los fuentes de Cocoon para su compilación. Dependiendo de la versión de Lenya a instalar será necesaria la descarga de una versión distinta de Cocoon. Por ejemplo, si queremos instalar la versión 1.2.4 necesitaremos Cocoon 2.1.7. Pero si queremos instalar la versión 1.4 (actualmente en desarrollo y no indicada aún para producción) necesitaremos también la última versión de desarrollo de Cocoon.

El proceso de instalación en el caso de la distribución binaria o el de compilación e instalación en el de la versión que viene con el código fuente no es complicado. Para la versión binaria, simplemente se requiere un contenedor de servlets donde se pueda desplegar la aplicación web. Para la instalación mediante la compilación y el despliegue del paquete con el código fuente, es necesario seguir los pasos dados en el apartado *Installing Instructions* de la web de Lenya. Si se siguen punto a punto las indicaciones realizadas en él, no hay muchos problemas para conseguir hacer funcionar a Lenya.

Una vez instalado, Lenya viene con dos publicaciones: la publicación por defecto y un blog. El blog es muy básico, necesita bastantes mejoras para asimilarse a otros blogs que vienen empaquetados por defecto con otros gestores de contenidos. Se intenta que sea un esquema sobre el que poder trabajar para crear un blog más complejo, un portal de noticias o una variación sobre estos elementos.

La publicación por defecto es bastante más compleja que el blog, pero aún así no tiene muchas de las funcionalidades que otros Sistemas de Gestión de Contenidos poseen ya cuando se han descargado. Esta publicación funciona muy bien en un portal pequeño-mediano en el que las secciones y la navegación estén claramente definidas y sólo se necesite añadir nuevas páginas dentro de estas secciones o actualizar las páginas existentes. La puesta en marcha en este caso implicaría sólo cambios en el estilo y la presentación, sin modificación de las funcionalidades de la publicación. Si el portal necesita definir distintas vistas sobre el contenido o multicategorización, esta publicación no podría cumplir los objetivos.

Si lo que se desea es añadir nuevas funcionalidades como un calendario de eventos, un foro o una zona común de descarga, la conexión con una base de datos, etc. es necesario implementarlas ya que no se pueden encontrar en la distribución ni como módulos que se puedan añadir a la misma. Este es uno de los mayores problemas que se pueden encontrar en la implantación de Lenya como CMS, la necesidad de implementar desde cero funcionalidades que son comunes o habituales dentro de otros Sistemas de Gestión de Contenidos.

La documentación del proyecto, aunque comienza a tener cierto volumen y valor, sigue siendo uno de los puntos débiles de este proyecto. Hay muchos elementos sobre los que es realmente complicado encontrar algo de documentación explicativa y consistente. La documentación oficial que se puede encontrar en la página de Lenya cubre muchos aspectos sobre la arquitectura y los componentes del mismo. De cualquier forma, no se llega a entrar en la profundidad necesaria en los mismos para una búsqueda rápida de información, haciéndose necesario consultar a la lista de correo. En la wiki del proyecto también puede encontrarse información útil, sobre todo, acerca de tareas relacionadas con la implementación. La wiki no está muy bien organizada y puede resultar confusa en un primer momento.

El código fuente de Lenya suele estar bien comentado y sigue las reglas de estilo dadas por la Fundación Apache. En los *sitemaps* y los archivos xml y xsl también se suelen encontrar algunos comentarios, aunque más escasos. De cualquier modo, siempre es engorroso enfrentarse al código fuente de un programa. Existe javadoc del proyecto tanto para las versiones de la rama 1.2.x como para la versión de desarrollo 1.4.

5.3.1.3.Madurez del producto

Apache Lenya es un proyecto basado en otros muchos de la Fundación Apache, como son Cocoon, Forrest, Jackrabbit, etc. La robustez de Cocoon está sobradamente probada, ya que es un producto que lleva mucho tiempo en el mercado y que no para de revisarse y evolucionar. Tiene una comunidad muy grande a sus espaldas que cuida y marca sus líneas de desarrollo. De la misma forma, los demás proyectos relacionados con Lenya pertenecientes a la Fundación Apache son proyectos de primer nivel y, por tanto, tienen el mismo respaldo que Lenya y Cocoon.

La arquitectura de Lenya proporciona facilidades para que el cambio entre versiones no sea traumático. La separación entre el núcleo de Lenya y las publicaciones es un punto a su favor, ya que permite personalizar aspectos puntuales de una publicación sin modificar el motor del CMS. También es cierto que con la llegada de la versión 1.4 y la inclusión en el proyecto del repositorio de Jackrabbit (una implementación del *Content Repository for Java Technology API* -JCR-), se hará necesaria una migración de contenidos. Sin ésta, no se aprovecharían al máximo las nuevas posibilidades de la versión 1.4. Será necesario también, por tanto, proveer mecanismos que permitan la migración automatizada de los datos al nuevo repositorio. Este objetivo, si bien está fijado, aún no ha sido claramente definido ni ha tenido una asignación

temporal clara, aunque hay que tener en cuenta que hasta que no se haya incluido de forma satisfactoria el repositorio no hay necesidad de facilitar la migración.

Por otro lado, se está haciendo esperar la liberación de la versión 1.4, planeada para hace unos meses, y la versión 1.2 se ha quedado estancada en la versión 1.2.5. Todos los esfuerzos de los desarrolladores están centrados por el momento en la inclusión del citado repositorio y la liberación de la versión 1.4, aunque a cierre de este Proyecto Fin de Carrera, no hay aún una fecha definitiva y fija para la misma.

5.3.1.4.Fuerza en el mercado

Lenya es una herramienta flexible y escalable. Al estar basado en estándares tiene la ventaja de que sus fundamentos son definidos por entidades encargadas de la estandarización de los mismos y no por una empresa que puede cambiarlos según sus necesidades. Además, la adecuación a estándares siempre es atractiva para cualquier cliente o empresa que quiere decidirse por un producto determinado.

Si observamos a Lenya desde el punto de vista de la flexibilidad, podemos afirmar que es una herramienta tremendamente flexible, que permite realizar prácticamente cualquier tarea por sí misma o basándose en Cocoon. Esta flexibilidad lleva como contraposición una gran complejidad. La posibilidad de, por ejemplo, integrar un editor web externo dentro de la arquitectura de Lenya no es una tarea simple. Tampoco lo es la creación de nuevos tipos de documentos, o el cambio de las vistas de la web dependiendo del contexto. Son tareas realizables aunque complejas, si bien es cierto que estas tareas resultan menos complejas si se parte con ciertos conocimientos sobre Cocoon y programación en Java.

Lenya puede funcionar bajo el protocolo HTTPS, por lo que es posible cifrar el intercambio de datos.

El rendimiento de Lenya mejora conforme se van cacheando las peticiones de páginas por parte de los usuarios. El rendimiento depende fuertemente de la cantidad de contenido dinámico que haya en las páginas, ya que éste parámetro influye en las capacidades de cacheo tanto del servidor como del navegador. Los mecanismos de cacheo de Lenya están basados en los de Cocoon y son bastante avanzados. Una vez que se solicita un *sitemap*, una hoja xslt o un archivo xml, se provoca un cacheo del mismo en la memoria del servidor de forma que se mantiene mientras no sea modificado, mejorando así el tiempo de servicio. También es posible la exportación estática del contenido del portal, pero estos mecanismos son bastante básicos y no aportan ninguna ventaja en una publicación que posea mucho contenido dinámico, ya que haría imposible servir las páginas bajo demanda.

5.3.1.5.Componentes estándar

Sigue estando en mente en el proyecto adecuarse a estándares en futuros desarrollos dentro del mismo, como puede ser la implementaciones del JCR en la versión 1.4.

La arquitectura de Lenya permite el uso de código y componentes externos, aunque no es trivial su inclusión. Es una tarea ardua y difícil de realizar a no ser que se tengan conocimientos amplios y profundos sobre las arquitecturas de Lenya (y Cocoon) y el componente cuyo código se quiere introducir. Si en lugar de hablar de componentes completos hablamos de funcionalidades escritas en otros frameworks, la tarea puede resultar menos compleja, pero aún así será necesario tener un buen conocimiento sobre los fundamentos de funcionamiento de Lenya.

5.3.1.6.Conjunto de características

El conjunto de características de Lenya es ambicioso, pero no refleja completamente la realidad del producto. Hay algunas características que no están implementadas completamente o que no se han implementado tal y como se indica que se ha hecho.

5.3.1.7.Ajuste a estándares

Uno de los puntos fuertes de Lenya es que se basa en componentes estándar. Se acoge a estándares como XML, XSLT, CSS y XHTML, con todas las ventajas que siempre acarrea esta opción. Para el control de acceso implementa también gestión LDAP.

En estos momentos, Lenya no aporta a los estándares más que su uso y publicitación.

5.3.1.8.Uso

La interfaz de usuario de Lenya es bastante distinta a las de otros gestores de contenidos. La separación conceptual entre área de edición, de administración, de administración del sitio y de producción puede resultar algo confusa en un primer momento.

Cada una de estas áreas tiene sus propias pantallas de configuración y edición y aunque mantienen el mismo esquema en los menús y la misma disposición de los elementos es necesario ser consciente en cada momento de dónde se está y qué se quiere hacer.

Tal vez una distribución distinta de los menús o una mayor diferenciación entre las áreas de gestión haría más simple la comprensión de la interfaz.

5.3.2.OpenCms

5.3.2.1.Comunidad

OpenCMS es un producto de la empresa Alkacon, que es la encargada del desarrollo de sus nuevas versiones. No se conoce en qué grado se introducen las contribuciones que la comunidad de usuarios y desarrolladores que lo utilizan puedan hacer ni tampoco existen proyectos paralelos que tengan relación o estén basados en el mismo. También ofrecen soporte técnico, cursos de formación sobre la plataforma y desarrollo de proyectos a la medida. Existe una lista de correo bastante activa en la página oficial del producto.

5.3.2.2.Entrada a bajo nivel

El producto es relativamente fácil de usar, dependiendo la facilidad para introducir nuevos datos de un tipo concreto en la sencillez del formulario existente para el mismo (deben desarrollarse a parte nuevos formularios para tipos de contenidos nuevos no soportados en origen por la plataforma). Hace uso masivo de Javascript y no cumple normas de accesibilidad. Existe documentación de uso del producto pero la documentación del código fuente de la plataforma es pobre. No existen documentos sobre la arquitectura ni el diseño de la misma. La calidad del código fuente es pobre y, generalmente, no cumple los principios de buenas prácticas para la ingeniería del software y el diseño es bastante deficiente.

5.3.2.3.Madurez del producto

El código es enrevesado y cualquier estudio de métricas sobre él pone de relieve la dificultad que hay para mantenerlo o ampliarlo. Los errores son bastante difíciles de depurar. No está clara la arquitectura seguida y ésta no está particularmente diseñada para su extensión en el futuro (de hecho, ha tenido que ser rehecha en la versión 6.0).

5.3.2.4.Fuerza en el mercado

Se permite la creación de módulos con funcionalidades no proporcionadas por la plataforma. Durante el uso normal no se producen fallos en la plataforma, aunque las nuevas funcionalidades pueden provocar defectos. La aplicación es es lenta y sobrecarga bastante el servidor donde se ejecuta.

5.3.2.5.Componentes estándar

No se han utilizado componentes estándares para el desarrollo de la plataforma. Asimismo, ésta no implementa ningún estándar de contenidos ni presentación (por ejemplo, especificación de portlets o Java Content Repository, JSR 168 y JSR 170, respectivamente). Se facilita la implementación de nuevas funcionalidades mediante la creación de módulos.

5.3.2.6.Conjunto de características

Las características ofrecidas por el producto son bastante básicas (sistema de permisos tipo Unix, gestión de grupos de usuarios con poca granularidad, publicación en distintos proyectos, etc.) y funcionan correctamente, aunque se echan en falta otras más avanzadas que están siendo implementadas en productos más nuevos: gestión avanzada de grupos de usuarios, integración con diferentes fuentes de datos como LDAP, cumplimento de estándares para su integración con servicios de autenticación globales (como *Single Sign On*), etc.

5.3.2.7.Ajuste a estándares

No se cumplen los estándares de la industria para entornos de gestión de contenidos. La formalización para los datos almacenados se basa en descripciones en XML.

5.3.2.8.Uso

La interfaz de usuario es fácilmente utilizable por personal no técnico (ha sido utilizada con éxito por documentalistas u otro personal sin formación técnica para la creación de contenidos de sitios web corporativos). La consistencia de la interfaz viene determinada en gran parte por la calidad de los formularios de datos (requieren de un desarrollo propio).

5.3.3.Alfresco.

5.3.3.1.Comunidad

Alfresco es un CMS creado por uno de los primeros desarrolladores del Gestor Documental *Documentum*, un Sistema de Gestión Documental muy famoso, de amplio uso y distribuido bajo licencia comercial.

Alfresco ha nacido como un proyecto *OpenSource*, aunque realmente hay dos versiones. Una de ellas tiene un conjunto de características reducido y es gratuita, la otra tiene un conjunto de características más amplio y es de pago.

Los contribuyentes de Alfresco participan a través de medios comunes, foros, listas de correo, etc. Pero debido al poco tiempo que lleva el producto en el mercado todavía no posee una comunidad muy amplia. Aún así, ha sufrido un crecimiento espectacular en su andadura, lo que hace prever que en un tiempo relativamente corto su comunidad también crezca.

5.3.3.2.Entrada a bajo nivel

Alfresco es un producto con muchas funcionalidades en su distribución básica, por lo que una vez descargado e instalado puede comenzar a usarse. La interfaz de usuario está muy conseguida y ofrece un método simple y rápido de interacción.

La documentación es aún escasa respecto a otros CMS del mercado, pero esto es normal si se tiene en cuenta el tiempo que hace que se liberó la primera versión del mismo.

El código está bien estructurado y comentado, por lo que una familiarización con el mismo es menos ardua que con otros CMS.

5.3.3.3.Madurez del producto

Alfresco ha nacido con una arquitectura robusta, extensible y consistente. Está desarrollado por personas que han trabajado en la creación de *Documentum*, por lo que se tenía más constancia de los puntos claves a implementar y los puntos a desechar. Se ha puesto mucho interés en la integración simple de *webservices* y funcionalidades escritas en varios lenguajes de programación distintos.

Aunque lleva poco tiempo en el mercado, es un producto robusto y funcional y se espera que siga mejorando rápidamente en los próximos meses.

5.3.3.4.Fuerza en el mercado

Alfresco posee un buen rendimiento en carga. Se ha diseñado para ser fácilmente escalable. Para ello se ha ofrecido la posibilidad de desarrollar *webservices* para la inclusión de nuevas funcionalidades y la creación de Portlets, que permitirán incluir en una página diferentes piezas de contenido de fuentes diversas.

5.3.3.5.Componentes estándar

Alfresco utiliza componentes estándar de forma extensiva. Es una de las bases de su desarrollo. Aparte del JSR 168 y el JSR 127, Alfresco soporta WebDAV y está planeado implementar JSR 170 si la comunidad lo requiere.

Alfresco es modular hasta el punto de soportar varios lenguajes de programación para el desarrollo de sus componentes o sus *webservices*, por lo que la integración con herramientas ya desarrolladas es algo más simple que en otros gestores y no requiere de tanta especialización en un lenguaje de programación particular.

5.3.3.6.Conjunto de características

Alfresco tiene un conjunto de características muy realista. Todas las listadas en la página web del proyecto se encuentran ya implementadas y disponibles para la comunidad, ya sea en la versión de pago o en la versión gratuita.

Además, tiene un conjunto de características a desarrollar muy ambicioso y atractivo. Un punto a favor es que las características creadas exclusivamente para la versión de pago son liberadas a la versión gratuita en un espacio de tiempo relativamente corto.

5.3.3.7.Ajuste a estándares

Entre los objetivos de Alfresco se encuentra el de ajustarse al máximo a los estándares de la industria. Utiliza XML para la creación y el mantenimiento de los metadatos, por ejemplo. Además, este ajuste se puede ver en la intención de desarrollar todas las funcionalidades

nuevas de acuerdo a los estándares creados por los diferentes organismos para ellas. Como ejemplo, en la integración del repositorio, se piensa implementar el JSR 170.

Alfresco acepta y utiliza los estándares pero no propone ninguno nuevo por el momento.

5.3.3.8.Uso

La interfaz de usuario de Alfresco es muy intuitiva y agradable en lo visual. Ha sido utilizada por personal no técnico sin problemas. La incorporación del protocolo de comunicación de archivos de Windows hace que el tratamiento de las carpetas sea muy similar al seguido en este sistema operativo.

Además, esta interfaz está muy orientada desde un primer momento al trabajo colaborativo. Proporciona muchas opciones de configuración, lo que permite, por ejemplo, no tener que entrar a bajo nivel en la aplicación para la configuración del esquema de Workflow. Prácticamente todo es configurable o modificable desde la interfaz de usuario de Alfresco.

5.4.Conclusiones

5.4.1.Comparativa funcional

En el se han resumido los diferentes apartados, puntuando entre 1(muy mal) y 6 (excelente) cada uno de los mismos. El resumen de los resultados es:

	Alfresco	Lenya	OpenCMS
Necesidades del Sistema	4	5	3
Seguridad	4	5	4
Soporte	5	3	1
Facilidad de Uso	6	2	3
Rendimiento	6	6	3
Gestión	5	4	3
Interoperabilidad	6	4	3
Flexibilidad	6	5	5
Aplicaciones integradas	3	2	4
Total	45	34	25

En cualquier caso, los resultados deben ser tomados con mucha cautela por los siguientes motivos:

1. Son los propios desarrolladores de cada proyecto los que rellenan la lista de funcionalidades. En este sentido, Alfresco, parece tener una tendencia mas comercial.
2. Es posible la creación de nuevas funcionalidades en cualquier portal. Dependiendo de la función será mas o menos costoso. Por ejemplo: se podría hacer un servicio de blog en Lenya con poco esfuerzo, sin embargo sería complicado introducir cambios estructurales como el soporte de bases de datos que no estén en Hibernate para Alfresco.

5.4.2.Resultados gráficos y discusión.

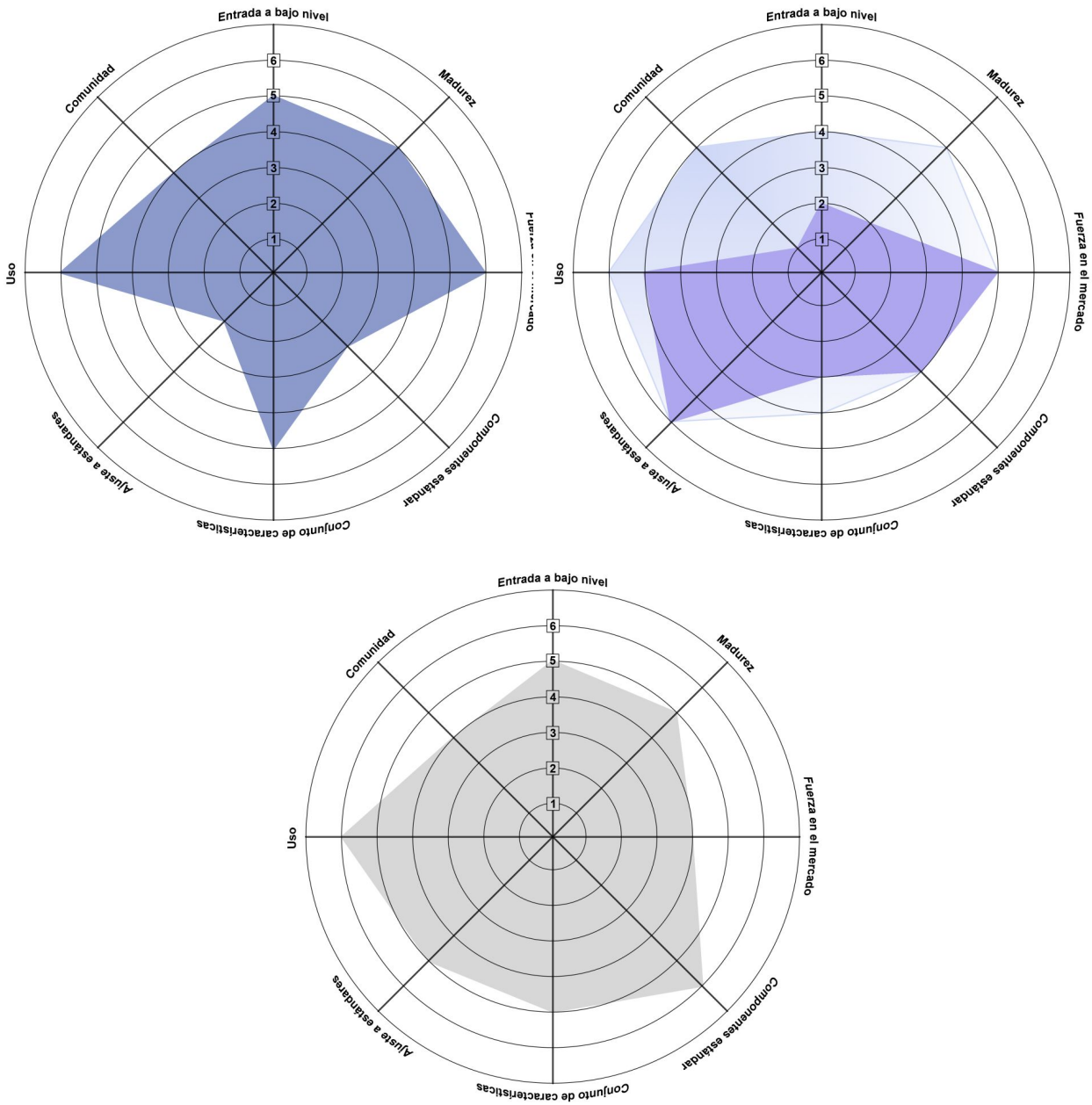

Figura 28: Comparativa entre OpenCms (izquierda), Lenya 1.2 y 1.4 (derecha) y Alfresco (abajo)

En base a las figuras anteriores y las características expuestas en los apartados 5.3.1, 5.3.2 y 5.3.3 vamos a entrar a valorar los puntos fuertes y débiles de cada una de estas herramientas.

5.4.2.1.Entrada a bajo nivel.

Una nota baja implica grandes inversiones iniciales en formación del equipo de trabajo y adaptación del código fuente del proyecto. Por el contrario, también indica la necesidad de una alta cualificación técnica y la diferenciación de la competencia frente a los clientes.

En este apartado claramente, Lenya obtiene una baja nota en cualquiera de sus versiones. Aunque el código es claro y esta bien documentado, la escasa documentación acerca del proyecto así como la falta de soporte comercial hacen todavía mas difícil la tarea de desarrollar nuevos módulos para Lenya. Tarea que es necesaria debido a la falta de módulos en el proyecto. El proyecto de Alfresco, por el contrario, ofrece soporte comercial y libera módulos nuevos con relativa frecuencia.

Podríamos resumir diciendo que es el **Principal Punto débil de Lenya**.

5.4.2.2.Madurez del Producto.

La mayoría de las preguntas se refieren al código. A su robustez y facilidad para mantenerlo o hacer cambios.

En la actualidad podríamos decir que los tres sacan igual puntuación, aunque hasta la versión 6 de OpenCMS este iba por detrás. Lenya tiene un código muy bien estructurado y comentado. Lenya se basa en cocoon con lo cual la solidez y madurez del producto están fuera de toda duda. OpenCMS 5 tiene un código peor organizado y comentado. En la versión 6 se ha realizado una reestructuración completa de la arquitectura de OpenCMS y se ha mejorado mucho este aspecto.

Alfresco nació de una base solida (el famoso gestor documental *Documentum*), por lo que desde un principio ha intentado consolidar un código robusto y mantenible.

5.4.2.3.Fuerza en el mercado

Lenya tiene buenas notas en cuanto a disponibilidad, estabilidad, rendimiento, redundancia, escalabilidad independencia de plataforma. La re-utilización de Lenya es muy alta y su sistema de plantilla de publicaciones hace que reproducir una publicación existente (versión 1.4), realizar cambios menores y cambiar el aspecto sea sólo cuestión de días. También sería relativamente sencillo utilizar código de aplicaciones Cocoon existentes.

OpenCMS 5 presenta algunos problemas de escalabilidad y rendimiento (sobre todo en la parte de administración) que se han intentado solucionar en la versión 6. La falta de estándares y la dependencia con la base de datos, hace que sea muy difícil aprovechar código de otros proyectos. Ofrece muchas funcionalidades en forma de módulos.

Alfresco tiene más aspecto de producto reutilizable y escalable. Su arquitectura y la implementación del estándar Java Portlets hace aún más incapié en este aspecto. Además, está muy orientado a servicios web. Además, ofrece más servicios e implementaciones de funcionalidades necesarias en un Sistema de Gestión de Contenidos que Lenya.

Según todo lo dicho hasta el momento, los tres CMS tienen la misma puntuación en cuanto a la seguridad y el rendimiento. Sin embargo, la disponibilidad de funcionalidades necesarias para la creación de portales es mucho mayor en OpenCMS y Alfresco que en Lenya. Aunque las figuras indiquen lo contrario, es opinión del autor que **Lenya y Alfresco superarían a OpenCMS en igualdad de uso en condiciones similares de experiencia. Por otro lado, Alfresco parece el producto que va a tener más fuerza en el mercado, aunque lleve menos tiempo en el mismo.**

5.4.2.4.Componentes Estándar

Lenya utiliza componentes estándar. De hecho es una de sus principales características y quizás su punto mas fuerte.

En OpenCMS no se han utilizado componentes estándar para el desarrollo de la plataforma. Asimismo, ésta no implementa ningún estándar de contenidos ni presentación (por ejemplo, especificación de Portlets o Java Content Repository, JSR 168 y JSR 170,

respectivamente). Se facilita la implementación de nuevas funcionalidades mediante la creación de módulos.

Alfresco por su parte también está muy centrado en la utilización de componentes estándar, implementando por ejemplo el JSR 168.

Aunque las figuras indican un empate, **Lenya y Alfresco superan claramente a OpenCMS en este apartado.**

5.4.2.5.Conjunto de Características

En Lenya se ofrece un conjunto de características realistas necesarias en los gestores de contenidos. Se echa de menos una mejor integración con windows o la multicategorización. Aún así, el conjunto de características ofrecidas por Lenya puede resultar más pobre que en otros CMS.

En OpenCMS, las características ofrecidas por el producto son bastante básicas (sistema de permisos tipo Unix, gestión de grupos de usuarios con poca granularidad, publicación en distintos proyectos, etc.) y funcionan correctamente, aunque se echan en falta otras más avanzadas que están siendo implementadas en productos más nuevos: gestión avanzada de grupos de usuarios, integración con diferentes fuentes de datos como LDAP, cumplimento de estándares para su integración con servicios de autenticación globales (como Single Sign On), etc.

En Alfresco se están implementando un conjunto de características muy atractivo dentro de los CMS. Aparte de la multicategorización, la adición de Portlets, los *webservices* y la integración con Windows, se dispone de herramientas para el manejo y la personalización de los metadatos y para la gestión del Workflow, dotándolo de una granularidad muy grande.

En este apartado **Alfresco saca la mejor puntuación y Lenya está algo por debajo de OpenCMS**.

5.4.2.6.Ajuste a Estándares

Este apartado apenas merece discusión. Lenya y Alfresco cumplen todos los estándares del W3C para la gestión de contenidos. OpenCMS se basa en una solución que no cumple estándares. El cumplimiento de estándares es importante de cara al propio marketing del producto. Además si estos se cumplen siempre será mas fácil su integración e interoperabilidad con otras aplicaciones.

Lenya y Alfresco son muy superiores a OpenCMS en este apartado.

5.4.2.7.Uso

En Lenya la interfaz de usuario es relativamente sencilla de utilizar por personal no técnico aunque quizá no tanto como en otros sistemas. Necesitaría de algunos cambios si se quiere usar multicategorización y otras funcionalidades. La interfaz tampoco es especialmente atractiva desde el punto de vista del diseño.

En OpenCMS la interfaz de usuario es fácilmente utilizable por personal no técnico (ha sido utilizada con éxito por documentalistas u otro personal sin formación técnica para la creación de contenidos de sitios web corporativos). La consistencia de la interfaz viene determinada en gran parte por la calidad de los formularios de datos (requieren de un desarrollo propio).

En Alfresco, la interfaz de usuario es simple y potente además de atractiva en lo visual. Conforma uno de los puntos fuertes de este Sistema de Gestión de Contenidos.

En este apartado OpenCMS y Alfresco son superiores a Lenya.

5.4.2.8.Comunidad

Como se ha comentado con anterioridad, no se sabe hasta qué punto se toman en cuenta las opiniones de la comunidad de usuarios de OpenCMS. La empresa que lo creó, Alkacon, es la encargada de dirigir los destinos del mismo.

Lenya ha tenido hasta ahora una comunidad poco activa. Además en Lenya habría que distinguir entre la comunidad de Cocoon y la propia de Lenya (la primera es más activa que la segunda). En la actualidad se ha convertido en proyecto de primer nivel dentro de la Fundación Apache con lo que es de prever que el apoyo de la comunidad mejore. En cualquier caso, apenas existe documentación ni manuales de desarrollo.

El proyecto de Alfresco también está dirigido por una empresa, aunque su impacto en la comunidad parece mayor que el de OpenCMS. Además, el acceso a la misma está mucho más claro que en Lenya y OpenCMS, puesto que se muestra en la página principal del proyecto.

En este apartado los tres empatarían con una nota baja.

5.4.3.Conclusión final.

Una vez valorados todos los aspectos, podemos concluir que **Alfresco parece ser el CMS mejor posicionado para su uso y personalización en un tiempo relativamente bajo**. Mientras es un proyecto que ha crecido mucho en un tiempo muy limitado, Lenya parece haberse estancado en la incorporación del repositorio para su versión 1.4 (debería haber sido liberada en octubre de 2005). OpenCMS 5 fue un desastre. Muy utilizado por muchas empresas pequeñas y medianas, pronto mostró sus carencias y sus problemas de estabilidad y rendimiento en tareas relativamente simples. Debido a este motivo, en la versión 6 se ha tenido que rehacer toda su arquitectura, perdiendo así cualquier compatibilidad con proyectos y portales realizados en versiones anteriores.

Por tanto, si se desea crear un CMS altamente flexible, escalable y reutilizable además de propio, la opción más adecuada sería la elección de Apache Lenya. Si por el contrario se desea un CMS capaz de realizar tareas complejas en cuanto a la multicategorización de sus contenidos y muy granular en cuanto a los permisos para trabajo en grupo, la opción más acertada sería Alfresco. Además, es opinión del autor que este CMS es el más indicado para soportar múltiples *webservices* y el que podría ofrecer una mejor reutilización de sus componentes debido al uso del estándar Java Portlets.

Por otro lado, si lo que se desea es la creación de un portal que no tendrá muchos requerimientos en el plano de rendimiento, que esté basado en alguna plantilla predeterminada para no tener que desarrollar mucho el aspecto y que necesite una puesta en marcha rápida, tal vez la mejor opción sería OpenCMS ya que se trata de una herramienta de instalación simple y con muchas funcionalidades incluidas en la distribución básica.

capítulo 6: conclusiones y líneas de avance

6.1.Conclusiones

A lo largo del presente Proyecto Fin de Carrera hemos mostrados las líneas maestras de la arquitectura de Lenya y sus opciones de personalización. También hemos realizado una comparativa funcional entre Sistemas de Gestión de Contenidos *OpenSource* para observar sus diferencias y poseer los mecanismos necesarios para realizar una elección entre ellos.

Lenya tuvo una entrada en el mercado muy esperanzadora, con un fuerte apoyo de la Fundación Apache y un conjunto de características y filosofía de desarrollo muy adecuadas a los asuntos relacionados con la Gestión de Contenidos. Sin embargo, la flexibilidad que se le ha querido otorgar desde un primer momento lo convierten más en un framework para el desarrollo de una herramienta de gestión de contenidos que en un CMS propiamente dicho.

Su arquitectura es robusta y escalable, permitiendo el desarrollo de Sistemas de Gestión más avanzados o con mayores funcionalidades y minimizando los problemas relacionados con la incompatibilidad de futuras versiones de desarrollo.

Como hemos indicado anteriormente, Lenya ha perdido fuerza entre los CMS. Esto se ha debido a retrasos en la incorporación de la implementación del JSR-170 al núcleo, uno de los objetivos fundamentales de la versión 1.4. Sin embargo, el proyecto no se haya estancado. Se podría decir que se encuentra en un punto de desarrollo fundamental y necesario para asegurar su futuro dentro de los Gestores de Contenidos *OpenSource.*

Debido a todo lo expuesto y como conclusión del presente Proyecto Fin de Carrera, podríamos decir que Apache Lenya ha sentado una base firme para la creación se Sistemas de Gestión de Contenidos basados en tecnologías de la Fundación Apache. Sin embargo, también podemos concluir que Apache Lenya en estos momentos no es un Sistema de Gestión de Contenidos propiamente dicho, sino más bien un *framework* de desarrollo para Sistemas de Gestión de Contenidos. La flexibilidad antes indicada le otorga la posibilidad a cualquier empresa u organización de moldear Lenya y ajustarlo a sus necesidades, creando una herramienta de Gestión personalizada y funcional.

6.2.Líneas de avance

Después del estudio realizado sobre la arquitectura y las funcionalidades de Apache Lenya y del análisis realizado de su funcionamiento y características, podemos definir las siguientes líneas de avance para el presente Proyecto Fin de Carrera.

- Creación de módulos funcionales para Lenya.

  Como hemos comentado durante la presente memoria, Lenya carece de módulos funcionales de los que disponen otras herramientas para la Gestión de Contenidos. Entre ellas se echa de menos un calendario de eventos, un foro de discusión, un agregador de noticias por RSS o una zona reservada a usuarios. El desarrollo de estas herramientas podría formar parte de una línea de avance sobre este proyecto, aprovechando la información proporcionada sobre la arquitectura de Lenya y su personalización.

- Multicategorización.

  En los portales actuales, la multicategorización es un factor muy importante. Debido a la inclusión de nuevos tipos de contenidos en las páginas web y a que estos son cada vez más pesados, es muy útil asignar varias categorías a un mismo contenidos. De esta forma se puede conseguir que diferentes vistas del portal tengan acceso a un contenido particular sin necesidad de tener varias copias del mismo. Lenya no dispone de mecanismos para la implementación de estas características. Debido a

esto, otra línea de avance para el presente proyecto sería la implementación de mecanismos de gestión, asignación y presentación de multicategorías para un tipo de recurso.

- Creación de un contenedor de activos.

En la implementación actual de Apache Lenya, los documentos y las imágenes accesibles desde una página se encuentran ligados a ella, siendo imposible hacerles referencia desde otra página distinta sin duplicarlos. Esta característica es muy restrictiva y provoca la existencia de una redundancia de activos a todas luces innecesaria. Por tanto, otra posible línea de avance para este proyecto sería la implementación de un contenedor de activos accesible para todo el portal. De esta forma, la creación de una galería de imágenes o de una zona de descarga sería un trabajo poco costoso y se conseguiría reducir la redundancia de activos, disminuyendo también la carga del servidor.

bibliografía

Bibliografía

[ALFRESCO2006] Página oficial del producto. Disponible en internet en <http://www.alfresco.com>.

[ANT2006] Apache Ant. Página oficial del Proyecto. Disponible en internet en <http://ant.apache.org>.

[ASPCMS] Fraser, Stephen R. G. "Real-World ASP.NET. Building a Content Management System". Ed. Apress, 2002.

[CMSBIBLE2002] Boiko, Bob. “Content Management Bible”. Ed. Wiley Publishing, 2002.

[COCOON2006] Cocoon 2.1. Página oficial del Proyecto. Disponible en internet en <http://cocoon.apache.org>.

[COCOONSOC2006] Documento electrónico. Disponible en internet en <http://cocoon.apache.org/2.1/introduction.html>

[DUBLIN2006] Dublin Core Metadata Initiative (DCMI). Página oficial. Disponible en internet en <http://dublincore.org>.

[EXCALIB2006] Apache Excalibur. Página oficial del Proyecto. Disponible en internet en <http://excalibur.apache.org>.

[FORREST2006] Apache Forrest. Página oficial del Proyecto. Disponible en internet en <http://forrest.apache.org>.

[FORRESTSOURCE2006] Documentación Forrest. SourceTypeAction. Disponible en internet en <http://forrest.apache.org/docs_0_70/cap.html>

[ISO 8879:1986] ISO 8879:1986. Information processing -- Text and office systems -- Standard Generalized Markup Language (SGML).

[JACKR2006] Apache Jackrabbit. Página oficial del Proyecto. Disponible en internet en <http://jackrabbit.apache.org>.

[JAVAXML2001] McLaughlin, Brett. “Java y XML”. Ed. ANAYA Multimedia, 2001.

[JAVAXSLT2002] Burke, Eric M. “Java y XSLT”. Ed. ANAYA Multimedia, 2002.

[LENYA2006] Página oficial de Lenya. Disponible en interner en <http://lenya.apache.org>

[LENYANAV2006] Página oficial de Apache Lenya. Componentes de la navegación. Disponible en internet <http://lenya.apache.org/1_2_x/components/layout/navigation.html>

[LENYAPUBTEM2006] Página oficial de Apache Lenya. Publication Templating. Disponible en internet <http://lenya.apache.org/1_4/reference/publication-templating/index.html>.

[LENYAUSEC2006] Página oficial de Apache Lenya. Usecase Framework. Disponible en

internet <http://lenya.apache.org/1_4/reference/usecase-framework/index.html>.

[LENYAWEBD2006] Página oficial de Apache Lenya. Módulo WebDAV. Disponible en internet en <http://lenya.apache.org/modules/webdav/index.html>.

[LENYAWIKWEB2006] Wiki de Apache Lenya. Cómo editar con WebDAV. Disponible en internet en <http://wiki.apache.org/lenya/HowToEditWithWebDAV>.

[OPENCMS2006] Página oficial del producto. Disponible en internet en <http://www.opencms.org>.

[RECOMXML2004] W3C Recommendation 04 February 2004. Extensible Markup Language (XML) 1.0 (Third Edition). Disponible en internet: <http://www.w3.org/TR/REC-xml/>

[RECOMXPATH1999] W3C Recommendation 16 November 1999. XML Path Language (XPath) Version 1.0. Disponible en internet <http://www.w3.org/TR/xpath>.

[RECOMXSLT1999] W3C Recommendation 16 November 1999. XSL Transformations (XSLT) Version 1.0. Disponible en internet <http://www.w3.org/TR/xslt>.

[RELAXNG2001] OASIS Committee Specification, 3 December 2004. RELAX NG Tutorial. Disponible en internet <http://www.relaxng.org/tutorial-20011203.html>

[SLIDE2006] Jakarta Project. Slide. Página oficial del Proyecto. Disponible en internet en <http://jakarta.apache.org/slide>.

[SOC2006] Separation of Concerns. Disponible en internet en <http://en.wikipedia.org/wiki/Separation_of_concerns>

[WAI2006] Web Accessibility Initiative. Página oificial. Disponible en Internet en <http://www.w3.org/WAI/>.

[WEBDAV2006] Página oficial de WebDAV. Disponible en internet en <http://www.webdav.org/>.

[WEBDAV2006] WebDAV Resources. Página de la comunidad. Recursos en documentación, especificaciones y software. Disponible en internet en <http://www.webdav.org/>

[XMLEJEM2001] Gutiérrez, Abraham y Martínez, Raul. “XML a través de ejemplos”. Ed. Ra-Ma, 2001.


Archivo
Editar
Administration
Inicio
Usuarios
Grupos
Rangos de IP
Vaciar papelera
Buscar

apéndices

Apéndice A: Un ejemplo XSLT.

Para realizar transformaciones XSLT se necesitan tres componentes: una fuente de datos XML, una hoja de estilo XSLT y un procesado XSLT. La hoja de estilo XSLT es en realidad un documento XML bien formado, por lo que el procesador XSLT incluirá o utilizará, además, un analizador gramatical XML. Apache Lenya utiliza Xalan como procesador XSLT y Xerces como analizador gramatical. El siguiente ejemplo presenta un primer prototipo de una página inicial para un foro de debate. Estos son los datos XML en bruto, sin instrucciones de formato ni HTML. Como se puede observar, la página inicial lista los paneles de mensajes entre los que el usuario puede escoger.

Listado A1.1

```
<? xml version="1.0" encoding="UTF-8"?>
<discussionForumHome>

        <messageBoard id="1" name="Java Programming"/>
        <messageBoard id="2" name="XML Programming"/>
        <messageBoard id="3" name="XSLT Questions"/>

</discussionForumHome>
```

Se supone que estos datos serán generados de forma dinámica como resultado de una consulta a la base de datos, en vez de ser codificados manualmente como un archivo estático XML. Sea cual sea su origen, los datos XML no dicen nada sobre el modo en que aparecerá realmente la página Web. Para que resulte más claro, haremos que la hoja de estilo SXLT sea bastante sencilla en este aspecto. La estructura de la hoja de estilo XSLT puede ser ampliada más tarde, sin comprometer ninguna de las estructuras de datos XML soportadas. Lo que es más importante, el código Java que va a generar los datos XML no tiene que estar plagado de HTML ni de lógica de interfaz de usuario, sino que simplemente produce los datos XML básicos. Después de que se haya definido el formato de los datos, un programador Java puede comenzar a trajavar en la lógica de la base de datos y en el código de generación de XML, mientras otro miembro del equipo comienza a escribir las hojas de estilo XSLT.

El Listado A.2 muestra una hoja de estilo XSLT para producir la página inicial.

Listado A1.2 DiscussionForumHome.xslt

```
<? xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
        version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:output method="html">

        <!-- corresponde a la raíz del documento -->
        <xsl:template match="/">
          <html>
                <head>
                  <title>Discussion Forum Home Page</title>
                </head>
                <body>
                  <h1>Discussion Forum Home Page</h1>
                  <h3>Please select a message board to view: </h3>
                  <ul>
                        <xsl:apply-templates select="discussionForumHome messageBoard">
                  </ul>
                </body>
          </html>
        </xsl:template>

        <!-- corresponde a un elemento <messageBoard> -->
        <xsl:template match="messageBoard">
          <li>
                <a href="viewForum?id={@id}">
                  <xsl:value-of select="@name"/>
```

```
                    </a>
                </li>
            </xsl:template>
        </xsl:stylesheet>
```

Como podemos observar, esta hoja XSLT es también un documento XML bien formado. Al igual que en otros archivos XML, la primera línea de la hoja de estilo es una declaración XML:

```
<? xml version="1.0" encoding="UTF-8"?>
```

A menos que tratemos con temas sobre internacionalización, eso no cambiará en cada hoja de estilo que escribamos. Esta línea va seguida inmediatamente por el elemento raíz del documento, que contiene el resto de la hoja de estilo:

```
<xsl:stylesheet
        version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

El elemento `<xsl:stylesheet>` tiene dos atributos en este caso. El primero, `version="1.0"`, identifica la versión de la especificación XSLT. En el siguiente atributo declara el espacio de nombres XML, definiendo el significado del prefijo `xsl:` que podemos ver en todos los elementos XSLT. En el caso de XSLT, el prefijo del espacio de nombres no tiene que ser xsl, aunque se ha expandido su uso. Sin embargo, su valor sí debe ser http://www.s3.org/1999/XSL/Transform.

La siguiente línea en la hoja de estilo indica sencillamente que el árbol de resultados debería tratare como un documento HTML y no como un documento XML:

```
<xsl:output method="html">
```

En la versión 1.0 de XSLT no se requiere que los procesadores soporten por completo este elemento. Sin embargo, Xalan sí lo soporta. Dado que la propia hoja de estilo debe está escrita en XML bien formado, resulta difícil incluir algunos indicadores HTML. En lugar de escribir `<hr>` deberemos escribir `<hr />` en nuestra hoja de estilo. Cuando el método de salida es html, los procesadores como Xalan eliminarán el carácter barra oblicua (`/`) del árbol del resultado, lo cual produce el HTML que esperan los buscadores Web comunes.

Lo que resta de la hoja de estilo son dos plantillas. Cada una de ellas se corresponde con un modelo del documento de salida XML y es responsable de crear una salida al árbol. La primera plantilla se repite del siguiente modo:

```
<xsl:template match="/">
  <html>
    <head>
       <title>Discussion Forum Home Page</title>
    </head>
    <body>
       <h1>Discussion Forum Home Page</h1>
       <h3>Please select a message board to view: </h3>
       <ul>
         <xsl:apply-templates select="discussionForumHome messageBoard">
       </ul>
    </body>
  </html>
```

```
</xsl:template>
```

Cuando el procesador XSLT comienza el proceso de transformación, observa la hoja de estilo en busca de una plantilla que se corresponda con el modelo "/". Dicho modelo corresponde al documento fuente XML que se está transformando. Este es siempre el punto de arranque del proceso, por lo que casi todas las hojas de estilo que escribamos contendrán una plantilla similar a esta. Dado que esta es la primera plantilla de la que se crea una instancia, es también donde crearemos el esqueleto del documento HTML resultante. La segunda plantilla, que corresponde al modelo "messageBoard" se ignora en este punto. Esto se debe a que el procesador está observando la raíz del documento XML y el elemento <messageBoard> se encuentra anidado bajo el elemento <discussionForumHome>.

La mayor parte de los indicadores de esta plantilla no comienzan con <xsl:, por lo que sencillamente se copian en el árbol de resultado. De hecho, el único contenido dinámico perteneciente a esta plantilla en concreto es la línea que aparece a continuación, la cual dice al procesador que continúe el proceso de transformación:

```
<xsl:apply-templates select="discussionForumHome messageBoard">
```

Sin esta línea, el proceso de transformación estaría completo porque ya estaba colocado el modelo "/" y se había creado una instancia de la plantilla correspondiente. El elemento <xsl:apply-templates> dice al procesador XSLT que comience una nueva búsqueda de elementos en el documento fuente XML que se correspondan don el modelo "discussionForumHome/messageBoard" y que cree una instancia de una plantilla adicional que corresponda. Este proceso de transformación es recursivo y debe estar guiado por elementos como <xsl:apply-templates>. Simplemente incluir uno o más elementos <xsl:template> en una hoja de estilo no significa que se creará una instancia de los mismos.

En el ejemplo, el elemento <xsl:apply-templates> dice al procesador XSLT que seleccione todos los elementos <discussionForumHome> del nodo actual. El nodo actual es "/" o el comienzo del documento, por lo que sólo selecciona el elemento <discussionForumHome> que aparece en el nivel raíz del documento. Si se anida profundamente otro <discussionForumHome> en el documento XML, este modelo no lo seleccionará. Suponiendo que el procesador localiza el elemento <discussionForumHome>, a continuación, éste busca todos sus hijos <messageBoard>.

Para cada hijo <messageBoard>, el procesador busca la plantilla en la hoja de estilo que ofrece la mejor correspondencia. Dado que nuestra hoja de estilo contiene una plantilla que corresponde con el modelo "messageBoard", se creará una instancia de ésta para cada elemento. La labor de esta plantilla consiste en producir un indicador del elemento de la lista HTML único para cada elemento <messageBoard>:

```
<xsl:template match="messageBoard">
   <li>
       <a href="viewForum?id={@id}">
         <xsl:value-of select="@name"/>
       </a>
   </li>
 </xsl:template>
```

Como se puede ver, el elemento de la lista habrá de estar debidamente finalizado; los indicadores <li> autónomos de estilo HTML no están permitidos porque incumplen el requisito de que las hojas de estilo XSLT estén en XML bien construido. Finalizar el elemento con también funciona con HTML, por lo que esta es la técnica que se debe emplear. En la hoja de estilo se utiliza '@' para seleccionar los valores de atributos.

Cuando se ejecuta el procesador de la hoja de estilo y se genera el árbol de resultado,

generamos el HTML que aparece en el Listado AA.2. El HTML es mínimo en este punto debido a que se trata de un ejemplo simple para mostrar el funcionamiento del procesado de un archivo XML mediante una hoja de estilo XSLT.

Listado AA.2: Código HTML generado.

```
<html>

  <head>
     <title>Discussion Forum Home Page</title>
  </head>

  <body>
     <h1>Discussion Forum Home Page </h1>
     <h3>Please select a message board to view:</h3>
     <ul>
          <li>
            <a href="viewForum?id=1">Java Programming</a>
          </li>
          <li>
            <a href="viewForum?id=2">XML Programming</a>
          </li>
          <li>
            <a href="viewForum?id=3">XSLT Questions</a>
          </li>
     </ul>
  </body>
</html>
```

Apéndice B: El *sitemap* de Lenya.

En este apéndice se muestra el listado del código del *sitemap* principal de Lenya. Es el *sitemap* en el que se realiza toda la configuración de componentes y se redirigen las peticiones hacia los *sitemaps* específicos de las publicaciones o hacia los *sitemaps* de gestión.

Listado AB.1 El sitemap de Lenya

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Copyright 1999-2004 The Apache Software Foundation

  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.
-->

<!-- $Id: sitemap.xmap 160354 2005-04-07 00:56:17Z gregor $ -->

<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">

<!-- =========================== Components ================================= -->

 <map:components>

  <!-- ========================================================================== -->
  <!-- Generators -->
  <!-- ========================================================================== -->

  <map:generators default="file">
    <map:generator name="directory" label="content,data"
logger="sitemap.generator.directory" pool-grow="2" pool-max="16" pool-min="2"
src="org.apache.cocoon.generation.DirectoryGenerator"/>
    <map:generator name="extractor" label="data" logger="sitemap.generator.extractor"
src="org.apache.cocoon.generation.FragmentExtractorGenerator"/>
    <map:generator name="file" label="content,data" logger="sitemap.generator.file"
pool-grow="4" pool-max="32" pool-min="8" src="org.apache.cocoon.generation.FileGenerator"/>
    <map:generator name="file-nolabel" label="content,data"
logger="sitemap.generator.file-nolabel" pool-grow="4" pool-max="32" pool-min="8"
src="org.apache.cocoon.generation.FileGenerator"/>
    <map:generator name="html" label="content,data"
src="org.apache.cocoon.generation.HTMLGenerator"/>
    <map:generator name="imagedirectory" label="content,data"
logger="sitemap.generator.imagedirectory"
src="org.apache.cocoon.generation.ImageDirectoryGenerator"/>
    <map:generator name="jx" label="content" logger="sitemap.generator.jx"
src="org.apache.cocoon.generation.JXTemplateGenerator"/>
    <map:generator name="notifying"
src="org.apache.cocoon.sitemap.NotifyingGenerator"/>
    <map:generator name="proxy" label="content" logger="sitemap.generator.proxy"
src="org.apache.cocoon.generation.HttpProxyGenerator"/>
    <map:generator name="request" label="data" logger="sitemap.generator.request"
pool-grow="2" pool-max="16" pool-min="2"
src="org.apache.cocoon.generation.RequestGenerator"/>
    <map:generator name="serverpages" label="content,data"
logger="sitemap.generator.serverpages" pool-grow="2" pool-max="32" pool-min="4"
src="org.apache.cocoon.generation.ServerPagesGenerator"/>
    <map:generator name="servletproxy"
src="org.apache.lenya.cms.cocoon.generation.ProxyGenerator"/>
    <map:generator name="status" label="data" logger="sitemap.generator.status" pool-
grow="2" pool-max="16" pool-min="2" src="org.apache.cocoon.generation.StatusGenerator"/>
    <map:generator name="stream" label="content,data"
logger="sitemap.generator.stream" pool-grow="2" pool-max="16" pool-min="1"
src="org.apache.cocoon.generation.StreamGenerator"/>
    <map:generator name="xpathdirectory" label="content"
logger="sitemap.generator.xpathdirectory"
src="org.apache.cocoon.generation.XPathDirectoryGenerator"/>
```

```
          <!--
          <map:generator label="content,data" logger="sitemap.generator.profiler"
name="profiler" src="org.apache.cocoon.generation.ProfilerGenerator"/>
          -->
          <map:generator name="sitetree" logger="sitemap.generator.sitetree" pool-grow="2"
pool-max="16" pool-min="2" src="org.apache.lenya.cms.cocoon.generation.SiteTreeGenerator"/>
        </map:generators>

        <!-- ================================================================================ -->
        <!-- Transformers -->
        <!-- ================================================================================ -->

        <map:transformers default="xslt">

          <map:transformer name="access-control-sitetree"
logger="lenya.sitemap.transformer.accesscontrolsitetree"
src="org.apache.lenya.cms.cocoon.transformation.AccessControlSitetreeTransformer"/>
          <map:transformer name="cinclude" logger="sitemap.transformer.cinclude" pool-
grow="2" pool-max="16" pool-min="2"
src="org.apache.cocoon.transformation.CIncludeTransformer"/>

          <map:transformer name="deli" logger="sitemap.transformer.deli" pool-grow="2" pool-
max="32" pool-min="8" src="org.apache.cocoon.transformation.DeliTransformer">
            <use-request-parameters>false</use-request-parameters>
            <use-session-parameters>false</use-session-parameters>
            <use-cookie-parameters>false</use-cookie-parameters>
            <xslt-processor-role>xalan</xslt-processor-role>
          </map:transformer>

          <map:transformer name="encodeURL" logger="sitemap.transformer.encodeURL"
src="org.apache.cocoon.transformation.EncodeURLTransformer"/>
          <map:transformer name="extractor" logger="sitemap.transformer.extractor"
src="org.apache.cocoon.transformation.FragmentExtractorTransformer"/>

          <map:transformer name="i18n" logger="sitemap.transformer.i18n"
src="org.apache.cocoon.transformation.I18nTransformer">
            <catalogues default="cmsui">
               <catalogue id="cmsui" name="cmsui" location="cocoon://i18n-catalogue/"/>
             </catalogues>
             <untranslated-text>untranslated</untranslated-text>
             <cache-at-startup>true</cache-at-startup>
          </map:transformer>

          <map:transformer name="index" logger="lenya.sitemap.transformer.index"
src="org.apache.lenya.cms.cocoon.transformation.DocumentIndexTransformer"/>
          <map:transformer name="lexer" logger="sitemap.transformer.lexer"
src="org.apache.cocoon.transformation.LexicalTransformer"/>
          <map:transformer name="ldap"
src="org.apache.cocoon.transformation.LDAPTransformer"/>
          <map:transformer name="link-rewrite" logger="lenya.sitemap.transformer.link-
rewrite" src="org.apache.lenya.cms.cocoon.transformation.LinkRewritingTransformer"/>
          <map:transformer name="log" logger="sitemap.transformer.log" pool-grow="2" pool-
max="16" pool-min="2" src="org.apache.cocoon.transformation.LogTransformer"/>
          <map:transformer name="jpath" logger="sitemap.transformer.jpath"
src="org.apache.cocoon.transformation.JPathTransformer"/>
          <map:transformer name="filter" logger="sitemap.transformer.filter"
src="org.apache.cocoon.transformation.FilterTransformer"/>
          <map:transformer name="parser" logger="sitemap.transformer.lexer"
src="org.apache.cocoon.transformation.ParserTransformer"/>
          <map:transformer name="pattern" logger="sitemap.transformer.lexer"
src="org.apache.cocoon.transformation.PatternTransformer"/>
          <map:transformer name="readDOMsession" logger="sitemap.transformer.readDOMsession"
src="org.apache.cocoon.transformation.ReadDOMSessionTransformer"/>
          <map:transformer name="session" pool-grow="4" pool-max="32" pool-min="8"
src="org.apache.cocoon.webapps.session.transformation.SessionTransformer"/>
          <map:transformer name="session-post" pool-grow="4" pool-max="32" pool-min="8"
src="org.apache.cocoon.webapps.session.transformation.SessionPostTransformer"/>
          <map:transformer name="session-pre" pool-grow="4" pool-max="32" pool-min="8"
src="org.apache.cocoon.webapps.session.transformation.SessionPreTransformer"/>
          <map:transformer name="sql" logger="sitemap.transformer.sql"
src="org.apache.cocoon.transformation.SQLTransformer"/>
          <map:transformer name="usecasemenu" logger="lenya.sitemap.transformer.usecasemenu"
src="org.apache.lenya.cms.cocoon.transformation.UsecaseMenuTransformer"/>
          <map:transformer name="workflowmenu"
logger="lenya.sitemap.transformer.workflowmenu"
src="org.apache.lenya.cms.cocoon.transformation.WorkflowMenuTransformer"/>
          <map:transformer name="writeDOMsession"
logger="sitemap.transformer.writeDOMsession"
src="org.apache.cocoon.transformation.WriteDOMSessionTransformer"/>
```

```
          <map:transformer name="write-source" logger="sitemap.transformer.write-source"
src="org.apache.cocoon.transformation.SourceWritingTransformer"/>

          <map:transformer name="xinclude" logger="sitemap.transformer.xinclude" pool-
grow="2" pool-max="16" pool-min="2"
src="org.apache.cocoon.transformation.XIncludeTransformer"/>

          <map:transformer name="xlink"
src="org.apache.lenya.cms.cocoon.transformation.IncludeTransformer">
            <publication type=""/>
          </map:transformer>

          <!-- NOTE: This is the default XSLT processor. -->
          <map:transformer logger="sitemap.transformer.xslt" name="xslt" pool-grow="2" pool-
max="32" pool-min="8" src="org.apache.cocoon.transformation.TraxTransformer">
            <use-request-parameters>false</use-request-parameters>
            <use-session-parameters>false</use-session-parameters>
            <use-cookie-parameters>false</use-cookie-parameters>
            <xslt-processor-role>xalan</xslt-processor-role>
            <check-includes>true</check-includes>
          </map:transformer>

          <!-- NOTE: This is the same as the default processor but with a different name
(for compatibility) -->
          <map:transformer logger="sitemap.transformer.xalan" name="xalan" pool-grow="2"
pool-max="32" pool-min="8" src="org.apache.cocoon.transformation.TraxTransformer">
            <use-request-parameters>false</use-request-parameters>
            <use-session-parameters>false</use-session-parameters>
            <use-cookie-parameters>false</use-cookie-parameters>
            <xslt-processor-role>xalan</xslt-processor-role>
            <check-includes>true</check-includes>
          </map:transformer>

          <!-- NOTE: You can also try XSLTC as the default processor. If you use Xalan
extensions, use the "xalan" transformer. -->
          <map:transformer logger="sitemap.transformer.xsltc" name="xsltc" pool-grow="2"
pool-max="32" pool-min="8" src="org.apache.cocoon.transformation.TraxTransformer">
            <use-request-parameters>false</use-request-parameters>
            <use-session-parameters>false</use-session-parameters>
            <use-cookie-parameters>false</use-cookie-parameters>
            <xslt-processor-role>xsltc</xslt-processor-role>
            <check-includes>true</check-includes>
          </map:transformer>
        </map:transformers>

        <!-- ========================================================================== -->
        <!-- Serializers -->
        <!-- ========================================================================== -->

        <map:serializers default="html">

          <map:serializer name="html" logger="sitemap.serializer.html" mime-type="text/html"
pool-grow="4" pool-max="32" pool-min="4"
src="org.apache.cocoon.serialization.HTMLSerializer">
            <doctype-public>-//W3C//DTD HTML 4.01 Transitional//EN</doctype-public>
            <doctype-system>http://www.w3.org/TR/html4/loose.dtd</doctype-system>
            <buffer-size>1024</buffer-size>
            <encoding>UTF-8</encoding>
          </map:serializer>
      <!-- This serializer (Cocoon-2.1.5) has an issue with namespaces -->
      <!--
          <map:serializer name="html" logger="sitemap.serializer.html" mime-type="text/html"
pool-grow="4" pool-max="32" pool-min="4"
src="org.apache.cocoon.components.serializers.HTMLSerializer">
            <encoding>UTF-8</encoding>
          </map:serializer>
      -->

          <map:serializer name="html-no-dtd" logger="sitemap.serializer.html-no-dtd" mime-
type="text/html" pool-grow="4" pool-max="32" pool-min="4"
src="org.apache.cocoon.serialization.HTMLSerializer">
            <buffer-size>1024</buffer-size>
          </map:serializer>

          <map:serializer name="htmlnoi" logger="sitemap.serializer.htmlnoi" mime-
type="text/html" pool-grow="4" pool-max="32" pool-min="4"
src="org.apache.cocoon.serialization.HTMLSerializer">
            <indent>no</indent>
            <doctype-public>-//W3C//DTD HTML 4.01 Transitional//EN</doctype-public>
```

```
            <doctype-system>http://www.w3.org/TR/html4/loose.dtd</doctype-system>
            <encoding>UTF-8</encoding>
          </map:serializer>

          <map:serializer name="xhtml" logger="sitemap.serializer.xhtml" mime-
type="text/html" pool-grow="2" pool-max="64" pool-min="2"
src="org.apache.cocoon.serialization.XMLSerializer">
            <doctype-public>-//W3C//DTD XHTML 1.0 Strict//EN</doctype-public>
            <doctype-system>http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd</doctype-
system>
            <encoding>UTF-8</encoding>
          </map:serializer>
      <!-- This serializer (Cocoon-2.1.5) has an issue with namespaces -->
      <!--
          <map:serializer name="xhtml" logger="sitemap.serializer.xhtml" mime-
type="text/html" pool-grow="2" pool-max="64" pool-min="2"
src="org.apache.cocoon.components.serializers.XHTMLSerializer">
            <encoding>UTF-8</encoding>
          </map:serializer>
      -->

          <map:serializer name="xml" logger="sitemap.serializer.xml" mime-type="text/xml"
src="org.apache.cocoon.serialization.XMLSerializer">
            <encoding>UTF-8</encoding>
          </map:serializer>

      <!-- This serializer (Cocoon-2.1.5) has an issue with namespaces -->
      <!--
          <map:serializer name="xml" logger="sitemap.serializer.xml" mime-type="text/xml"
src="org.apache.cocoon.components.serializers.XMLSerializer">
            <encoding>UTF-8</encoding>
          </map:serializer>
      -->

          <map:serializer name="links" logger="sitemap.serializer.links"
src="org.apache.cocoon.serialization.LinkSerializer"/>

          <map:serializer name="wml" logger="sitemap.serializer.wml" mime-
type="text/vnd.wap.wml" src="org.apache.cocoon.serialization.XMLSerializer">
           <doctype-public>-//WAPFORUM//DTD WML 1.1//EN</doctype-public>
           <doctype-system>http://www.wapforum.org/DTD/wml_1.1.xml</doctype-system>
           <encoding>ASCII</encoding>
           <omit-xml-declaration>yes</omit-xml-declaration>
          </map:serializer>

          <map:serializer name="svgxml" logger="sitemap.serializer.svgxml" mime-
type="image/svg-xml" src="org.apache.cocoon.serialization.XMLSerializer">
           <doctype-public>-//W3C//DTD SVG 20000303 Stylable//EN</doctype-public>
           <doctype-system>http://www.w3.org/TR/2000/03/WD-SVG-20000303/</doctype-system>
          </map:serializer>

          <map:serializer name="text" logger="sitemap.serializer.text" mime-
type="text/plain" src="org.apache.cocoon.serialization.TextSerializer">
            <encoding>UTF-8</encoding>
          </map:serializer>

          <map:serializer name="text-iso-8859-1" logger="sitemap.serializer.text-iso-8859-1"
mime-type="text/plain" src="org.apache.cocoon.serialization.TextSerializer">
            <encoding>ISO-8859-1</encoding>
          </map:serializer>

          <map:serializer name="fo2pdf" logger="sitemap.serializer.fo2pdf" mime-
type="application/pdf" src="org.apache.cocoon.serialization.FOPSerializer"/>
          <map:serializer name="fo2ps" logger="sitemap.serializer.fo2ps" mime-
type="application/postscript" src="org.apache.cocoon.serialization.FOPSerializer"/>
          <map:serializer name="fo2pcl" logger="sitemap.serializer.fo2pcl" mime-
type="vnd.hp-PCL" src="org.apache.cocoon.serialization.FOPSerializer"/>
          <map:serializer name="svgxml" logger="sitemap.serializer.svgxml" mime-
type="image/svg-xml" src="org.apache.cocoon.serialization.XMLSerializer">
            <doctype-public>-//W3C//DTD SVG 20000303 Stylable//EN</doctype-public>
            <doctype-system>http://www.w3.org/TR/2000/03/WD-SVG-20000303/</doctype-system>
          </map:serializer>
          <map:serializer name="svg2jpeg" logger="sitemap.serializer.svg2png" mime-
type="image/jpeg" src="org.apache.cocoon.serialization.SVGSerializer">
            <parameter name="quality" type="float" value="0.9"/>
          </map:serializer>
          <map:serializer name="svg2png" logger="sitemap.serializer.svg2png" mime-
type="image/png" src="org.apache.cocoon.serialization.SVGSerializer"/>
```

```
        </map:serializers>

        <!-- ========================================================================== -->
        <!-- Readers -->
        <!-- ========================================================================== -->

        <map:readers default="resource">
          <map:reader name="resource" logger="sitemap.reader.resource" pool-max="32"
src="org.apache.cocoon.reading.ResourceReader"/>
        </map:readers>

        <!-- ========================================================================== -->
        <!-- Matchers -->
        <!-- ========================================================================== -->

        <map:matchers default="wildcard">
          <map:matcher name="wildcard" logger="sitemap.matcher.wildcard"
src="org.apache.cocoon.matching.WildcardURIMatcher"/>
          <map:matcher name="regexp" logger="sitemap.matcher.regexp"
src="org.apache.cocoon.matching.RegexpURIMatcher"/>
          <map:matcher name="request-parameter" logger="sitemap.matcher.request-parameter"
src="org.apache.cocoon.matching.RequestParameterMatcher"/>
          <map:matcher name="referer-match" logger="sitemap.matcher.referer-match"
src="org.apache.cocoon.matching.WildcardHeaderMatcher">
            <header-name>referer</header-name>
          </map:matcher>
          <map:matcher name="usecase" logger="sitemap.matcher.usecase"
src="org.apache.cocoon.matching.WildcardRequestParameterMatcher">
            <parameter-name>lenya.usecase</parameter-name>
          </map:matcher>
          <map:matcher name="step" logger="sitemap.matcher.step"
src="org.apache.cocoon.matching.WildcardRequestParameterMatcher">
            <parameter-name>lenya.step</parameter-name>
          </map:matcher>
          <map:matcher name="cookie" logger="sitemap.matcher.cookie"
src="org.apache.cocoon.matching.CookieMatcher"/>
          <map:matcher name="header" logger="sitemap.matcher.header"
src="org.apache.cocoon.matching.HeaderMatcher"/>
          <map:matcher name="parameter" logger="sitemap.matcher.parameter"
src="org.apache.cocoon.matching.ParameterMatcher"/>
          <map:matcher name="sessionstate" logger="sitemap.matcher.sessionstate"
src="org.apache.cocoon.matching.WildcardSessionAttributeMatcher">
            <attribute-name>org.apache.cocoon.SessionState</attribute-name>
          </map:matcher>
        </map:matchers>

        <!-- ========================================================================== -->
        <!-- Selectors -->
        <!-- ========================================================================== -->

        <map:selectors default="browser">

         <map:selector name="browser" logger="sitemap.selector.browser"
src="org.apache.cocoon.selection.BrowserSelector">
          <!--+
              | NOTE: The appearance indicates the search order. This is very important
since
              |       some words may be found in more than one browser description. (MSIE is
              |       presented as "Mozilla/4.0 (Compatible; MSIE 4.01; ...")
              +-->
          <browser name="explorer" useragent="MSIE"/>
          <browser name="pocketexplorer" useragent="MSPIE"/>
          <browser name="handweb" useragent="HandHTTP"/>
          <browser name="avantgo" useragent="AvantGo"/>
          <browser name="imode" useragent="DoCoMo"/>
          <browser name="opera" useragent="Opera"/>
          <browser name="lynx" useragent="Lynx"/>
          <browser name="java" useragent="Java"/>
          <browser name="wap" useragent="Nokia"/>
          <browser name="wap" useragent="UP"/>
          <browser name="wap" useragent="Wapalizer"/>
          <browser name="mozilla5" useragent="Mozilla/5"/>
          <browser name="mozilla5" useragent="Netscape6/"/>
          <browser name="netscape" useragent="Mozilla"/>
         </map:selector>
         <map:selector name="exception" logger="sitemap.selector.exception"
src="org.apache.cocoon.selection.ExceptionSelector">
           <exception name="sax" class="org.xml.sax.SAXException" unroll="true"/>
           <exception name="resourcenotfound"
```

```
class="org.apache.cocoon.ResourceNotFoundException" unroll="true"/>
          <exception name="document-does-not-exist"
class="org.apache.lenya.cms.publication.DocumentDoesNotExistException"/>
          <exception name="invalid-continuation"
class="org.apache.cocoon.components.flow.InvalidContinuationException"/>
          <exception class="org.apache.cocoon.ProcessingException" unroll="true"/>
          <!-- The statement below tells the selector to unroll as much exceptions as
possible -->
          <exception class="java.lang.Throwable" unroll="true"/>
         </map:selector>
        <map:selector name="header" logger="sitemap.selector.header"
src="org.apache.cocoon.selection.HeaderSelector">
         <!-- <header-name>myparam</header-name> -->
        </map:selector>
        <map:selector name="host" logger="sitemap.selector.host"
src="org.apache.cocoon.selection.HostSelector"/>
        <map:selector name="parameter" logger="sitemap.selector.parameter"
src="org.apache.cocoon.selection.ParameterSelector"/>
        <map:selector name="request-attribute" logger="sitemap.selector.request-attribute"
src="org.apache.cocoon.selection.RequestAttributeSelector">
         <!-- <attribute-name>myparam</attribute-name> -->
        </map:selector>
        <map:selector name="request-method" logger="sitemap.selector.request-method"
src="org.apache.cocoon.selection.RequestMethodSelector"/>
        <map:selector name="request-parameter" logger="sitemap.selector.request-parameter"
src="org.apache.cocoon.selection.RequestParameterSelector">
         <!-- Define now which request parameter to use; or do it later,
              when using this selector, via "parameter-name" parameter.
         <parameter-name>myparam</parameter-name>
         -->
        </map:selector>
        <map:selector name="resource-exists" logger="sitemap.selector.resource-exists"
src="org.apache.cocoon.selection.ResourceExistsSelector"/>
        <map:selector name="session-attribute" logger="sitemap.selector.session-attribute"
src="org.apache.cocoon.selection.SessionAttributeSelector">
         <!-- <attribute-name>myparam</attribute-name> -->
        </map:selector>
       </map:selectors>

       <!-- ========================================================================= -->
       <!-- Actions -->
       <!-- ========================================================================= -->

       <map:actions>
        <map:action name="authenticator"
src="org.apache.lenya.cms.cocoon.acting.DelegatingAuthenticatorAction"
logger="lenya.sitemap.action.authenticator"/>
        <map:action name="authorizer"
src="org.apache.lenya.cms.cocoon.acting.DelegatingAuthorizerAction"
logger="lenya.sitemap.action.authorizer"/>
        <map:action name="clear-cache" logger="sitemap.action.clear-cache"
src="org.apache.cocoon.acting.ClearCacheAction"/>
        <map:action name="clear-persistent-store" logger="sitemap.action.clear-persistent-
store" src="org.apache.cocoon.acting.ClearPersistentStoreAction"/>
        <map:action name="default-create"
src="org.apache.lenya.cms.cocoon.acting.DefaultCreatorAction"
logger="sitemap.action.default-create">
          <tree-authoring href="content/authoring/sitetree.xml"/>
          <docs href="content/authoring"/>
          <doctypes href="config/doctypes/"/>
        </map:action>
        <map:action name="document-id-exists" logger="sitemap.action.document-id-exists"
src="org.apache.lenya.cms.cocoon.acting.DocumentIdExistsAction" />
        <map:action name="form-validator" logger="sitemap.action.form-validator"
src="org.apache.cocoon.acting.FormValidatorAction"/>
        <map:action name="language-exists" logger="lenya.sitemap.action.language-exists"
src="org.apache.lenya.cms.cocoon.acting.LanguageExistsAction"/>
        <map:action name="oneformeditorsave" logger="sitemap.action.oneformeditormsave"
src="org.apache.lenya.cms.cocoon.acting.OneFormEditorSaveAction" />
        <map:action name="parent-child" logger="sitemap.action.parent-child"
src="org.apache.lenya.cms.cocoon.acting.ParentChildCreatorAction">
          <tree-authoring href="content/authoring/tree.xml"/>
          <docs href="content/authoring"/>
          <doctypes href="config/doctypes/"/>
        </map:action>
        <map:action name="request" logger="sitemap.action.request"
src="org.apache.cocoon.acting.RequestParamAction"/>
        <map:action name="request-parameter-exists"
```

```
logger="sitemap.action.requestParameterExists"
src="org.apache.cocoon.acting.RequestParameterExistsAction" />
          <map:action name="reserved-checkin"
src="org.apache.lenya.cms.cocoon.acting.ReservedCheckinAction"
logger="sitemap.action.reserved-checkin"/>
          <map:action name="reserved-checkout"
src="org.apache.lenya.cms.cocoon.acting.ReservedCheckoutAction"
logger="sitemap.action.reserved-checkout"/>
          <map:action name="reserved-checkout-test"
src="org.apache.lenya.cms.cocoon.acting.ReservedCheckoutTestAction"
logger="sitemap.action.reserved-checkout-test"/>
          <map:action name="resource-exists" logger="sitemap.action.resource-exists"
src="org.apache.cocoon.acting.ResourceExistsAction"/>
          <map:action name="resource-exists-enhanced" logger="sitemap.action.resource-exists-
enhanced" src="org.apache.lenya.cms.cocoon.acting.ResourceExistsAction"/>
          <map:action name="rollback"
src="org.apache.lenya.cms.cocoon.acting.RollbackAction"/>
          <map:action name="save" logger="sitemap.action.save"
src="org.apache.lenya.cms.cocoon.acting.HTMLFormSaveAction" />
          <map:action name="session"
src="org.apache.cocoon.webapps.session.acting.SessionAction"/>
          <map:action name="session-form"
src="org.apache.cocoon.webapps.session.acting.SessionFormAction"/>
          <map:action name="session-isvalid" logger="sitemap.action.session-isvalid"
src="org.apache.cocoon.acting.SessionIsValidAction"/>
          <map:action name="session-propagator" logger="sitemap.action.session-propagator"
src="org.apache.cocoon.acting.SessionPropagatorAction"/>
          <map:action name="session-state" logger="sitemap.action.session-state"
src="org.apache.cocoon.acting.SessionStateAction"/>
          <map:action name="set-header" logger="sitemap.action.set-header"
src="org.apache.cocoon.acting.HttpHeaderAction"/>
          <map:action name="task" logger="sitemap.action.task"
src="org.apache.lenya.cms.cocoon.acting.TaskAction"/>
          <map:action name="upload" logger="sitemap.action.upload"
src="org.apache.lenya.cms.cocoon.acting.UploadAction">
            <resources-root href="resources/images/live"/>
            <docs-root href="content/authoring"/>
            <meta-root href="content/authoring/"/>
             <insert-image-before value="false"/>
          </map:action>
          <map:action name="uriparametrizer"
src="org.apache.lenya.cms.cocoon.acting.URIParametrizerAction"
logger="sitemap.action.uriparametrizer"/>
          <map:action name="validate" logger="sitemap.action.validateAction"
src="org.apache.lenya.cms.cocoon.acting.ValidateAction" />
          <map:action name="workflow" logger="sitemap.action.workflow"
src="org.apache.lenya.cms.cocoon.acting.WorkflowInvokerAction"/>
          <map:action name="xopushandler" logger="sitemap.action.xopus"
src="org.apache.lenya.cms.cocoon.acting.XopusHandlerAction">
            <xml href="content/authoring/"/>
            <xsl href="xslt"/>
            <xsd href="config/doctypes/schemas"/>
            <temp href="temp"/>
            <rcmlDirectory href="content/rcml"/>
            <backupDirectory href="content/rcbak"/>
          </map:action>
        </map:actions>

          <map:pipes default="caching">
          <!--<map:pipes default="profile-caching">--> <!-- NOTE: Enables profiling, but
makes BEWARE if slowing down the processing! -->
            <map:pipe name="caching"
src="org.apache.cocoon.components.pipeline.impl.CachingProcessingPipeline"/>
            <map:pipe name="caching-point"
src="org.apache.cocoon.components.pipeline.impl.CachingPointProcessingPipeline">
              <autoCachingPoint>On</autoCachingPoint>
            </map:pipe>
            <map:pipe name="noncaching"
src="org.apache.cocoon.components.pipeline.impl.NonCachingProcessingPipeline"/>
            <!-- The following two can be used for profiling:-->
            <map:pipe name="profile-caching"
src="org.apache.cocoon.components.profiler.ProfilingCachingProcessingPipeline"/>
            <map:pipe name="profile-noncaching"
src="org.apache.cocoon.components.profiler.ProfilingNonCachingProcessingPipeline"/>
          </map:pipes>

       </map:components>

      <!-- ========================== Views ================================== -->
```

```
    <!--+
        | Views provide diffent, well, views to resources. Views are
        | orthogonal to pipelines. Please refer to the docs.
        +-->
    <map:views>

      <map:view name="first" from-position="first">
        <map:serialize type="xml"/>
      </map:view>

      <map:view name="last" from-position="last">
        <map:serialize type="xml"/>
      </map:view>

      <map:view from-label="content" name="content">
        <map:serialize type="xml"/>
      </map:view>

      <map:view from-label="data" name="pretty-content">
        <map:transform src="stylesheets/simple-xml2html.xsl"/>
        <map:serialize type="html"/>
      </map:view>

      <map:view from-position="last" name="links">
        <map:serialize type="links"/>
      </map:view>

      <map:view from-label="aggregate" name="aggregate">
        <map:serialize type="xml"/>
      </map:view>
    </map:views>

  <!-- ========================== Resources ================================= -->

    <map:resources>

      <map:resource name="style-cms-page">
        <map:transform type="i18n">
          <map:parameter name="locale" value="{request:locale}"/>
        </map:transform>
        <map:transform src="cocoon://lenya-screen.xsl"/>
        <map:transform src="lenya/xslt/util/strip_namespaces.xsl"/>
        <map:serialize/>
      </map:resource>

   </map:resources>

  <!-- ========================== Pipelines ================================= -->

    <map:pipelines>

      <!--
      <map:pipeline>
        <map:match pattern="profiler.html">
          <map:generate type="profiler"/>
          <map:transform src="lenya/xslt/admin/profile2html.xsl">
            <map:parameter name="use-request-parameters" value="true"/>
           </map:transform>
           <map:serialize type="html"/>
        </map:match>
      </map:pipeline>
      -->

      <!-- I18N  -->
      <map:pipeline>
        <map:match pattern="i18n-catalogue/**">
          <map:mount uri-prefix="" src="lenya/i18n.xmap" check-reload="true" reload-
method="synchron"/>
        </map:match>
      </map:pipeline>

      <map:pipeline>

        <!-- Lenya GUI screen -->
        <map:match pattern="lenya-screen.xsl">
          <map:generate src="{fallback:xslt/util/page2xhtml.xsl}"/>
          <map:transform src="lenya/xslt/util/page2xslt.xsl">
            <map:parameter name="contextprefix" value="{request:contextPath}"/>
```

```
              </map:transform>
              <map:serialize type="xml"/>
            </map:match>

            <map:match type="usecase" pattern="login">

              <map:match type="step" pattern="showscreen">
                <map:generate type="serverpages" src="{fallback:content/ac/login.xsp}"/>
                <map:transform src="{fallback:xslt/ac/login.xsl}">
                  <map:parameter name="publication_name" value="{page-envelope:publication-
id}"/>
                </map:transform>
                <map:call resource="style-cms-page"/>
              </map:match>

              <map:match type="step" pattern="login">
                <map:act type="authenticator">
                  <map:redirect-to uri="{request:requestURI}" session="true"/>
                </map:act>
                <map:redirect-to uri="{request:requestURI}?
lenya.usecase=login&amp;lenya.step=showscreen&amp;status=failed" session="true"/>
              </map:match>

            </map:match>

            <map:match type="usecase" pattern="logout">
              <map:generate type="serverpages" src="{fallback:content/ac/logout.xsp}"/>
              <map:transform src="{fallback:xslt/ac/logout.xsl}">
                <map:parameter name="publication_name" value="{page-envelope:publication-
id}"/>
                <map:parameter name="contextprefix" value="{request:contextPath}"/>
              </map:transform>
                <map:call resource="style-cms-page"/>
            </map:match>

          <!-- favicon -->
          <map:match pattern="favicon.ico">
            <map:read mime-type="image/x-icon" src="lenya/resources/images/lenya.ico"/>
          </map:match>

          <!-- images -->
          <map:match pattern="images/*.gif">
            <map:read mime-type="images/gif" src="resources/images/{1}.gif"/>
          </map:match>

          <!-- CSS stylesheets -->
          <map:match pattern="styles/*.css">
            <map:read mime-type="text/css" src="resources/styles/{1}.css"/>
          </map:match>

          <!-- JavaScript scripts -->
          <map:match pattern="scripts/*.js">
            <map:read mime-type="text/javascript" src="resources/scripts/{1}.js"/>
          </map:match>

            <map:handle-errors>
              <map:generate type="notifying"/>
              <map:transform src="stylesheets/system/error2html.xslt">
                <map:parameter name="contextPath" value="{request:contextPath}"/>
              </map:transform>
              <map:serialize type="html"/>
            </map:handle-errors>

          </map:pipeline>

          <map:pipeline internal-only="true">
            <map:match pattern="**">
              <map:mount uri-prefix="" src="global-sitemap.xmap" check-reload="true" reload-
method="synchron"/>
            </map:match>

            <map:handle-errors>
              <map:generate type="notifying"/>
              <map:transform src="stylesheets/system/error2html.xslt">
                <map:parameter name="contextPath" value="{request:contextPath}"/>
              </map:transform>
              <map:serialize type="html"/>
            </map:handle-errors>
```

```
          </map:pipeline>

          <map:pipeline>
            <map:match pattern="**">
              <map:act type="authorizer">
                <map:mount uri-prefix="" src="global-sitemap.xmap" check-reload="true"
reload-method="synchron"/>
              </map:act>
              <map:redirect-to uri="{request:requestURI}?
lenya.usecase=login&amp;lenya.step=showscreen" session="true"/>
            </map:match>

            <map:handle-errors>
              <map:generate type="notifying"/>
              <map:transform src="stylesheets/system/error2html.xslt">
                <map:parameter name="contextPath" value="{request:contextPath}"/>
              </map:transform>
              <map:serialize type="html"/>
            </map:handle-errors>

          </map:pipeline>

        </map:pipelines>

      </map:sitemap>
```

Apéndice C: Generación de una página en Lenya.

En este Apéndice se tratarán de esbozar los mecanismos que utiliza Lenya para la generación de una página como resultado de una petición. Para observar los procedimientos que dan lugar a la compleción de la operación, se ha cambiado el nivel de depuración de la aplicación, cambiando el valor de *logging* de *ERROR* a *DEBUG* en el archivo WEB-INF/logkit.xconf y se ha observado la salida en el archivo WEB-INF/logs/sitemap.log filtrando por el término *DEBUG*.

Tecleando la URL *http://localhost:8080/lenya/default/authoring/index.html*, Lenya nos redirigirá a la página de inicio de sesión de la publicación por defecto.

Observando el archivo *sitemap.log* vemos que se escribe la siguiente entrada:

```
"Matcher 'wildcard' matched prepared pattern '*/**' at webapps/lenya/sitemap.xmap:760:33"
```

Que corresponde a :

```
<!-- Resources -->
<map:match pattern="*/**">
  <map:act type="resource-exists">
    <map:parameter name="url" value="lenya/pubs/{1}/resources/{2}"/>
      <map:mount uri-prefix="" src="lenya/resources.xmap" check-reload="true" reload-method="synchron"/>
  </map:act>
</map:match>
```

No se registra una coincidencia, así que se continúa el viaje a través del *sitemap*. La siguiente línea en el log es:

```
"Matcher 'wildcard' matched prepared pattern '*/**' at webapps/lenya/sitemap.xmap:768:33"
```

Que corresponde a:

```
<!-- Enter the actual publication -->
<map:match pattern="*/**">
     <map:mount uri-prefix="{1}" src="lenya/pubs/{1}/sitemap.xmap" check-reload="true" reload-method="synchron"/>
</map:match>
```

En este punto, los valores para las claves del *sitemap* son:

- {0} default/live/index.html
- {1} default
- {2} live/index.html

A través de este patrón, se monta el *sitemap* específico de la publicación por defecto, esto es, webapps/lenya/lenya/pubs/default/sitemap.xmap.

La siguiente línea en el log es:

```
"Matcher 'wildcard' matched prepared pattern '**' at webapps/lenya/lenya/default/sitemap.xmap:93:35"
```

Que se corresponde a:

```
<map:match pattern="**">
  <map:mount uri-prefix="" src="publication-sitemap.xmap"/>
</map:match>
```

Por lo que ahora estamos en el dominio del *publication-sitemap.xmap*.

```
"Matcher 'wildcard' matched 'wildcard' matched prepared pattern '**.html' at webapps/lenya/lenya/pubs/default/publication-sitemap.xmap:104:36"
```

Ese *match* se encuentra en el siguiente trozo de código:

```
<map:match pattern="**.html">
   <map:act type="uriparametrizer" src="{1}.html">
     <map:parameter name="doctype"
       value="cocoon://uri-parameter/{page-envelope:publication-id}/doctype"/>
     <map:parameter name="document-id"
         value="cocoon://uri-parameter/{page-envelope:publication-id}/document-id"/>
     <map:aggregate element="lenya" label="aggregation">
       <map:part src="cocoon:/lenyamenubar/{page-envelope:area}/article.xml"/>
       <map:part
                  src="cocoon:/lenyabody/{page-envelope:publication-id}/{page-envelope:area}/{doctype}/{document-id}"/>
     </map:aggregate>
     <map:transform src="xslt/{doctype}-{page-envelope:area}.xsl"/>
   </map:act>
   <map:serialize type="html"/>
</map:match>
```

En este punto se llama a la acción *uriparametrizer* que se encarga de estudiar la URI que le ha llegado y obtener de ella una colección de parámetros, el primero de los cuales es el *doctype.* Esta acción, llama de nuevo al *sitemap* principal de Lenya mediante `cocoon://uri-parameter/default/doctype/live/index.html`

que se intercepta en el *sitemap* pirncipal de Lenya, webapps/lenya/sitemap.xmap:665:51

```
<!-- uri-parameter/{publication-id}/{parameter}/{area}/{uri} -->
<map:match pattern="uri-parameter/*/*/*/**">
  <map:mount uri-prefix="uri-parameter/{1}/{2}/"
          src="lenya/pubs/{1}/parameter-{2}.xmap"
          check-reload="true" reload-method="synchron"/>
</map:match>
```

Al capturar la URI, se provoca el montado del *parameter-doctype.xmap* que ejecuta una página XSP para generar el valor del parámetro. Este procedimiento se realiza tanto para el parámetro *doctype* como para el *document-id*.

Estos valores se usarán en el siguiente paso, una agregación de contenido. Esta acción agregación de contenido construye la barra de menú usando una página XSP

```
<map:match pattern="lenyamenubar/live/article.xml">
     <map:generate type="serverpages" src="../../content/menus/live.xsp"/>
     <map:serialize type="xml"/>
</map:match>
```

En este punto, llama a la *pipeline lenyabody* para generar el resto de los componentes de la página.

```
<!-- This is the pipeline that builds the page. It aggregates all
the navigational elements (breadcrumb, tabs, menu) with the actual
content of the document. -->
<map:pipeline>
<!-- /lenyabody/{publication-id}/{area}/{doctype}/{document-id} -->
  <map:match pattern="lenyabody/*/*/*/**">
    <map:aggregate element="cmsbody">
    <!-- <map:aggregate element="body"
```

```
        prefix="page" ns="http://www.lenya.org/2003/page"
        label="aggregation"> -->
    <map:part src="cocoon://navigation/{1}/{2}/breadcrumb/{4}.xml"/>
    <map:part src="cocoon://navigation/{1}/{2}/tabs/{4}.xml"/>
    <map:part src="cocoon://navigation/{1}/{2}/menu/{4}.xml"/>
    <map:part src="cocoon://{1}/{2}/{3}/{4}.xml"/>
   </map:aggregate>
   <map:transform src="xslt/page2xhtml.xsl">
    <map:parameter name="root" value="{page-envelope:context-prefix}/{1}"/>
    </map:transform>
    <map:serialize type="xml"/>
  </map:match>
</map:pipeline>
```

Como paso final, se llama al serializador y se devuelve el resultado de la petición.

Apéndice D: Creación de una página en Lenya

Para realizar el seguimiento de las acciones que se llevan a cabo para crear un nuevo documento en Lenya, hemos procedido de igual forma que en el Apéndice B. Los números que se hayan después de los ':' en los nombres de los archivos indican el número de línea.

Comencemos creando una nueva página en la sección tutorial de la publicación por defecto. Para ello deberemos solicitar la URL (aunque también podría realizarse esta tarea a través del menú del CMS):

```
http://localhost:8080/lenya/default/authoring/tutorial.html?lenya.usecase=create&lenya.step=showscreen&doctype=xhtml
```

Mediante esta URL le estamos diciendo a Lenya que debe crear una página nueva, con contenido en XHTML, dentro de la sección tutorial de la publicación por defecto.

En el log observamos que la petición ha generado la siguiente entrada:

```
"Matcher 'usecase' matched prepared pattern 'create' at webapps/lenya/sitemap.xmap:703:52"
```

Observando el archivo `webapps/lenya/sitemap.xmap:703:52`:

```
<map:match type="usecase" pattern="create">
  <map:mount uri-prefix=""
    src="lenya/usecase.xmap" check-reload="true"
    reload-method="synchron"/>
</map:match>
```

Vemos que al capturar el caso de uso, Lenya monta el *sitemap usecase.xmap.*

El *mactcher* para los casos de uso está definido en `webapps/lenya/sitemap.xmap:279`.

```
<map:matcher name="usecase" logger="sitemap.matcher.usecase"
  src="org.apache.cocoon.matching.WildcardRequestParameterMatcher">
  <parameter-name>lenya.usecase</parameter-name>
</map:matcher>
```

Por tanto, lo único que se ha hecho es definir un *WildcardRequestParameterMatcher* que busca el parámetro `lenya.usecase` en la petición y devuelve que lo ha encontrado si está presente.

La petición continúa hacia `webapps/lenya/usecase.xmap` a través de `"Matcher 'wildcard' matched prepared pattern '*/*/**' at webapps/lenya/lenya/usecase.xmap:119:33"`

Este grupo no captura el caso de uso, así que pasamos al siguiente:

```
"Matcher 'wildcard' matched prepared pattern '*/*/**' at webapps/lenya/lenya/usecase.xmap:210:33"
```

De nuevo, no se captura, así que se continúa buscando:

```
"Matcher 'wildcard' matched prepared pattern '*/*/**' at webapps/lenya/lenya/usecase.xmap:255:35"
```

Se ha producida la captura del caso de uso. Mirando en el interior del *usecase.xmap* en la línea especificada, encontramos lo siguiente:

```
"Matcher 'usecase' matched prepared pattern 'create' at webapps/lenya/lenya/usecase.xmap:257:46"
```

```
<!-- Create -->
<map:pipeline>
```

```
  <!-- {publication-id}/{area}/{uri}-->
  <map:match pattern="*/*/**">

    <map:match type="usecase" pattern="create">

      <map:match type="step" pattern="showscreen">
        <map:call resource="usecase-create">
          <map:parameter name="publication-id" value="{../../1}"/>
        </map:call>
      </map:match>

      <map:match type="step" pattern="create">
        <map:act type="default-create">
          <!-- if the action suceeds it returns the referer and we -->
          <!-- simply redirect to it. -->
          <map:redirect-to uri="{request:requestURI}"/>
        </map:act>
        <!-- otherwise the action could not validate some of the -->
        <!-- input and we present the upload form again -->
        <map:call resource="usecase-create">
          <map:parameter name="publication-id" value="{../../1}"/>
        </map:call>
     </map:match>

   </map:match>
  </map:match>
</map:pipeline>
```

Lenya sabe que tiene que buscar un patrón `create` porque el *WildCardParameterRequest* ha asignado el valor de `lenya.usecase` a un parámetro usado por el *sitemap* montado.

Dentro de `<map:match type="usecase" pattern="create">`, Lenya busca un patrón para el parámetro de la petición `lenya.step`. El *matcher* `step` es otra instancia del *WildCardParameterMatcher*. En esta ocasión se captura el valor del parámetro, `showscreen`:

```
<map:match type="step" pattern="showscreen">
  <map:call resource="usecase-create">
    <map:parameter name="publication-id" value="{../../1}"/>
  </map:call>
</map:match>
```

Un recurso es un trozo de *pipeline* reutilizable. El recurso `usecase-create` se define al comienzo del archivo *usecase.xmap.*

Hay que notar que al recurso se le pasa como parámetro el identificador de la publicación, en este caso, *default*.

Hagamos un inciso para entender mejor el funcionamiento de los parámetros y los valores a lo largo del recorrido por los *sitemaps*.

Cuando se entra en el *sitemap*, la primera captura con el *usecase matcher* provoca que se le dé le valor de `create` al parámetro `usecase`.

En siguiente lugar se capturó el patrón `*/*/**`, del que se extraen los parámetros de las partes de la URI.

- 0 -> `default/authoring/tutorial.html`
- 1 -> `default`
- 2 -> `authoring`
- 3 -> `tutorial.html`

En siguiente lugar se capturó el valor `showscreen` para el parámetro `step`.

En este punto, se tienen tres niveles de parámetros. Se pueden encontrar en el log como:

```
PARAM: '0' VALUE: 'showscreen'
LEVEL 2
PARAM: '../0' VALUE: 'create'
LEVEL 1
PARAM: '../../3' VALUE: 'tutorial.html'
PARAM: '../../2' VALUE: 'authoring'
PARAM: '../../0' VALUE: 'default/authoring/tutorial.html'
PARAM: '../../1' VALUE: 'default'
```

Cada captura nueva provoca que la pila de parámetros baje un nivel. Así que para llamar a parámetros almacenados, se usa la notación `../` .

Volviendo al recurso `usecase-create` :

```
<map:resource name="usecase-create">
  <!-- use publication-specific schema if available -->
  <map:act type="resource-exists"
    src="pubs/{publication-id}/content/authoring/create.xsp">
    <map:generate type="serverpages"
      src="pubs/{../publication-id}/content/authoring/create.xsp"/>
    <map:transform src="pubs/{../publication-id}/xslt/authoring/create.xsl"/>
    <map:serialize />
  </map:act>

  <!-- otherwise use default schema -->
  <map:generate type="serverpages" src="content/authoring/create.xsp"/>
    <map:transform src="xslt/authoring/create.xsl"/>
  <map:serialize />
 </map:resource>
```

Vemos que en primer lugar se busca si hay una página XSP asociada a la publicación mediante la acción *ResourceExistsAction*.

Como existe esta página XSP, se ejecuta para generar el XML de la página. Éste pregunta por el identificador de la página (para la URI) y por su nombre (para el *sitemap* y la navegación). Después de estas operaciones se procede a su transformación mediante la hoja de estilo XSLT apropiada y se devuelve el formulario de creación de nueva página al navegador.

Apéndice E: Comparativa funcional.

Esta comparativa esta tomada de [CMSMATRIX2006] y, aunque el grado de fiabilidad es alto, son los propios usuarios de los portales lo que votan en muchos de los apartados y no un organismo independiente.

Producto	Alfresco Enterprise + Community	Apache Lenya 1.2.3	OpenCms 6.2.0
Última actualización	**4/26/2006**	**4/24/2005**	**04/10/06**
Necesidades del Sistema	**Alfresco**	**Apache Lenya**	**OpenCms**
Application Server	Any that supports Java 1.5	4 or more	Tomcat, JBoss, Bea Weblogic...
Approximate Cost	$7500/CPU/year (supported)		Free
Database	Hibernate-supported DBs	Any	MySQL, PostGreSQL, Oracle, MSSQL
License	Mozilla Public License	Apache-style	GNU LGPL
Operating System	Any	Any	Any
Programming Language	+ PHP interface	Java/XML/XSLT/Java script/JSP	Java 1.3+
Root Access	No	Yes	No
Shell Access	No	Yes	No
Web Server	Any (Can depend on applicatoin server)	Any	Tomcat, Apache, IIS
Security	**Alfresco**	**Apache Lenya**	**OpenCms**
Audit Trail	Yes	Yes	Yes
Captcha	No	No	No
Content Approval	Yes	Yes	Yes
Email Verification	No	No	No
Granular Privileges	Yes	Yes	Yes
Kerberos Authentication	Yes	No	No
LDAP Authentication	Yes	Yes	Costs Extra
Login History	No	Yes	Yes
NIS Authentication	Yes	No	No
NTLM Authentication	Yes	Yes	No
Pluggable Authentication	Yes	Yes	Costs Extra
Problem Notification	No	Yes	Yes
Sandbox	Yes	Yes	Yes

Session Management	Yes	Yes	No
SMB Authentication	Yes	Yes	No
SSL Compatible	Yes	Yes	Yes
SSL Logins	Yes	Yes	No
SSL Pages	Yes	Yes	Yes
Versioning	Yes	Yes	Yes
Support	**Alfresco**	**Apache Lenya**	**OpenCms**
Certification Program	Yes	No	No
Code Skeletons	Yes		No
Commercial Manuals	No	No	Yes
Commercial Support	Yes	Yes	Yes
Commercial Training	Yes	Yes	Yes
Developer Community	Yes	Yes	Yes
Online Help	Yes	Yes	Yes
Pluggable API	Yes	Yes	Yes
Professional Hosting	Yes	Yes	No
Professional Services	Yes	Yes	Yes
Public Forum	Yes	Yes	Yes
Public Mailing List	Yes	Yes	Yes
Test Framework	Yes		No
Third-Party Developers	Yes	Yes	Yes
Users Conference	No	Yes	No
Ease of Use	**Alfresco**	**Apache Lenya**	**OpenCms**
Drag-N-Drop Content	Yes	No	Limited
Email To Discussion	No	No	No
Friendly URLs	Yes	Yes	Yes
Image Resizing	Yes	No	Yes
Macro Language	Yes	No	No
Mass Upload	Yes	No	Yes
Prototyping	Yes	No	No
Server Page Language	Yes	Yes	Yes
Spell Checker	Yes	No	Free Add On
Style Wizard	Yes		No
Subscriptions	Yes	No	No

Template Language	Yes	Yes	No
UI Levels	Yes	Yes	No
Undo	Yes	Yes	Yes
WYSIWYG Editor	Yes	Yes	Yes
Zip Archives	Yes		No
Performance	**Alfresco**	**Apache Lenya**	**OpenCms**
Advanced Caching	Yes	Yes	Yes
Database Replication	Yes	No	No
Load Balancing	Yes	Yes	No
Page Caching	Yes	Yes	Yes
Static Content Export	Yes	Yes	Yes
Management	**Alfresco**	**Apache Lenya**	**OpenCms**
Advertising Management	No	No	No
Asset Management	Yes	Yes	Yes
Clipboard	Yes	Yes	No
Content Scheduling	Yes	Yes	Yes
Content Staging	Yes	Yes	No
Inline Administration	Yes	Yes	No
Online Administration	Yes	Yes	Yes
Package Deployment	Yes	Yes	Limited
Sub-sites / Roots	Yes	Yes	Yes
Themes / Skins	Yes	No	No
Trash	Yes	Yes	No
Web Statistics	No	Yes	No
Web-based Style/Template Management	Yes	No	Limited
Web-based Translation Management	Yes	No	No
Workflow Engine	Yes	Yes	No
Interoperability	**Alfresco**	**Apache Lenya**	**OpenCms**
Content Syndication (RSS)	Yes	Yes	No
FTP Support	Yes	No	No
UTF-8 Support	Yes	Yes	Yes
WAI Compliant	Yes	No	Limited

WebDAV Support	Yes	Yes	No
XHTML Compliant	Yes	Yes	Yes
Flexibility	**Alfresco**	**Apache Lenya**	**OpenCms**
CGI-mode Support	Yes	No	No
Content Reuse	Yes	Yes	Yes
Extensible User Profiles	Yes	No	No
Interface Localization	Yes	Yes	Yes
Metadata	Yes	Yes	Yes
Multi-lingual Content	Yes	Yes	Yes
Multi-lingual Content Integration	Yes	No	Limited
Multi-Site Deployment	Yes	Yes	Yes
URL Rewriting	Yes	Yes	Yes
Wiki Aware	Yes	No	No
Built-in Applications	**Alfresco**	**Apache Lenya**	**OpenCms**
Blog	No	Yes	No
Chat	No	No	No
Classifieds	No	No	No
Contact Management	No	No	No
Data Entry	No	No	No
Database Reports	No	No	Costs Extra
Discussion / Forum	Yes	No	Free Add On
Document Management	Yes	No	No
Events Calendar	No	No	Costs Extra
Events Management	No		No
Expense Reports	No	No	No
FAQ Management	No	No	Costs Extra
File Distribution	Yes	No	No
Graphs and Charts	No	No	No
Groupware	No	No	No
Guest Book	No	No	No
Help Desk / Bug Reporting	No	No	No
HTTP Proxy	Yes	Yes	No
In/Out Board	No	No	No

Job Postings	No	No	Costs Extra
Link Management	No	No	Yes
Mail Form	No	No	Yes
Matrix	No		No
My Page / Dashboard	Yes	No	No
Newsletter	No	No	No
Photo Gallery	No	No	Yes
Polls	No	No	No
Product Management	No	No	Costs Extra
Project Tracking	No	No	No
Search Engine	Yes	Yes	Yes
Site Map	No	No	Yes
Stock Quotes	No		No
Surveys	No	No	No
Syndicated Content (RSS)	Yes	Yes	No
Tests / Quizzes	No	No	No
Time Tracking	No	No	No
User Contributions	Yes	No	No
Weather	No		No
Web Services Front End	No	No	No
Commerce	**Alfresco**	**Apache Lenya**	**OpenCms**
Affiliate Tracking	No	No	No
Inventory Management	No	No	No
Pluggable Payments	No	No	No
Pluggable Shipping	No	No	No
Pluggable Tax	No	No	No
Point of Sale	No	No	No
Shopping Cart	No	No	No
Subscriptions	No	No	No
Wish Lists	No	No	No

Apéndice F: Manual de usuario.

En este Apéndice presentaremos un pequeño manual de usuario que sirva de referencia al que se enfrenta por primera vez a Lenya.

1.Autenticación

Antes de acceder a la interfaz de usuario es necesaria una autenticación mediante nombre de usuario y contraseña. Ésta debe ser facilitada por el administrador en el caso de que sea una cuenta de usuario local a Lenya. En caso de que su cuenta pertenezca a un servicio de directorio LDAP se utilizará la contraseña de usuario de dicho directorio.

La autenticación LDAP requiere que los usuarios y los grupos a los que pertenecen sean usuarios de Lenya es decir deben ser añadidos explícitamente por el administrador a través de la Interfaz de Administración del gestor.

Es importante comprender esto ya que el hecho de que un usuario exista en LDAP no quiere decir que tenga acceso a Lenya. El usuario ha de ser explícitamente añadido como usuario de Lenya al igual que a los grupos a los que pertenece.

Figura 29: Pantalla de acceso de Apache Lenya

2.Interfaz de usuario

Una vez realizada la autenticación se nos mostrará una interfaz web con un menú como el de la siguiente imagen:

Figura 30: Pestañas de la interfaz de usuario

La interfaz de usuario de Lenya está compuesta por cuatro pestañas:

- Pestaña *Admin*: Este es el área de administración utilizado por los administradores para la gestión de usuarios, grupos, rangos de ip...
- Pestaña *Site*: En esta pestaña podremos ver información sobre los documentos subir imágenes, ver el historial de modificaciones de los documentos y sus distintas versiones así como crear tareas de publicación o desactivación de documentos.
- Pestaña *Authoring*: Esta es la pestaña principal de creación y edición de documentos.
- Pestaña *Live*: Esta pestaña muestra los documentos se encuentran publicados en ese momento.

A continuación se examinará con más detalle cada una de las pestañas y las tareas que se pueden llevar a cabo en cada una de ellas dependiendo de los permisos que posea el usuario. En Lenya existen tres tipos de usuarios: usuarios con permisos de administración, usuarios con permisos de edición y usuarios con permisos de revisión.

- Usuarios con permisos de Administración: estos usuarios pertenecen al grupo *admin* de Lenya o se les ha dado permisos de administrador a través de la interfaz de administración en la pestaña *admin*. Su cometido es la gestión de usuarios y grupos, gestión de los posibles rangos o direcciones IP a los que se les podrá dar permisos especiales y los encargados de vaciar la papelera de reciclaje cuando sea necesario.
- Usuarios con permisos de Edición: estos usuarios pertenecen al grupo editor de Lenya o tienen permisos de edición sobre un subconjunto de los documentos del *site*. Son los encargados de la producción de documentos que a posteriori podrán ser publicados o rechazados por parte de los usuarios con permisos de revisión. Los usuarios editores podrán crear, modificar y enviar los documentos para su aprobación. Es posible dar privilegios de edición solamente sobre un conjunto de páginas a través de la interfaz de control de accesos de edición en la pestaña *site*. De esta manera cada usuario solamente tendrá asignada la edición de determinados documentos a diferencia de los usuarios que pertenecen al grupo editor que tienen permisos de edición sobre todos los documentos del *site*.
- Usuarios con permisos de Revisión: estos usuarios pertenecen al grupo revisor de Lenya y se encargarán de la supervisión de los contenidos producidos por los editores. Los usuarios revisores podrán aceptar o rechazar dichos contenidos lo se supone la publicación final de los documentos, o el rechazo de éstos para su posterior corrección por parte de los editores. Al igual que se explicó anteriormente, es posible la asignación de permisos de revisión sobre un conjunto de documentos individuales, permitiendo así la división de las responsabilidades de revisión entre distintos usuarios de una forma no solapada.

Estos permisos se asignan agregando a los usuarios a los distintos grupos por defecto, lo que les daría permisos sobre todos los documentos del *site*, o bien asignando permisos a usuarios o grupos de usuarios sobre un conjunto de documentos, lo que les daría permisos únicamente sobre dicho subconjunto. En el caso de utilizar permisos restringidos a un subconjunto de documentos es recomendable dar permisos de visita sobre el documento inicial, ya que en caso contrario la autenticación de los usuarios, aunque satisfactoria, NO daría acceso a la página inicial de edición en la pestaña *authoring*, lo que dificultaría la edición de los documentos por parte de los usuarios ya que estos tendrían que acceder directamente al gestor con la URL de los documentos sobre los que tienen permisos, situación que resultaría, cuando menos, incómoda. Así mismo y por la misma razón, si un usuario tiene asignados permisos sobre un documento dentro del árbol de documentos del *site*, es recomendable asignarle permisos de visita sobre todos los documentos que se encuentren por encima de éste, el objetivo es que el usuario pueda navegar a través del gestor hasta el documento sobre el que tiene permisos de edición o revisión.

3.Pestaña *Authoring*

Ésta es la pestaña principal de trabajo y es donde se podrán ver los documentos que se encuentran en desarrollo. Aquí los usuarios con permisos de edición podrán editar, modificar o crear nuevos documentos y aprobarlos para su posterior revisión por parte de los usuarios con permisos de revisión.

Los usuarios con permisos de revisión podrán examinar los documentos y decidir si estos se publican o si deben ser rechazados para su posterior corrección. También podrán desactivar documentos publicados y programar tareas automáticas que realicen las tareas de publicación y desactivación de páginas.

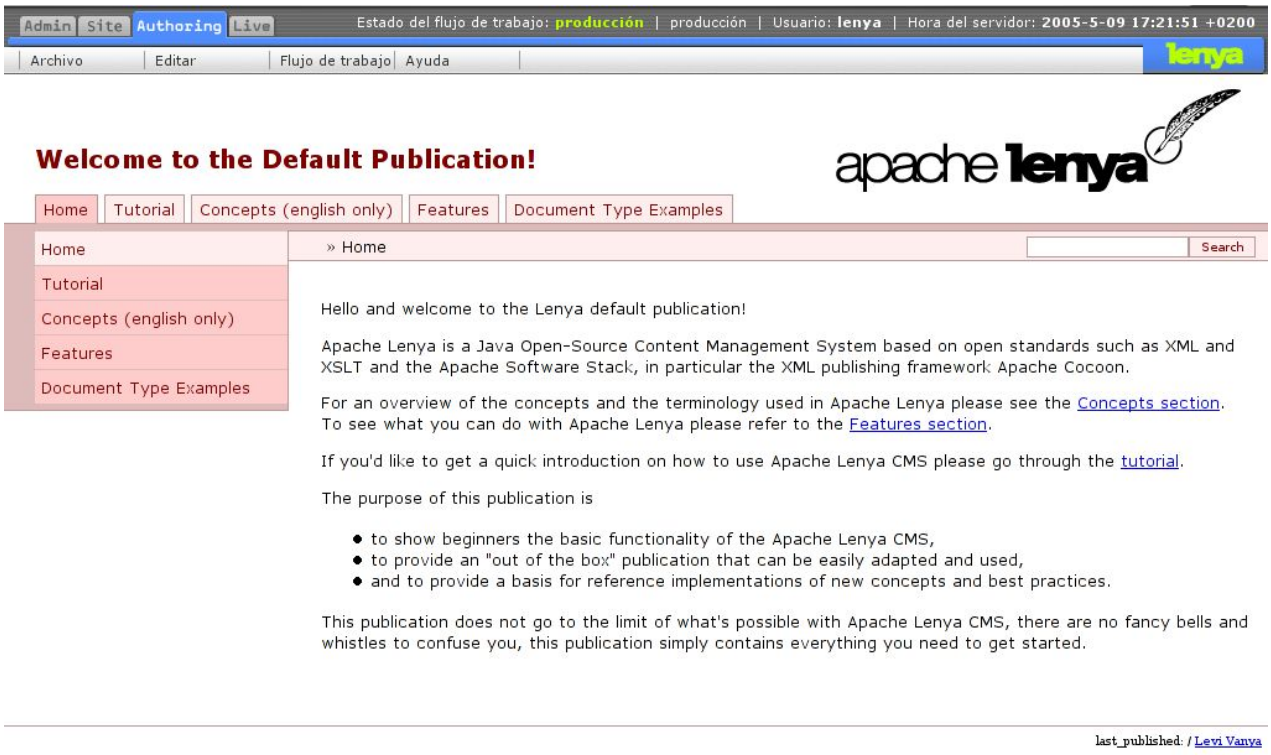

Figura 31: Pestaña Authoring

3.1.Creación, borrado y modificado de los documentos

Los editores pueden crear nuevos documentos a partir del menú archivo. En este menú se muestran los posibles tipos de documentos que es posible crear. Cada tipo de documento tiene una plantilla asociada para facilitar el trabajo del editor y darle una idea de la estructura del documento. En este menú también se podrán crear las versiones de los documentos que ya existen, en los distintos idiomas disponibles.

La forma de crear un documento es simple, el usuario, en primer lugar, hará click sobre el link bajo el cual se creará el nuevo documento. Si el usuario desea crear el documento en el nivel más alto deberá hacer click sobre el link de la página de inicio. A continuación hará click sobre el menú archivo donde se desplegarán los tipos de documentos que le será posible crear dentro de esa sección.

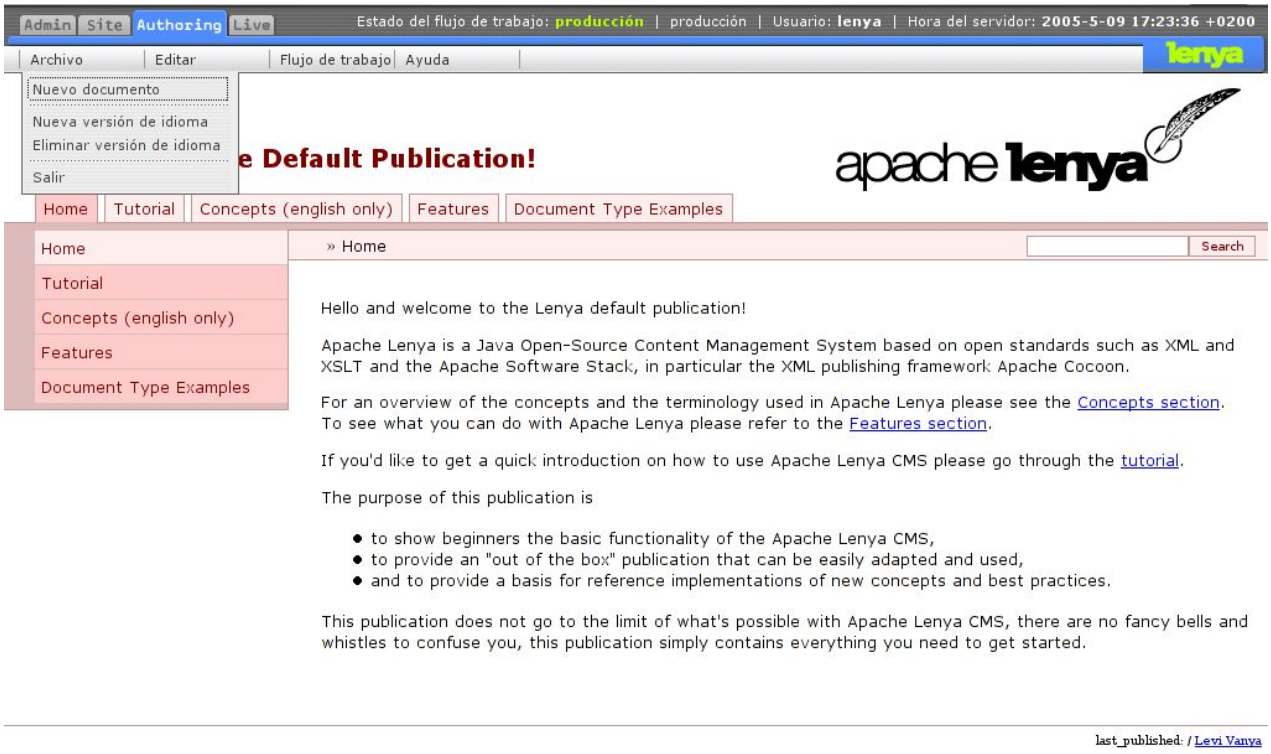

Figura 32: Menú Archivo

Si es posible crear nuevas versiones del documento actual en otros idiomas también le aparecerán los links Nueva versión de idioma activos.

3.2.Subida de imágenes y ficheros

En la edición de documentos es habitual la inclusión de imágenes o enlaces a otros tipos de documento (pdf, doc..). En Apache Lenya es posible incluir imágenes y otros tipos de documentos en el momento de la edición. Sin embargo las imágenes o los documentos han de estar previamente en la librería del Editor KUPU antes de poder ser utilizados. Este tipo de recursos Lenya los denomina 'Activos'. Para subir imágenes o documentos se hará click en la pestaña Site y a continuación se hará click en la pestaña Activos tal y como se muestra en la imagen:

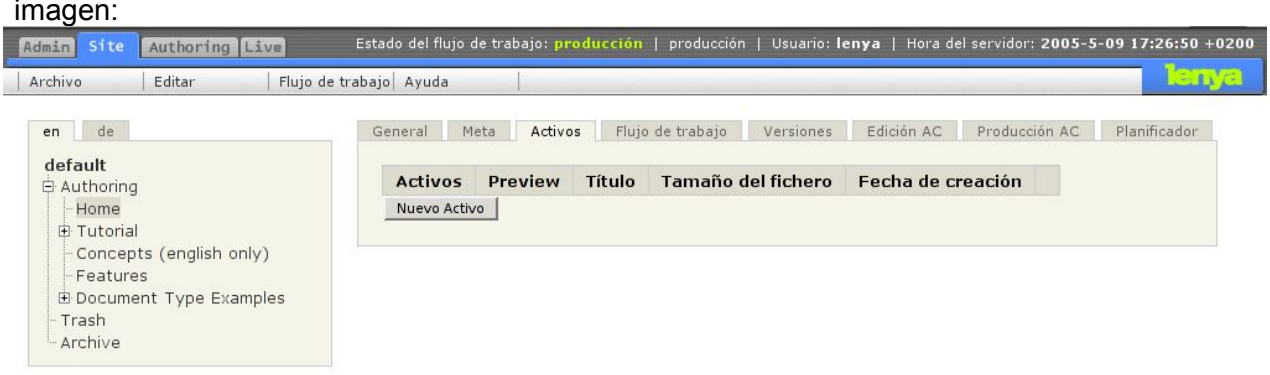

Figura 33: Subida de imágenes 1

A continuación pulsaremos el botón Nuevo Activo después de lo cual se nos abrirá una ventana del navegador desde la cual seleccionaremos la imagen deseada y pulsaremos aceptar:

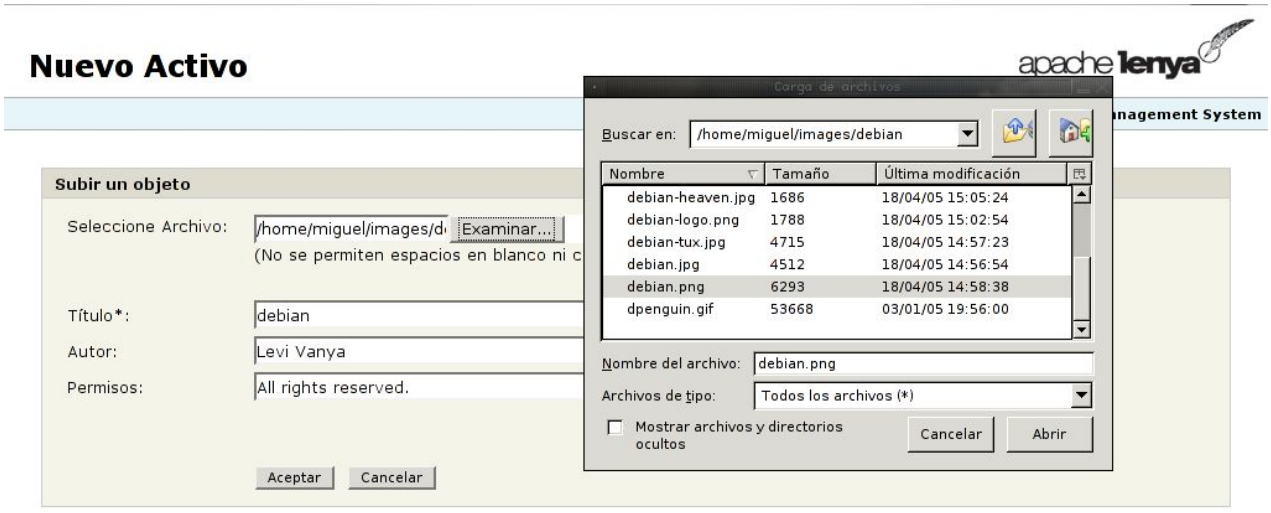

Figura 34: Subida de imágenes 2

A partir de este momento la imagen o el documento estará disponible para su inclusión desde el editor KUPU. Podremos ver todos los Activos disponibles en la pestaña Activos dentro de la pestaña Site:

Figura 35: Subida de imágenes 3

3.3.Edición de documentos con Kupu

Para la edición de documentos volveremos a la pestaña Authoring y haremos click sobre el documento que se desea editar. A continuación haremos click en el menú Editar y después en el elemento de menú Editar con KUPU.

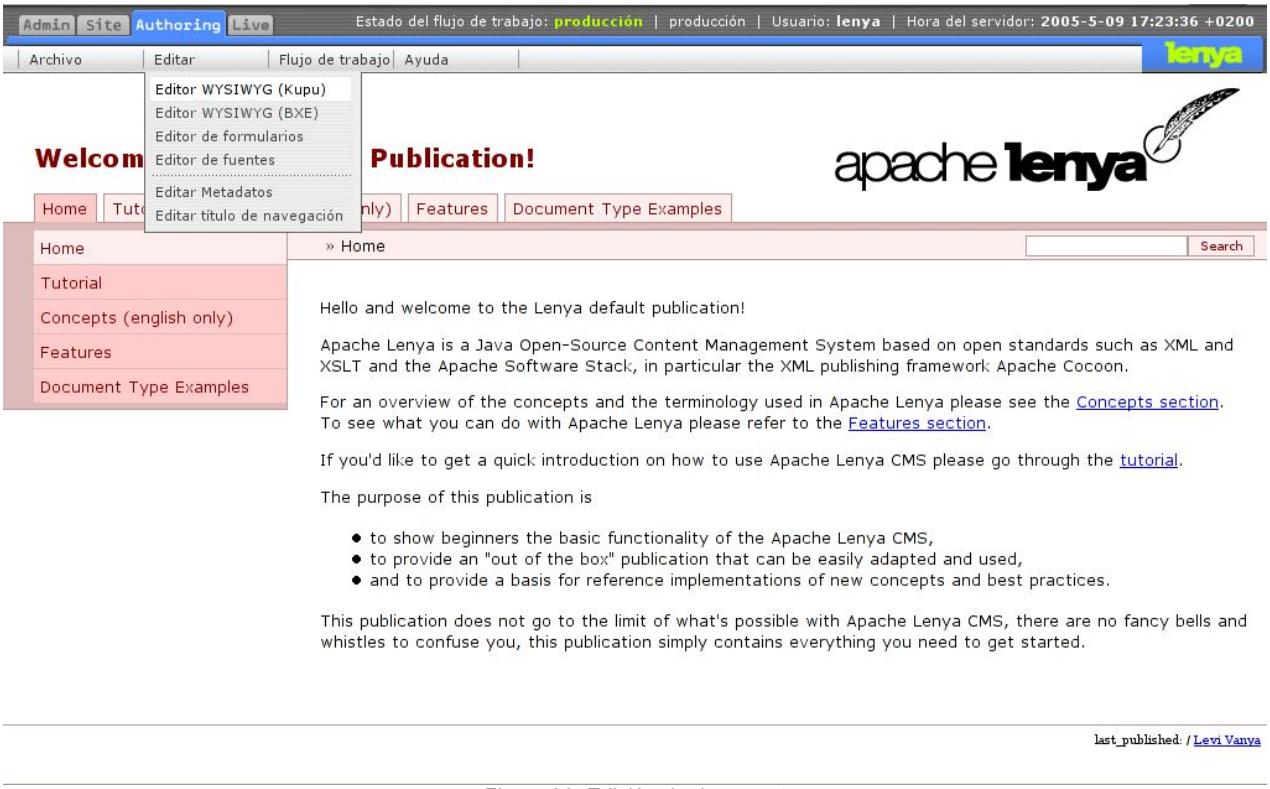

Figura 36: Edición de documentos

En este instante se nos mostrará la pantalla del editor tal y como aparece en la siguiente figura.

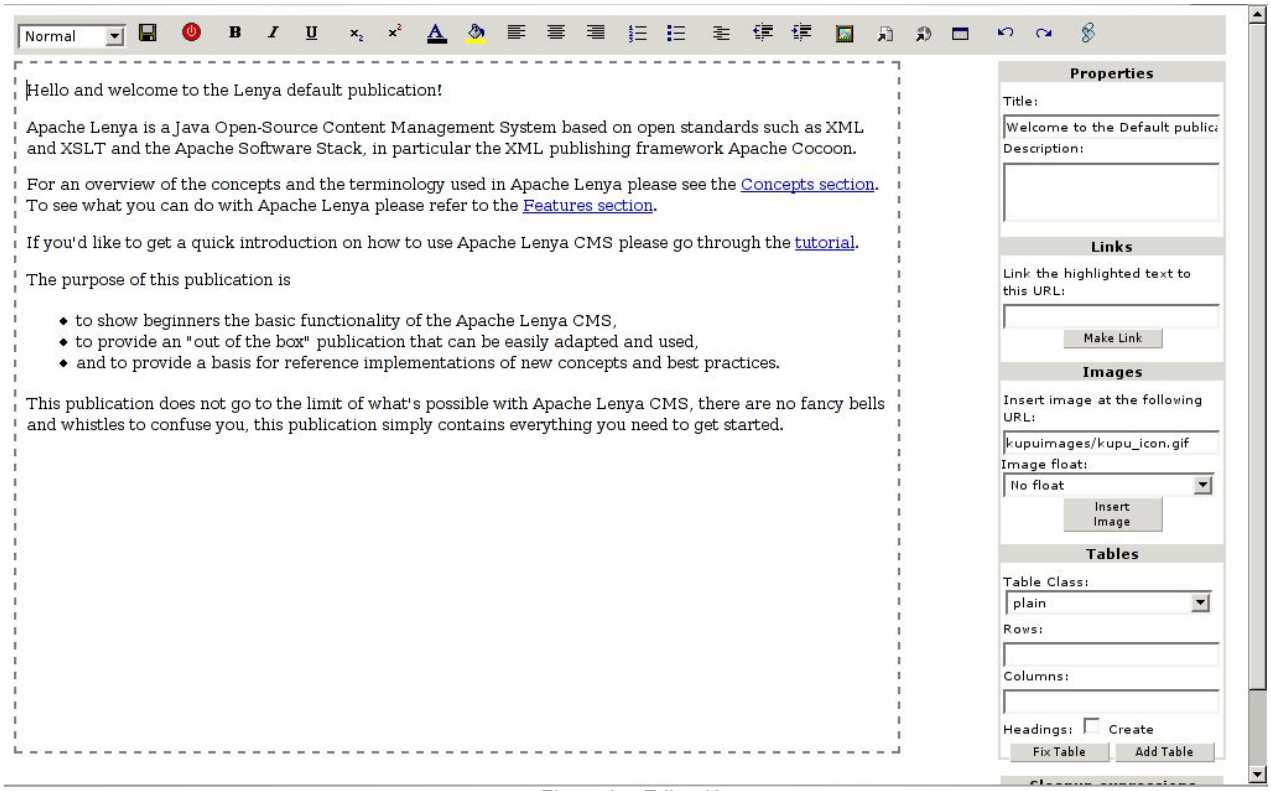

Figura 37: Editor Kupu

A partir de este momento podemos empezar a editar nuestro documento dándole formato con las opciones que nos brinda el editor KUPU. Podremos dar formato al texto incluir enlaces externos o internos, que podremos utilizar para hacer referencia a otras páginas o a documentos dentro de nuestro site, respectivamente, e incluir las imágenes disponibles. Recuerde que tanto las imágenes como los ficheros internos habrán de subirse previamente desde la pestaña site (Ver el apartado anterior).

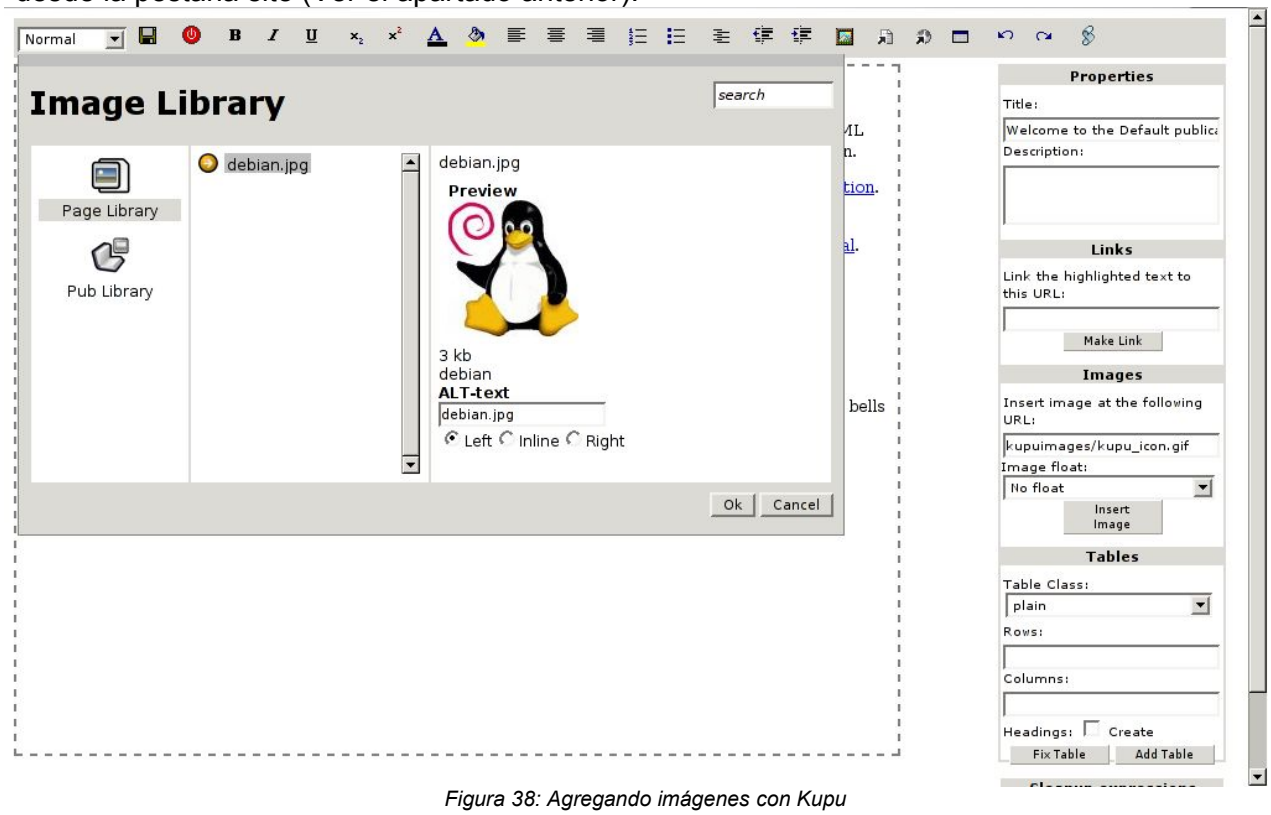

Figura 38: Agregando imágenes con Kupu

En el caso de que el fichero subido no sea una imagen su inclusión se realizará mediante un link al objeto en cuestión. Por ejemplo, si subimos un fichero llamado fichero.pdf a la página de inicio el enlace se hará como se muestra en la siguiente figura:

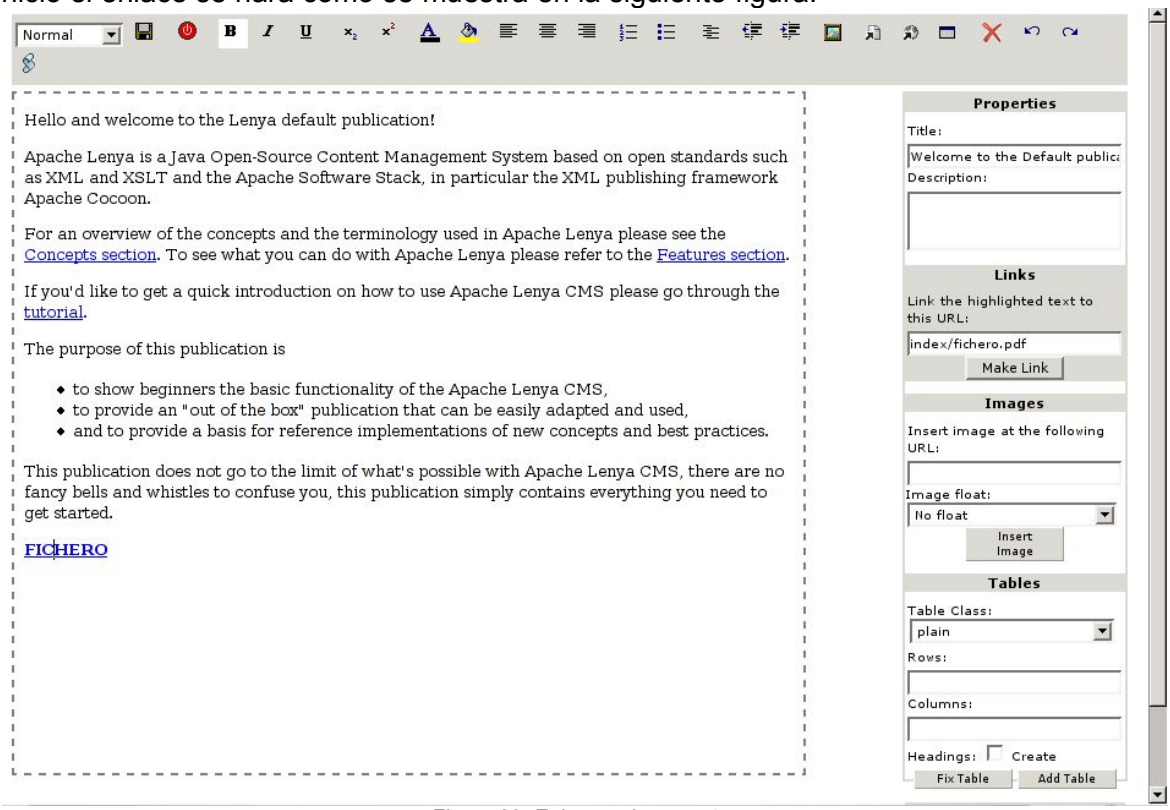

Figura 39: Enlace a documentos

Los pasos a seguir son:

- Escribir un texto significativo sobre el que se enlazará el documento.
- Seleccionar dicho texto.
- Escribir la ruta hacia el documento en el formulario llamado *links* de la parte derecha del editor y se hará *click* sobre el botón *Make Link*.

La ruta se escribirá dependiendo del documento sobre el que hemos subido el fichero en el árbol de documentos de la publicación y tendrá la siguiente estructura:

```
{nombre_del_documento_en_la_publicacion/nombre_de_fichero}
```

En el ejemplo el fichero *fichero.pdf* se subió a la página de inicio: *index.html*. Esto determina que el *link* hacia el fichero es *index/fichero.pdf*. Si en lugar de subirlo a *index.html* lo hubiéramos subido a *tutorial/new_doctype.html* la ruta hacia el documento sería *tutorial/new_doctype/fichero.pdf*.

3.4.Estapas de publicación (flujo de trabajo)

El flujo de trabajo consta de las etapas por las que pasa un documento desde que es creado hasta que es archivado o borrado. El flujo de trabajo por defecto en Apache Lenya es el siguiente:

- Los usuarios con permisos de edición crean un nuevo documento o modifican uno existente.
- Una vez terminada la edición del nuevo documento o la modificación el editor acepta el documento para su posterior revisión por parte de los usuarios con permisos de revisión.

- Los usuarios con permisos de revisión comprobarán los documentos y publicarán o rechazarán los contenidos. Ésto último provocaría la vuelta al punto uno del flujo de trabajo.
- Los usuarios con permisos de revisión podrán desactivar documentos que han superado su tiempo máximo de publicación así como proceder a su archivación o borrado.

Todas las tareas que se pueden llevar a cabo en el flujo de trabajo se realizan a través del menú “Flujo de Trabajo”.

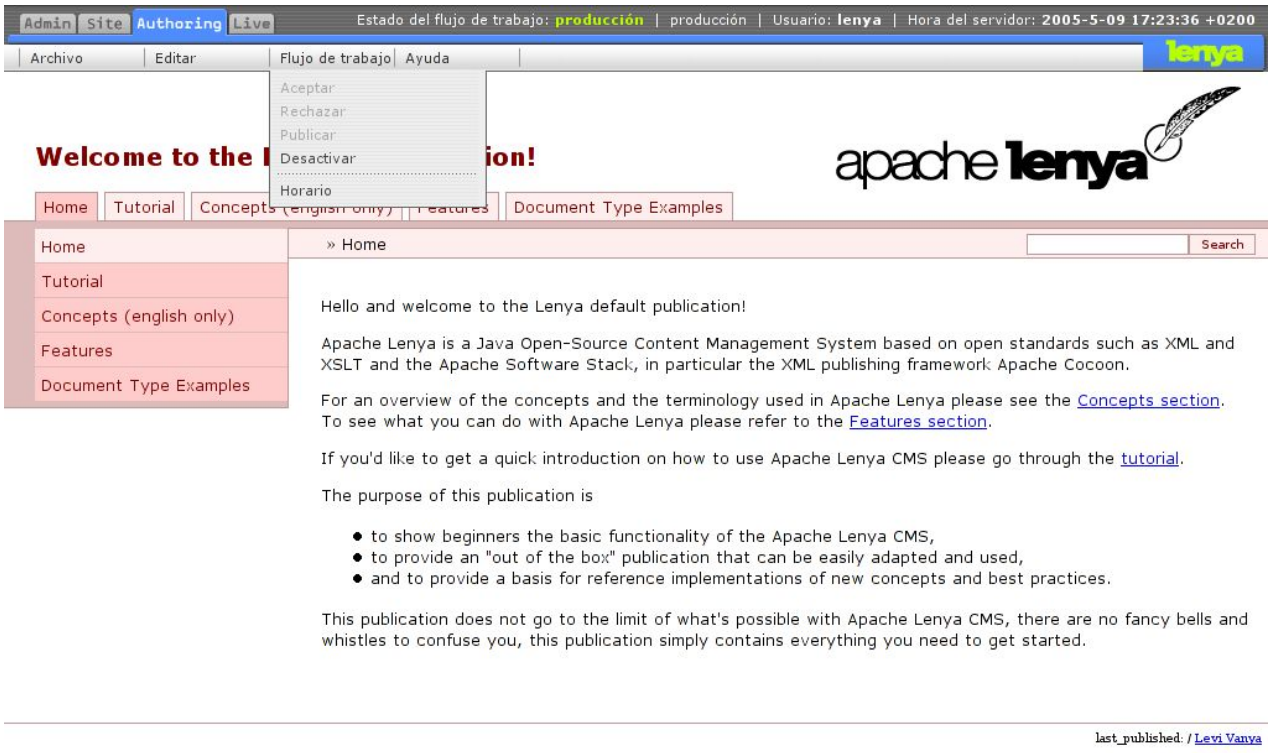

Figura 40: Menú del Flujo de Trabajo

Desde este menú los usuarios con permisos de revisión podrán realizar la programación de las tareas de publicación o desactivación de documentos. Para esto el documento habrá de ser aceptado previamente por los usuarios con permisos de edición. Para programar una tarea haremos click en el menú 'Flujo de Trabajo' y a continuación en 'Horario'. Se nos mostrará una pantalla en la que podremos programar la hora de publicación o desactivación del documento. No es obligatorio programar las dos ya que son totalmente independientes.

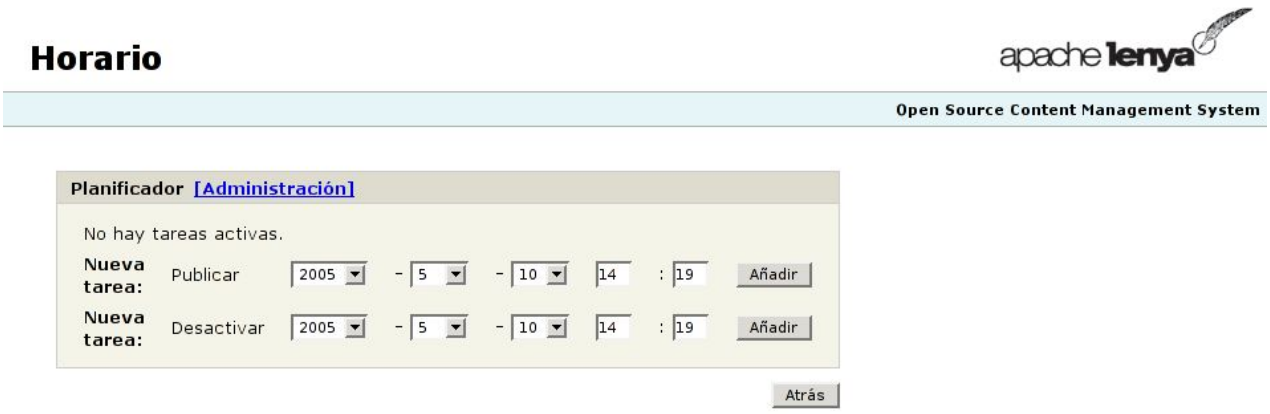

Figura 41: El Planificador

4.Pestaña *Site*

En esta pestaña se puede ver el mapa del sitio e información acerca del documento seleccionado.

Figura 42: Pestaña Site

Dentro de cada una de las pestañas que nos aparecen en el cuadro central se nos mostrará información sobre el documento o se nos permitirá realizar tareas relacionadas con la edición tales como la subida de imágenes o modificación de tareas en el planificador. A continuación veremos cada una de las pestañas por separado.

5.Pestaña Meta

Aquí podremos modificar la metainformación del documento, esto es, la información sobre el propio documento, tal como su autor, el tema sobre el que trata el documento, etc...

Figura 43: Pestaña Meta

5.1.Pestaña Activos

Esta pestaña nos permitirá subir otros ficheros tales como imágenes, pdf, doc, etc.

Figura 44: Pestaña Activos

5.2.Pestaña de Flujo de trabajo

Aquí se podrán ver todos los estados por los que ha pasado el documento y el estado del flujo de trabajo en el que se encuentra actualmente.

Figura 45: Pestaña del flujo de trabajo

5.3.Pestaña de versiones

Aquí podremos ver todas las versiones del documento a lo largo de su ciclo de vida, así como deshacer los cambios hasta una versión anterior.

Figura 46: Pestaña versiones

5.4.Pestaña Edición AC (Access Control) y Producción AC

En estas pestañas se podrán establecer permisos a usuarios o grupos sobre subconjuntos de documentos en edición o producción respectivamente.

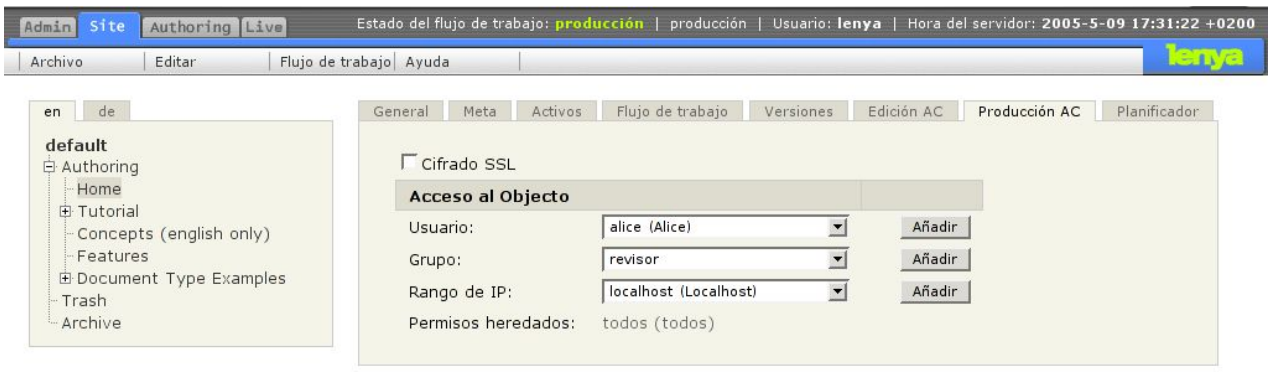

Figura 48: Pestaña Producción AC

5.5.Pestaña del Planificador

Desde esta pestaña podremos gestionar las tareas de publicación o desactivación de documentos. En esta pestaña se nos mostrarán tanto las tareas activas como las ya realizadas y nos permitirá modificar o borrar las tareas pendientes que previamente se programaron en el elmento de menú flujo de trabajo - horario.

Figura 49: El Planificador

6.Pestaña Administración

A esta pestaña sólo tendrán acceso los usuarios con permisos de administración y será desde aquí desde donde podrán gestionar los usuarios y grupos pertenecientes a lenya.

Figura 50: Pestaña de administración

6.1.Administración de usuarios

En este apartado se pueden añadir o borrar usuarios así como ver la lista de los usuarios existentes y los grupos a los que pertenecen.

Figura 51: Gestión de Usuarios

Lenya permite añadir tanto usuarios locales como usuarios pertenecientes a un servicio de directorio LDAP. Para ello, se puede consultar [LENYA_LDAP].

Figura 52: Creando un nuevo usuario 1

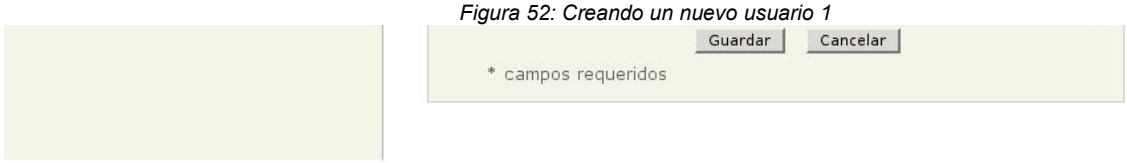

Figura 53: Creando un nuevo usuario 2

6.2.Administración de Grupos

Desde aquí se podrán crear, borrar o modificar los grupos de usuarios que pertenecerán al gestor de contenido.

Figura 54: Gestión de Grupos

Figura 55: Creando un nuevo Grupo

6.3.Administración de Rangos IP

Desde esta sección podremos gestionar los rangos de direcciones IP a los que podremos asignar opcionalmente un grupo de usuarios con sus correspondientes privilegios.

Figura 56: Administración de Rangos de IP 1

Figura 57: Administración de Rangos de IP 2

Figura 58: Administración de Rangos de IP 3

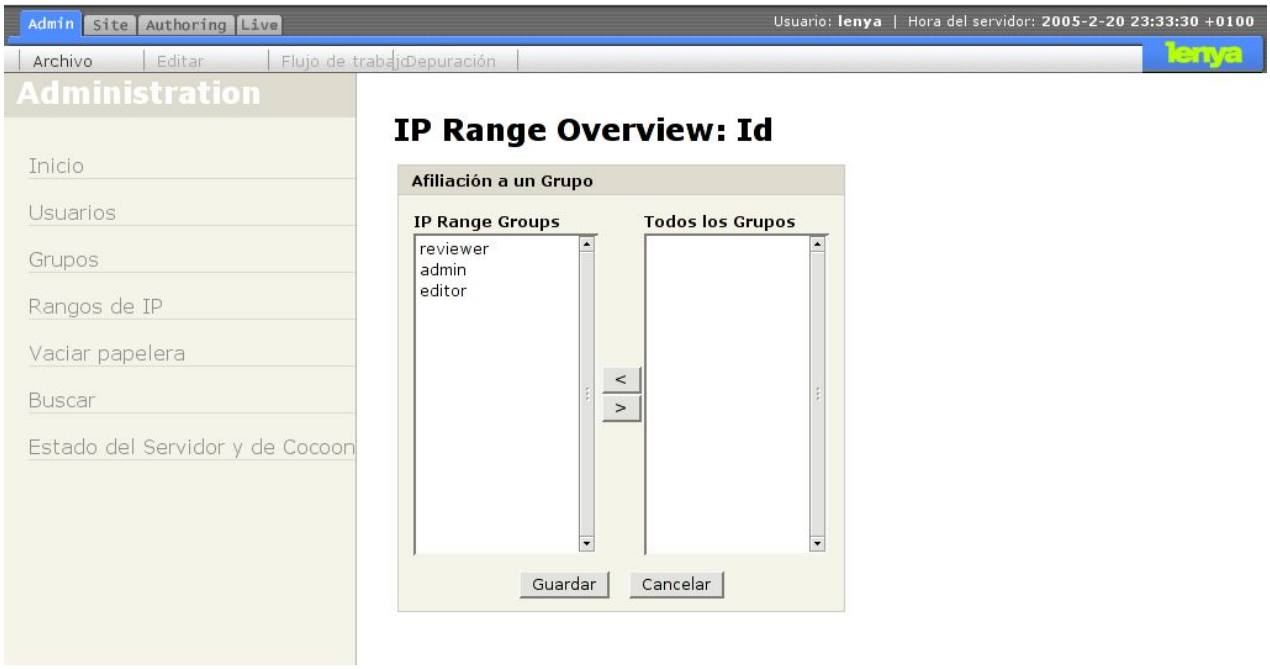

Figura 59: Administración de Rangos de IP 4

6.4.Estado del servidor

Este apartado nos mostrará información del estado del gestor de contenido (máquina virtual que utiliza, estado de la memoria, sistema operativo ....).

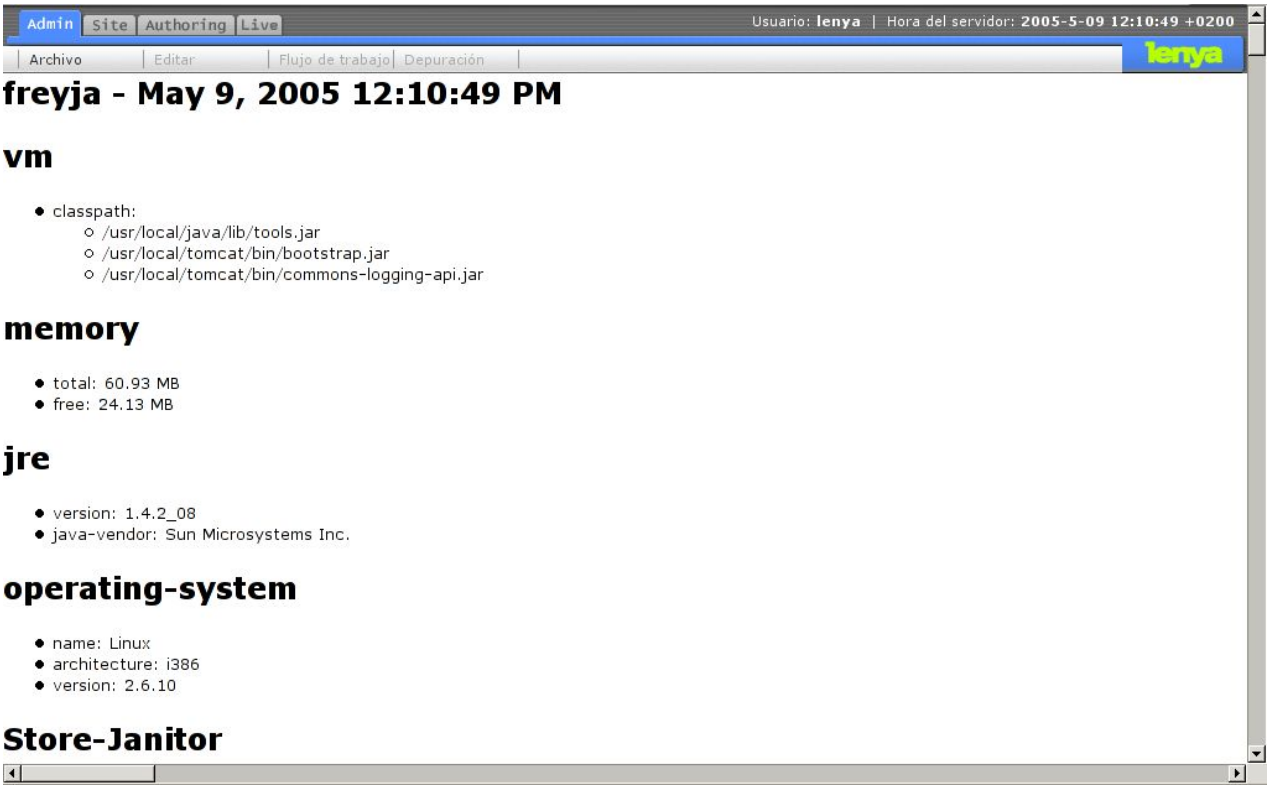

Figura 60: Estado del Servidor

7.Pestaña *Live*

Al hacer click en esta pestaña se abrirá una nueva ventana del navegador donde se nos mostrarán los documentos publicados actualmente en la web.