Máster Oficial en Software Libre

Proyecto de Final de Máster

Diseño e implementación de un editor LaTeX colaborativo

«CoLaTeX»

M. Colom

18 de junio de 2009

It's so easy love cos you got friends you can trust[1].

[1] De la canción *Friends Will Be Friends* del álbum *A Kind of Magic* de Queen, escrita por Freddie Mercury y John Deacon.

Índice

1. Licencias

Este documento está publicado bajo licencia Creative Commons, en concreto con la modalidad *Reconocimiento - CompartirIgual (by-sa)*.

Esto significa que en cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría. La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas. Se permite el uso comercial de la obra y de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

Es posible consultar en la web de Creative Commons tanto el resumen de esta licencia [13], como el texto legal completo [14].

En cuanto a los programas, tanto el núcleo de CoLaTeX como la aplicación web están bajo licencia GPLv3 [11].

2. Introducción

Este documento es la memoria de CoLaTeX, el proyecto final del Màster Oficial en Programari Lliure de la Universitat Oberta de Catalunya (UOC), en la modalidad de *Administración de Redes y de Sistemas Operativos en Entornos de Software Libre*.

Este PFM[2] está enmarcado dentro del proyecto *UniWiki*, fruto de la colaboración entre el INRIA[3]-Nancy (Francia) y la Universitat Rovira i Virgili.

El objetivo final es diseñar, desrrollar y implementar una aplicación distribuida y descentralizada con la que múltiples usuarios puedan colaborar en la creación y edición de un proyecto LaTeX.

Tradicionalmente, los sistemas centralizados de edición colaborativa de documentos, como las herramientas de *groupware* para LaTeX y otros formatos (también las *wikis*, como Wikipedia, y otros), han utilizado el paradigma de comunicación cliente-servidor. En este esquema, un único servidor[4] es el que almacena de forma centralizada toda la información relativa al contenido del proyecto LaTeX, y al que los clientes se conectan para consultar y modificar el contenido de forma concurrente.

Esta centralización, aunque facilita la gestión de la información al localizarla en un único punto conocido, tiene la desventaja de que propicia la aparición de múltiples problemas como, por ejemplo:

- Cuellos de botella o superutilización de recursos en algunas partes del sistema, como las conexiones de red con el servidor central o su base de datos.
- Riesgo de perder información en el caso de fallos en el hardware de almacenamiento. Incluso en el caso de que se utilice un clúster con sistemas redundantes, el sistema sigue siendo vulnerable, ya que todo el hardware comparte una misma localización (el centro de cálculo de una universidad, por ejemplo).
- Al estar centralizado, es muy fácil ejercer la censura contra el sistema, simplemente bloqueando el punto de entrada y salida de información. Por ejemplo, filtrando la dirección IP de servidor HTTP[5].

[2] Proyecto de Final de Máster.

[3] Institut national de recherche en informatique et en automatique.

[4] No tiene por qué estar implementado en una única máquina, sino que puede ser un clúster o varios de ellos interconectados, como es el caso de Wikipedia, por ejemplo.

[5] *HyperText Transfer Protocol*.

Es posible citar ejemplos reales en este sentido, como la decisión (luego rectificada) de un juez español de bloquear el acceso completo al servidor de *blogs* Wordpress[6] a un determinado ISP.

- Su disponibilidad puede verse fácilmente comprometida en caso de fallos en los equipos de red o hardware en general.

Una forma de resolver estos problema es precisamente evitar que la información esté centralizada en un único punto, y distribuirla en varios nodos de la red. Aunque esto hace que aparezcan nuevos problemas que se han de resolver:

- Cómo localizar un dato concreto (por ejemplo, una determinada imagen a partir de su nombre de fichero).
- Cómo guardar un dato en el sistema, de forma concurrente y asegurando la consistencia de la información.
- Cómo asegurar el buen funcionamiento del sistema en un entorno en el que los nodos se conectan y desconectan de forma imprevisible.
- Cómo hacer que el sistema no deje de funcionar aún cuando algunos participantes funcionen incorrectamente (errores en el software de la aplicación o el propio sistema operativo de la máquina en la que se ejecuta, por ejemplo).
- Qué operaciones debe llevar a cabo un nodo al recuperarse de un error y reincorporarse de nuevo a la red.

Concretamente, en este PFM se diseñará una aplicación distribuida de edición colaborativa de documentos en formato LaTeX que permitirá que cada usuario pueda editar *online* una serie de ficheros que componen un proyecto LaTeX, tanto los ficheros de texto (.tex, .bib, ...) como los binarios (.eps, .pdf, .png, ...). Su nombre es *CoLaTeX* (***Co****llaborative* ***LaTeX***)[7].

La arquitectura del sistema se explica detalladamente en la sección 5.1.

[6] Noticia en bandaancha.eu: *Jazztel reactiva el acceso a Wordpress tras una rectificación del juez* - http://bandaancha.eu/articulo/6412/jazztel-reactiva-acceso-wordpress-tras-rectificacion-juez

[7] LaTeX es asimismo la contracción de *Lamport TeX*, siendo TeX el sistema de tipografía creado por Donald Knuth en 1978.

3. Manual de usuario

En esta sección se explica cómo compilar, testear y utilizar la aplicación web de edición colaborativa, dirigida al usuario final.

En otras secciones de este informe es posible encontrar los detalles técnicos sobre la arquitectura y funcionamiento del núcleo y la aplicación web.

3.1. Compilación del núcleo

Para compilar el núcleo, en principio basta con ejecutar la orden *ant jar* en el directorio del proyecto *colatex*.

Se debería obtener una salida similar a la siguiente:

```
miguel@nenux:~/ENTREGAR/colatex$ ls
build.xml           easypastry-config.xml  manifest.mf  src   xml-resources
bunshin.properties  lib                    nbproject    test

miguel@nenux:~/ENTREGAR/colatex$ ant jar
Buildfile: build.xml

-pre-init:
-init-private:
-init-user:
-init-project:
-init-macrodef-property:
-do-init:
-post-init:
-init-check:
-init-macrodef-javac:
-init-macrodef-junit:
-init-debug-args:
-init-macrodef-nbjpda:
-init-macrodef-debug:
-init-macrodef-java:
```

```
-init-presetdef-jar:

init:
deps-jar:
-check-automatic-build:
-clean-after-automatic-build:
-verify-automatic-build:

-pre-pre-compile:
    [mkdir] Created dir: /home/miguel/ENTREGAR/colatex/build/classes

-pre-compile:
-compile-depend:

-do-compile:
    [javac] Compiling 13 source files to /home/miguel/ENTREGAR/colatex/build/classes
    [javac] Note: Some input files use unchecked or unsafe operations.
    [javac] Note: Recompile with -Xlint:unchecked for details.
     [copy] Copying 1 file to /home/miguel/ENTREGAR/colatex/build/classes

-post-compile:

compile:
-pre-jar:
-pre-pre-jar:
    [mkdir] Created dir: /home/miguel/ENTREGAR/colatex/dist
-do-jar-with-manifest:
-do-jar-without-manifest:
-do-jar-with-mainclass:
-do-jar-with-libraries:
 [copylibs] Building jar: /home/miguel/ENTREGAR/colatex/dist/colatex.jar
 [copylibs] Copy libraries to /home/miguel/ENTREGAR/colatex/dist/lib.
     [echo] To run this application from the command line without Ant, try:
     [echo] java -jar "/home/miguel/ENTREGAR/colatex/dist/colatex.jar"
```

```
-post-jar:
axis2-aar:
    [mkdir] Created dir: /home/miguel/ENTREGAR/colatex/build/axis2/WEB-INF/services
    [mkdir] Created dir: /home/miguel/ENTREGAR/colatex/xml-resources/axis2/lib
     [copy] Copying 1 file to /home/miguel/ENTREGAR/colatex/xml-resources/axis2/lib
      [jar] Building jar: /home/miguel/ENTREGAR/colatex/build/
                          axis2/WEB-INF/services/colatex.aar
   [delete] Deleting directory /home/miguel/ENTREGAR/colatex/xml-resources/axis2/lib

jar:

BUILD SUCCESSFUL
Total time: 2 seconds
```

Si la compilación finaliza correctamente, se habrán generado:

- El fichero JAR con la aplicació núcleo de CoLaTeX, en *dist/colatex.jar*.
- El directorio «lib» conteniendo las librerías necesarias para el funcionamiento de CoLaTeX (EasyPastry, WOOT, etc), en dist/lib.
- El fichero colatex.aar conteniendo los dos web services que opcionalmente se pueden desplegar en el servidor web. Se genera en *build/axis2/WEB-INF/services/colatex.aar*. En la sección 3.3.1 se explica en detalle cómo hacerlo.

También es posible generar toda la documentación *javadoc* mediante la orden *ant javadoc*. Se guardará en *dist/javadoc*.

Una vez compilado correctamente el núcleo, para ejecutarlo basta con ejecutar la orden *java -jar dist/colatex.jar* en el directorio *dist*.

El fichero *build.xml* que utiliza ant se generó automáticamente mediante el IDE NetBeans, y es posible que la compilación falle, dependiendo de la configuración del sistema en que se ejecute.

Si se diera esta situación, la mejor solución es abrir el proyecto con NetBeans y realizar la compilación desde el IDE con la orden *Clean and Build Project*. En la figura (1) se muestra cómo hacerlo.

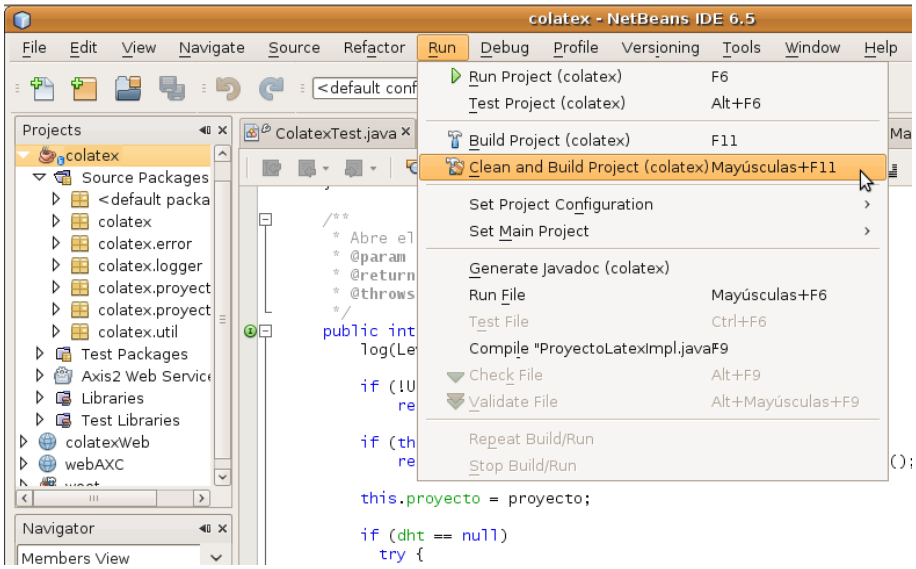

Figura 1: compilar el núcleo de CoLaTeX desde el IDE NetBeans.

3.2. Compilación de la aplicación web

Para compilar la aplicación web, hay que ejecutar la orden *ant* en el directorio del proyecto *colatexWeb*.

Se debería obtener una salida similar a la siguiente:

```
miguel@nenux:~/ENTREGAR/colatexWeb$ ls
build.xml  nbproject  setup  src  test  web

miguel@nenux:~/ENTREGAR/colatexWeb$ ant
Buildfile: build.xml

-pre-init:
-init-private:
-init-user:
-init-project:
-init-macrodef-property:
-do-init:
-post-init:
-init-check:
```

```
-init-macrodef-javac:
-init-macrodef-junit:
-init-macrodef-java:
-init-debug-args:
-init-macrodef-nbjpda:
-init-macrodef-nbjsdebug:
-init-macrodef-debug:
-init-taskdefs:

init:
deps-module-jar:
deps-ear-jar:
deps-jar:

-pre-pre-compile:
    [mkdir] Created dir: /home/miguel/ENTREGAR/colatexWeb/build/web/WEB-INF/classes

-pre-compile:

-copy-manifest:
    [mkdir] Created dir: /home/miguel/ENTREGAR/colatexWeb/build/web/META-INF
     [copy] Copying 1 file to /home/miguel/ENTREGAR/colatexWeb/build/web/META-INF

-copy-persistence-xml:

-copy-webdir:
     [copy] Copying 40 files to /home/miguel/ENTREGAR/colatexWeb/build/web

library-inclusion-in-archive:
library-inclusion-in-manifest:
-do-ws-compile:
-do-compile:
    [javac] Compiling 13 source files to
            /home/miguel/ENTREGAR/colatexWeb/build/web/WEB-INF/classes
```

```
     [copy] Copied 4 empty directories to 4 empty directories under
            /home/miguel/ENTREGAR/colatexWeb/build/web/WEB-INF/classes

-post-compile:

compile:
-pre-dist:
compile-jsps:

-do-dist-with-manifest:
    [mkdir] Created dir: /home/miguel/ENTREGAR/colatexWeb/dist
      [jar] Building jar: /home/miguel/ENTREGAR/colatexWeb/dist/colatexWeb.war

-do-dist-without-manifest:
do-dist:
-post-dist:

dist:

javadoc-build:
    [mkdir] Created dir: /home/miguel/ENTREGAR/colatexWeb/dist/javadoc
  [javadoc] Warning: Leaving out empty argument '-windowtitle'
  [javadoc] Generating Javadoc
  [javadoc] Javadoc execution
  [javadoc] Loading source file (...)/error/ColatexError.java...
  [javadoc] Loading source file (...)/executor/Executor.java...
  [javadoc] Loading source file (...)/executor/ProyectoLatexExecutor.java...
  [javadoc] Loading source file (...)/executor/ProyectosExecutor.java...
  [javadoc] Loading source file (...)/proyectoLatex/InfoDirectorio.java...
  [javadoc] Loading source file (...)/proyectoLatex/InfoFichero.java...
  [javadoc] Loading source file (...)/proyectoLatex/ListaDirectoriosYFicheros.java...
  [javadoc] Loading source file (...)/proyectoLatex/ProyectoLatex.java...
  [javadoc] Loading source file (...)/proyectos/InfoProyecto.java...
  [javadoc] Loading source file (...)/proyectos/Proyectos.java...
```

```
  [javadoc] Loading source file (...)/servlet/proyectoLatex.java...
  [javadoc] Loading source file (...)/servlet/proyectos.java...
  [javadoc] Loading source file (...)/util/Util.java...
  [javadoc] Constructing Javadoc information...
  [javadoc] Standard Doclet version 1.6.0_0
  [javadoc] Building tree for all the packages and classes...
  [javadoc] Building index for all the packages and classes...
  [javadoc] Building index for all classes...

javadoc-browse:
javadoc:
default:

BUILD SUCCESSFUL
Total time: 4 seconds
```

Si la compilación finaliza correctamente, se habrán generado:

- El fichero War con la aplicació web, en *dist/colatexWeb.war*.
- La documentación javadoc, en */dist/javadoc/*.

Una vez obtenido el fichero *colatexWeb.war*, basta con desplegarlo en el servidor elegido. Por ejemplo, en Jetty basta con copiarlo en la carpeta *webapps* y reiniciar el servidor.

El fichero *build.xml* que utiliza ant se generó automáticamente mediante el IDE NetBeans, y es posible que la compilación falle, dependiendo de la configuración del sistema en que se ejecute.

Si se diera esta situación, la mejor solución es abrir el proyecto con NetBeans y realizar la compilación desde el IDE con la orden *Clean and Build Project*, del mismo modo que se explicó para la compilación del núcleo desde el IDE NetBeans.

3.3. Inicio de la aplicación web

El siguiente paso consiste en iniciar la aplicación web. Suponiendo que previamente ha sido desplegada en el servidor web, simplemente basta con visitar la URL de la aplicación.

En el caso de que al ejecutar operaciones mediante la interficie web se obtengan errores relativos a RMI, posiblemente sea un problema de configuración de permisos en la máquina en la que corre el núcleo.

La localización del fichero *java.policy* que define la política de permisos de Java es diferente según la distribución o el sistema operativo utilizados. En Ubuntu GNU/Linux, por ejemplo, se guarda en */usr/lib/jvm/java-6-openjdk/jre/lib/security/java.policy*.

Una forma rápida de comprobar si el problema que se presenta está causa por los permisos para RMI, es añadir la directiva «AllPermission» en el fichero de política de seguridad y verificar posteriormente que las operaciones se ejecutan correctamente.

```
grant {
(...)
permission java.security.AllPermission;
};
```

En un entorno real no se debería establecer una política tan permisiva, sino que el administrador del sistema debería dar los permisos adecuados solamente al sistema RMI. Los detalles de esta configuración quedan fuera del abasto de este documento.

3.3.1. Instalación de los web services (Axis2)

Es posible utilizar todos los servicios de CoLaTex mediante el uso de web services, además de la interficie web y llamadas RMI.

En esta sección se explica cómo instalarlos en el servidor Jetty, si bien también es posible hacerlos en otros contenedores de servlets[8] como Tomcat o JBoss, entre otros, si se ha instalado Axis2.

Suponiendo que Jetty está instalado y escuchando peticiones en el puerto 8280, el primer paso consiste en acceder a la dirección http://localhost:8280/axis2/

Aparecerá la página que podemos ver en la figura (2).

Después de escribir el nombre de usuario y password (configurados en jetty.xml, por defecto son admin/axis2) hay que pinchar sobre el enlace *Tools/Upload Service* y aparecerá

[8]Los *servlets* son objetos del lenguaje Java que permiten leer una petición HTTP y dinámicamente generar la respuesta. Se puede implementar directamente su código, o generar automáticamente a partir de páginas JSP.

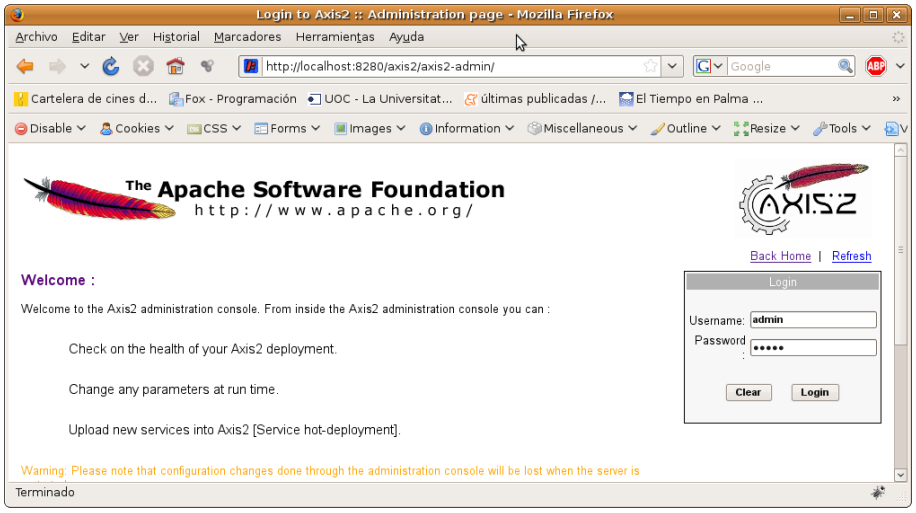

Figura 2: login en Axis2.

la página que podemos ver en la figura (3), desde la cual hay que subir el fichero de web services *colatex.aar*.

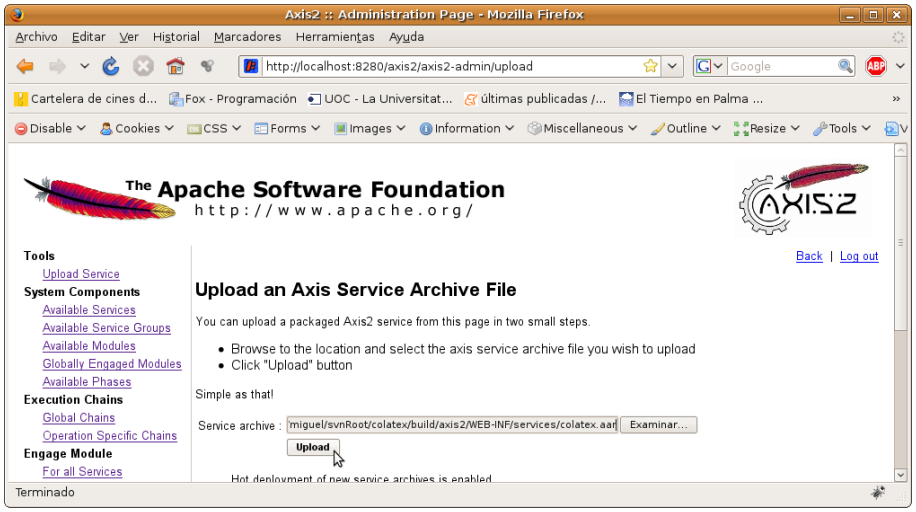

Figura 3: upload del fichero colatex.aar en Axis2.

Una vez finalizado el upload, podemos comprobar que los web services de CoLaTeX están disponibles haciendo click en la optión *System Components/Available Service Groups*. En la figura (4) podemos ver esta página, con los web services de CoLaTeX instalados correctamente y funcionando.

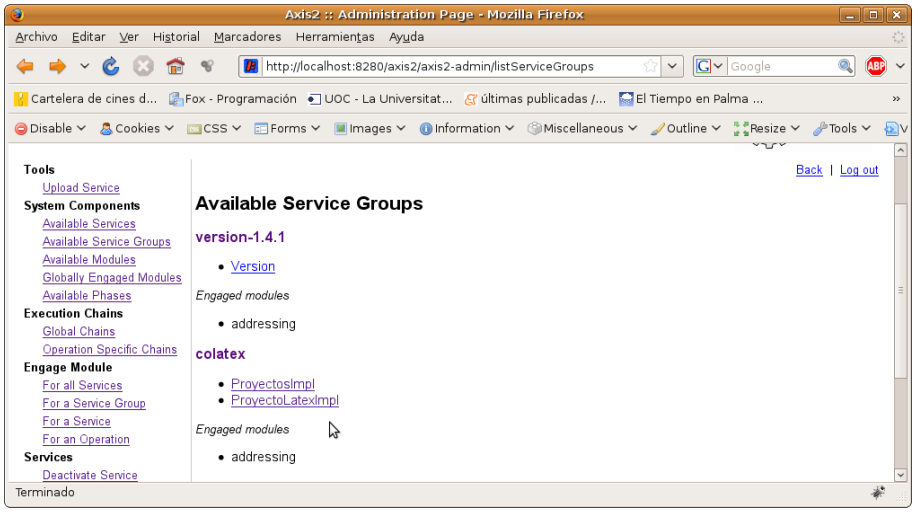

Figura 4: web services de CoLaTeX instalados y funcionando.

3.4. Página principal

La página inicial es la que podemos ver en la figura (5), a la que también se puede llegar desde la pestaña *Inicio*.

3.5. Creación de un nuevo proyecto

Para crear un nuevo proyecto, hay que pinchar sobre la pestaña *Proyectos*, en la que además se pueden editar y eliminar los existentes.

En la figura (6) podemos ver cómo es la pantalla.

Se piden los siguientes datos:

- Nombre del proyecto.
- Descripción.
- Directorio temporal de compilación. Se utiliza para generar el documento final (PDF, por ejemplo) a partir del estado actual del proyecto. Para ello guardará físicamente en el directorio especificado (por ejemplo, /tmp/arboria) del usuario los ficheros que componen el proyecto e invocará en compilador de LaTeX para generar el resultado final.

Figura 5: pestaña *Inicio*.

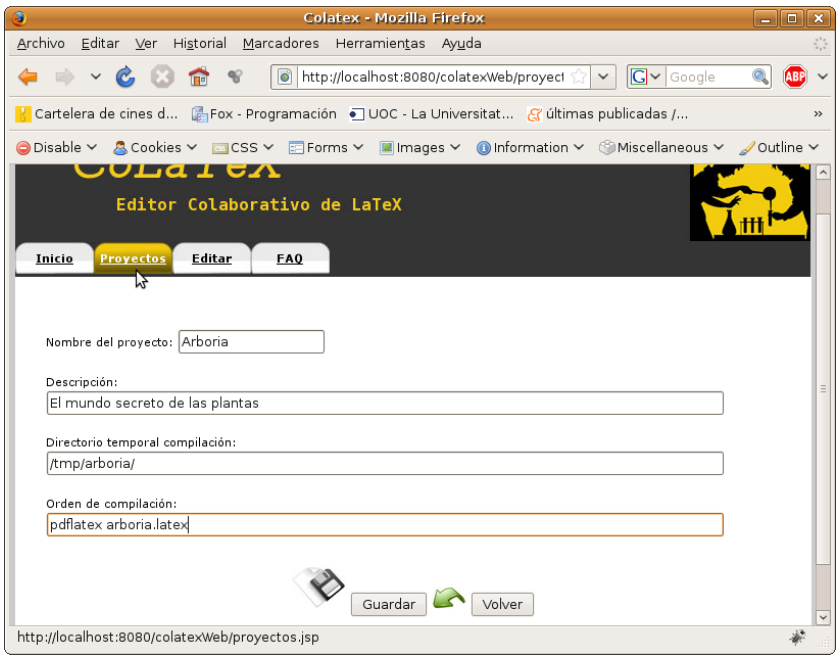

Figura 6: nuevo proyecto.

- Orden de compilación. La orden que es preciso invocar en el ordenador local del usuario para generar el documento final, suponiendo que se ejecuta en el directorio temporal especificado anteriormente. Por ejemplo, pdflatex arboria.latex

3.6. Ver los proyectos

Una vez creado el proyecto, aparecerá en el listado de proyectos (pestaña *Proyectos*), como podemos ver en la figura (7).

Figura 7: nuevo proyecto añadido.

3.7. Seleccionar proyecto para edición

Pinchando en la pestaña *Editar* podemos editar el contenido del proyecto actual.

Si no se ha elegido aún ningún proyecto, aparecerá una pantalla como la de la figura (8), invitando a elegir alguno de los proyectos dados de alta.

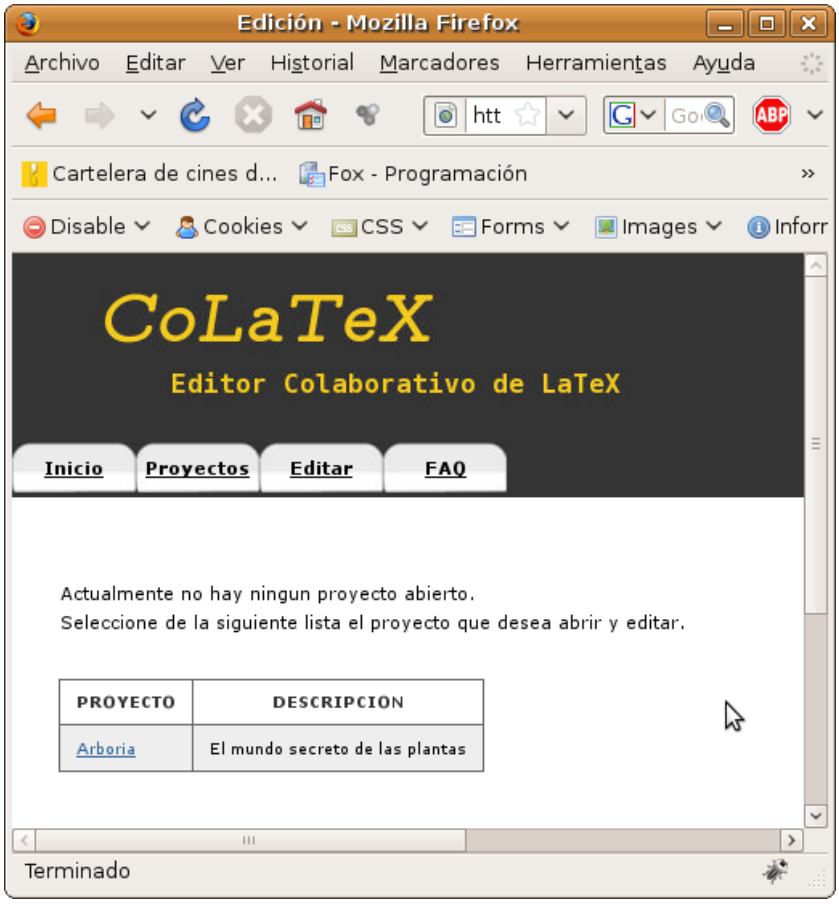

Figura 8: elegir un proyecto con el que trabajar.

3.8. Trabajar con el proyecto seleccionado

Una vez elegido un proyecto, en la pestaña *Editar* vamos a una página en la que tenemos todas las herramientas necesarias para trabajar con el, como podemos ver en la figura (9).

Figura 9: trabajando con un proyecto.

La página muestra:

- El nombre del proyecto con el que se trabaja.
- Botón *generar*: compila el proyecto y obtiene el resultado final en el directorio temporal especificado por el usuario en las propiedades del proyecto.
- Botón *cerrar proyecto*: termina la edición del proyecto actual y permite que el usuario pueda elegir posteriormente cualquier otro.
- Contenido del directorio actual: estructura de directorios y ficheros.
- Fichero seleccionado actualmente.

- Botón *Nuevo fichero*: permite crear un nuevo fichero de texto o binario en el directorio actual.
- Botón *Nuevo directorio*: permite crear un nuevo subdirectorio dentro del directorio actual.
- Botón *Borrar fichero*: borra el fichero seleccionado.
- Botón *Borrar directorio*: borrar el directorio actual, incluyendo todos sus subdirectorios y ficheros.

3.9. Creación de un fichero de texto

Mediante el botón *Nuevo fichero* es posible crear ficheros de texto y binarios, como podemos ver en la figura (10).

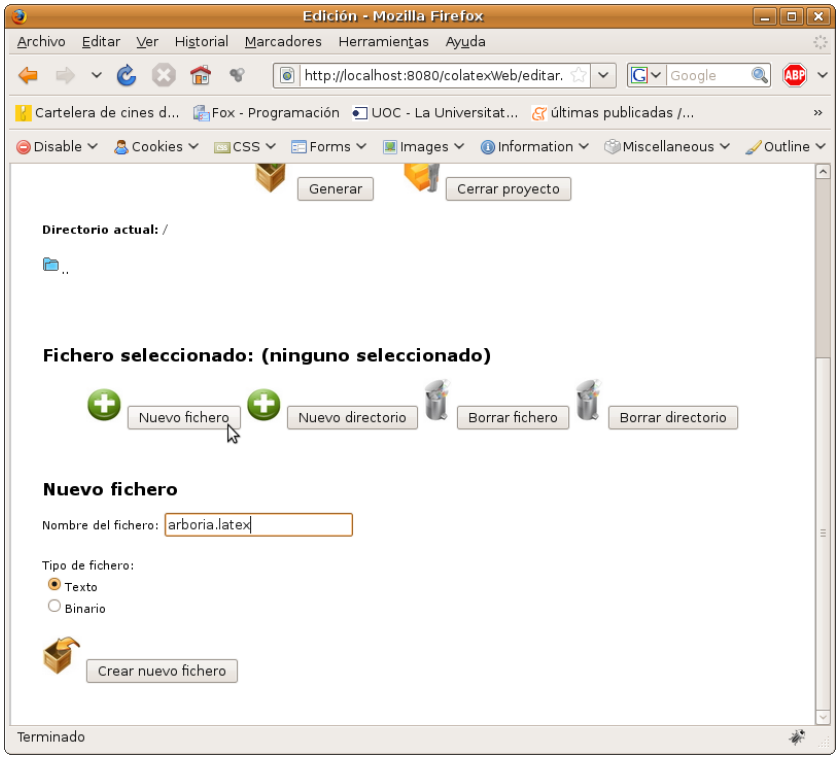

Figura 10: añadir un nuevo fichero de texto.

El fichero se creará en el directorio actual.

3.10. Creación de un directorio

Mediante el botón *Nuevo directorio* se pueden crear nuevos subdirectorios dentro del directorio actual, como se puede observar en la figura (11).

En principio, no existe ningún límite respecto al nivel de anidación.

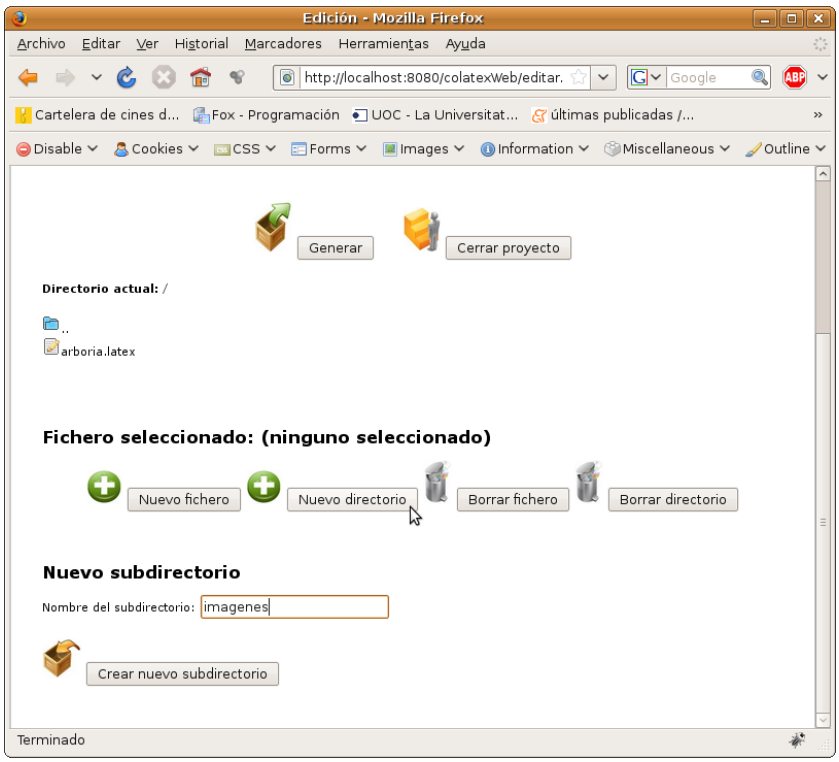

Figura 11: añadir un nuevo fichero de texto.

Para entrar en los directorios basta con pinchar sobre el nombre de directorio correspondiente.

3.11. Creación de un fichero binario

Mediante el botón *Nuevo fichero* también se pueden crear ficheros binarios (por ejemplo, que contengan imágenes en formato JPEG o PNG, entre otros). Simplemente hay que elegir la opción *Binario* al crearlo, como se puede observar en la figura (12).

El fichero se creará en el directorio actual.

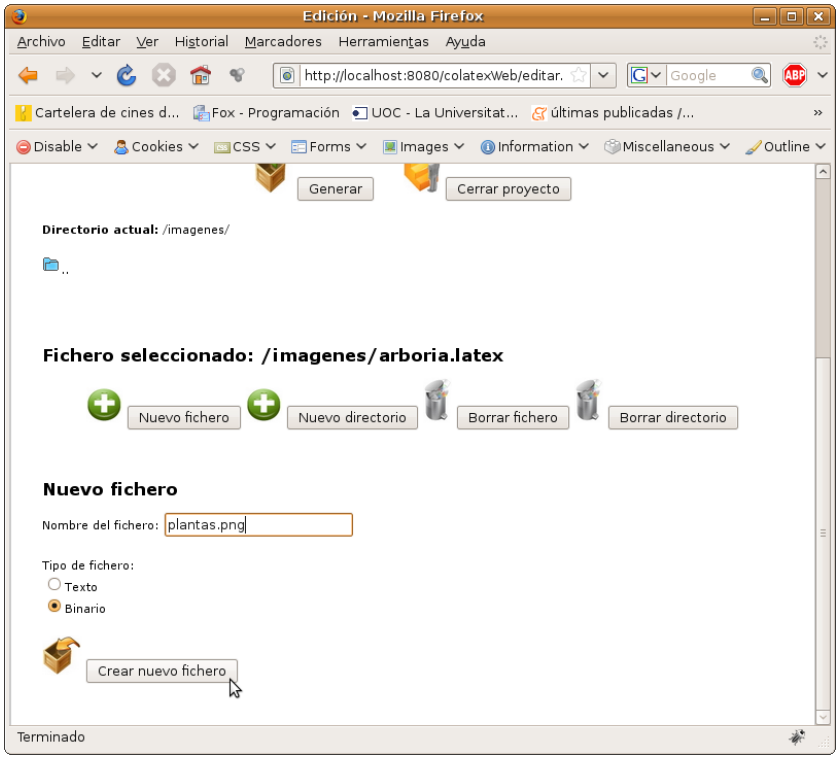

Figura 12: añadir un nuevo fichero binario.

3.12. Edición de un fichero de texto

Para editar un fichero de texto, basta con hacer click sobre su nombre. El sistema detectará que se trata de un fichero de texto y mostrará un área de texto con la que poder editarlo, como se puede observar en la figura (13).

Una vez terminada la edición, se han de enviar los cambios pulsado sobre el botón *Guardar*.

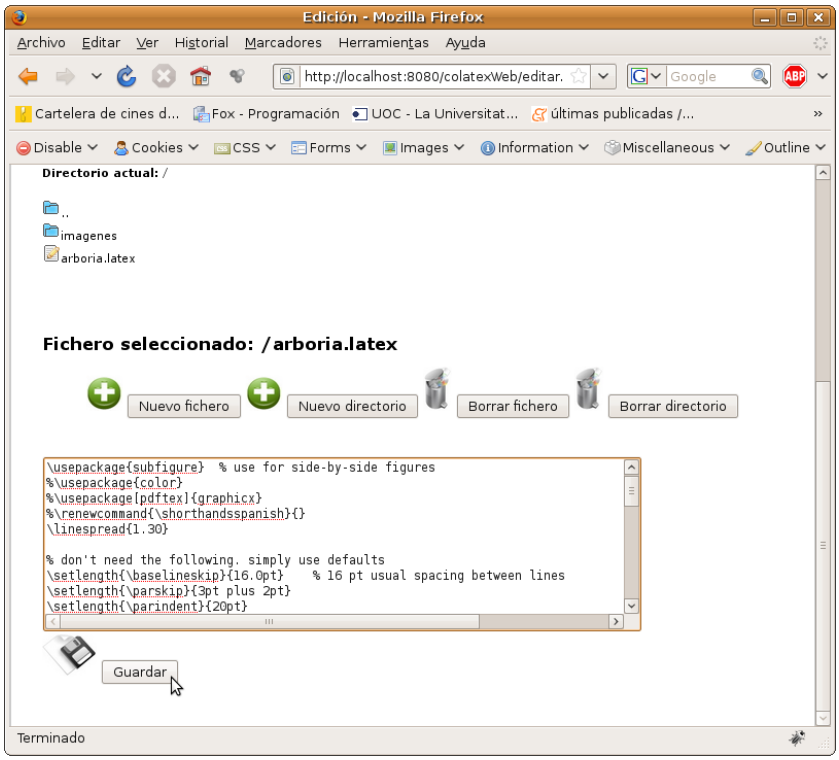

Figura 13: editar un fichero de texto.

Gracias al mecanismo de consistencia del sistema, no hay ningún problema si algún otro usuario ha estado editando y modificando al mismo tiempo el mismo fichero. A la hora de guardarlo, el sistema tendrá en cuenta todas las aportaciones de los usuarios, de forma individual y no excluyente.

Por lo tanto, es muy probable que después de guardar el documento, el resultado que se muestra no sea el mismo contenido que envío el usuario, sino que incluirá los cambios que los demás usuarios fueron enviando mientras se realizaba la edición.

3.13. Edición de un fichero binario

La edición de un fichero binario se inicia haciendo click sobre su nombre, de la misma forma que si fuera un fichero de texto.

El sistema detecta que el fichero es binario, y se muestra una nueva sección desde la cual enviar un fichero del cual se copiará el contenido, como se puede observar en la figura (14).

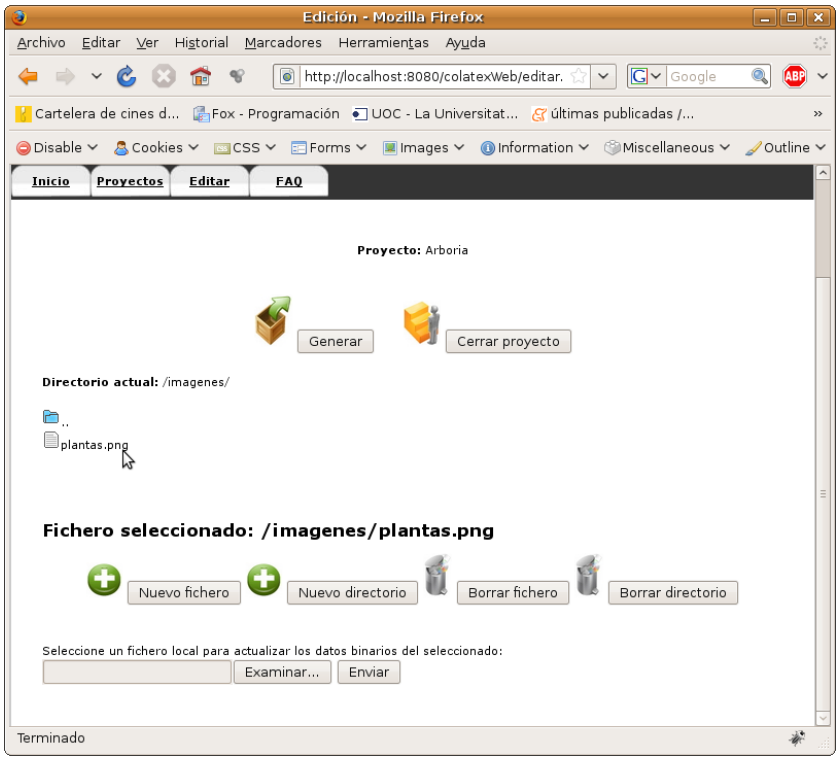

Figura 14: editar un fichero binario.

El mecanismo de consistencia utilizado, basado en el algoritmo WOOT, no contempla el uso de ficheros binarios.

Por lo tanto, cuando el fichero que se edita es binario, el último usuario que haya realizado el cambio será el que consiga guardarlo.

3.14. Generación del documento final

El usuario puede generar en cualquier momento una versión del documento final del proyecto, haciendo click sobre el botón *Generar*. Mientras se genera el resultado, se mostrará la animación que podemos ver en la figura (15).

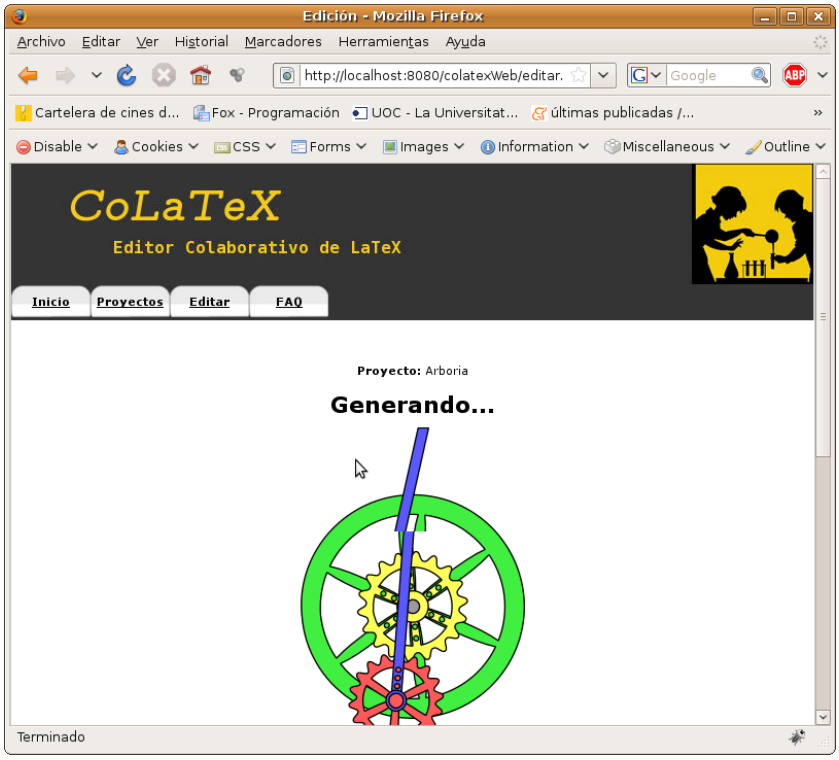

Figura 15: generar el documento final.

El sistema utilizará el directorio temporal que se configuró en ese proyecto para guardar ahí las versiones actuales de todos los ficheros del proyecto, reproduciendo su estructura de directorios.

Cuando los haya descargado todos, ejecutará la orden de generación del documento final en el directorio temporal y se obtendrá el documento final (por ejemplo, un fichero PDF).

3.15. Testing JUnit del núcleo

El funcionamiento de los dos tests JUnit de CoLaTeX se explican en detalle en la sección 7. Aquí se va a explica simplemente cómo ejecutarlos.

Estos dos tests realizan una simulación de la actividad de uno o varios usuarios utilizando la aplicación.

En primero de ellos (*ColatexTest*) crea un proyecto completo desde cero y va llevando a cabo diferentes operaciones, tales como creación de directorios, subida de ficheros binarios, edición del texto de ficheros, etc. Va comprobando que las operaciones se completan correctamente y con los resultados esperados.

El segundo (*ColatexConcTest*) está pensado para comprobar que la edición concurrente de los datos de texto de un mismo fichero funciona correctamente. Antes de ejecutarlo, se ha de haber configurado una red de pruebas con dos ordenadores, de forma que uno sea el nodo inicial y el segundo se conecte al primero.

Dado que los dos tests son muy diferentes entre sí y además el segundo requiere configurar el entorno de manera adecuada antes de lanzarlo, es aconsejable ejecutar los tests desde el IDE, de forma controlada.

En este proyecto se ha utilizado el IDE NetBeans, aunque se puede utilizar cualquier otro como, por ejemplo, Eclipse. Si se utiliza NetBeans, para iniciar el test basta desplegar la sección *Test Packages* y dentro del package *colatex.test* elegir uno de los test con el botón derecho del ratón y seleccionar *Run File*. En la figura (16) podemos ver un ejemplo en el que se inicia el test *ColatexConcTest*.

3.16. Ejecución del núcleo

CoLaTex está formado por dos parte bien diferenciadas, que son un núcleo que ejecuta toda la lógica de negocio, y el conjunto de todos los clientes que acceden al núcleo (aplicación web, clientes consumidores de web services y clientes RMI)

En primer lugar, es necesario iniciar el núcleo de la aplicación, ejecutando la orden *java -jar dist/colatex.jar* en el directorio del proyecto.

En la consola deberían aparecer mensajes log4j[9] similares a los siguientes:

[9]Sistema de registro (log) de mensajes desarrollado por la Apache Software Foundation.

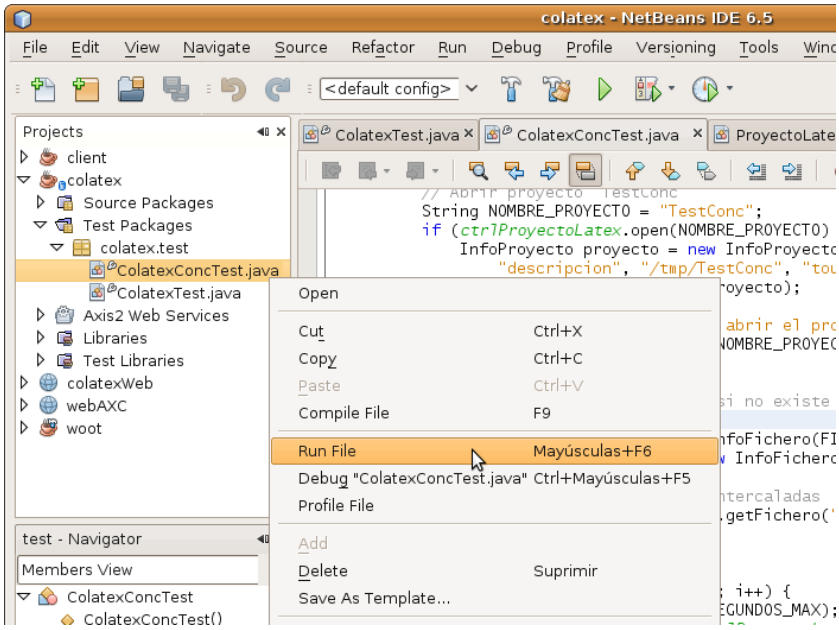

Figura 16: inicio de un test desde el IDE NetBeans.

```
2009-06-03 18:53:14,653 [main] INFO  colatex.Main    - Colatex iniciado.
2009-06-03 18:53:14,657 [main] DEBUG colatex.Main    - PastryKernel init.
:rice.pastry.socket:20090603.185314.861:Error binding to default IP,
using /10.0.0.2:5009 loading DHT : bunshin.properties
2009-06-03 18:53:15,076 [main] DEBUG colatex.Main    - Obteniendo conexion Pastry.
2009-06-03 18:53:15,077 [main] DEBUG colatex.Main    - Creando boot node.
Finished creating new node PastryNode[SNH: <0x6010DB..>//10.0.0.2:5009]
2009-06-03 18:53:18,288 [main] DEBUG colatex.Main    - Publicando servicios RMI.
2009-06-03 18:53:18,600 [main] INFO  colatex.Main    - Colatex durmiendo...
```

La dirección 10.0.0.2 y el puerto 5009 que aparecen en el log se refieren a algún nodo de la red. En principio es indiferente a qué nodo conectarse, pero es necesario conectarse a algún nodo como punto de partida.

Este nodo de partida se configura en los ficheros de configuración *bunshin.properties* y *easypastry-config.xml*.

En bunshin.properties:

```
BUNSHIN_HOST=10.0.0.2
```

```
BUNSHIN_PORT=5009
```

En easypastry-config.xml:

```
<entry key="host">10.0.0.2</entry>
<entry key="port">5009</entry>
```

4. Fase de planificación

La fase de planificación es la primera en el proceso de ingeniería. A partir de los objetivos y requerimientos del proyecto y teniendo en cuenta el punto de partida y el coste económico y temporal, se realizará una temporalización para la planificación, diseño, implementación y testeo.

4.1. Objetivos y requerimientos del PFM

Este PFM tiene los siguientes dos objetivos principales:

1. Concretamente, crear una aplicación de edición colaborativa de documentos LaTeX descentralizada, dentro del proyecto UniWiki.
2. De forma más general, colaborar con el proyecto UniWiki.

La primera parte consiste en crear una aplicación específica. En este caso, se trata que los usuarios puedan trabajar con los ficheros que componen un determinado proyecto de documento en formato LaTeX.

Principalmente, se han de resolver los problemas de la entrada y salida imprevisible de nodos en la DHT, la concurrencia en la edición (update) de los documentos y su consistencia final. Para ello, se cuenta con EasyPastry como middleware y WOOT como algoritmo de consistencia.

Respecto a los aspectos técnicos de la aplicación, se ha de conseguir lo siguiente:

1. La aplicación debe poderse instalar fácilmente en cualquier ordenador preparado para ello (servidor de aplicaciones Jetty), con lo cual pasará a formar parte de la red DHT como un nodo independiente.
2. Los usuarios deben ser capaces de poder crear documentos a partir de ficheros de un proyecto LaTeX, idealmente con la misma sencillez con la que pueden utilizar cualquier otra wiki no distribuida como, por ejemplo Wikipedia. No necesitan saber que están trabajando en un entorno distribuido, sino que estos detalles de la complejidad del sistema les quedan ocultos.

3. Los mensajes entre los nodos de almacenamiento de la DHT se transmitirán en formato XML[10]. Los *commit* y *updates* se realizarán mediante *web services*[11] gestionados por el motor Axis2[12] [10].

4. Se deberá utilizar tecnología Ajax[13] en la interficie web que comunica a los usuarios finales con el frontend.

5. Siempre quedará asegurada la consistencia, pese a la edición concurrente de los ficheros. Esto se conseguirá a partir del algoritmo WOOT.

6. Control de la aplicación mediante web services. Adicionalmente, también son posibles otros métodos, como comunicación mediante RMI o llamadas a los servlets correspondiente. La arquitectura del sistema permite la ampliación con otros métodos de interacción en el futuro.

7. Desarrollar juegos de pruebas (*unit test*) para comprobar la aplicación mediante JUnit[14].

8. Documentar el código de la aplicación mediante JavaDoc.

9. Utilizar Subversion para la gestión del código fuente.

Por otra parte, también es el objetivo del PFM colaborar en el desarrollo y mejora del proyecto UniWiki.

En concreto,

- Detectar, reportar y corregir errores en UniWiki. Para ello se utiliza la aplicación *trac* del proyecto en SourceForge.

- Colaborar con la comunidad de software libre dentro del proyecto UniWiki, mediante listas de correo, foros, etc.

- En general, contribuir a la mejora de UniWiki.

[10] *Extensible Markup Language* (lenguaje de marcas extensible).

[11] Los servicios web o *web services* son un conjunto de protocolos y estándares para el intercambio de información entre diferentes plataformas de hardware y software, con el objetivo de asegurar la interoperatividad.

[12] Versión 2 de la implementación del protocolo SOAP por parte de la Apache Foundation.

[13] *Asynchronous JavaScript And XML* (JavaScript asincrono y XML).

[14] *Framework* de clases Java creadas por Erich Gamma y Kent Beck para el desarrollo de tests unitarios.

4.2. Punto de partida

En el momento de comenzar este PFM ya se cuenta con una cierta infraestructura, ya que UniWiki hace uso de:

- EasyPastry como middleware[15].
- WOOT como algoritmo de consistencia.

El middleware es el software que provee de una capa común para los nodos que almacenan los datos de la wiki (o en general, de los nodos que pertecen a un sistema distribuido[16]), de forma que los abstrae de los detalles de la red subyacente, formada por máquinas funcionando con diversos sistemas operativos, lenguajes de programación, protocolos de comunicación, etc. A esta capa común se accede mediante una API[17] determinada que todos los participantes conocen y utilizan.

En el proyecto UniWiki se utiliza EasyPastry [4] como middleware. Está basado en la idea de la Common API [2], en la que se provee de un nivel de abstracción middleware formado por tres capas:

Capa 0 o KBR (Key-Based Routing) : define el encaminamiento y la topología de la red. Es una capa utilizada en la gran mayoría de aplicaciones distribuidas que utilizan una red superpuesta[18].

Capa 1 : aprovecha la infraestructura de la capa KBR para crear servicios de mayor nivel, como DHT[19], envío de mensajes *anycast* y *multicast* (CAST), y encaminamiento y localización descentralizada de objetos (DORL).

Capa 2 : la implementación concreta de los servicios definidos en la capa anterior. Se podría tener, por ejemplo, EasyPastry para KBR, Bunshin para la DHT o Scribe para la aplicación CAST, entre otros.

[15] Capa que abstrae al programador de los detalles de la red de comunicaciones subyacente, de los diferentes sistemas operativos presentes en ella y, en general, de la heterogeneidad de los componentes del sistema.

[16] Un sistema distribuido es un conjunto de ordenadores autónomos comunicados por una red de comunicaciones que gracias a un middleware específico forman una red homogénea de servicios integrados.

[17] *Application Programming Interface* (interfaz de programación de la aplicación.

[18] Red a nivel de aplicación en la que se encaminan los mensajes de los nodos de un sistema P2P.

[19] *Distributed Hash Table* (tabla de dispersión distribuida).

Para poder crear y utilizar la red DHT de la capa 1, EasyPastry hace uso internamente de Bunshin [5], el cual proporciona una tabla de hash distribuida, construida sobre una red P2P[20], con un sistema de replicación activo, además de *catching* adaptativo.

Bunshin queda encapsulado totalmente dentro de EasyPastry, abstrayendo al programador de los detalles de encaminamiento y replicación, entre otros. Es posible identificar esta situación con el patrón de diseño de tipo estructural denominado *Facade*. Para más información, es recomendable consultar la obra *Design Patterns* [7], considerada hoy en día una referencia clásica en cuanto a patrones de diseño.

Los usuarios acceden y escriben concurrentemente en los nodos de almacenamiento. Para asegurar la consistencia se hace uso del algoritmo WOOT [3].

WOOT garantiza la consistencia eventual (*eventual consistency*) y causal (*causal consistency*) y la preservación de intención (*intention perservation*).

En los nodos de almacenamiento de la UniWiki no se almacena realmente el contenido final de los documentos o ficheros binarios, sino que se guarda una descripción de los pasos que ha llevado a cabo el usuario para generar el contenido. Múltiples usuarios pueden estar editando la misma o diferente versión de un mismo documento, de forma que es necesario que se asegure la consistencia de la información, para evitar duplicados, pérdida de información o variación del orden temporal.

Para ello, se utiliza WOOT como algoritmo de consistencia, el cual se encarga de obtener la secuencia de operaciones que ha llevado a cabo cada usuario para una determinada versión del documento (*update*), para ponerlas en común (*merge*) y obtener finalmente un único documento consistente que luego será guardado en los nodos de almacenamiento de la DHT.

En la web del proyecto UniWiki en el repositorio SourceForge [6] existe un prototipo que hace uso de las librerías y algoritmos que se han presentado en esta sección.

4.3. Temporalización del proyecto

A la hora de planificar este PFM, hay que tener en cuenta que existen unas fechas ya fijadas para la entrega de cada una de las cuatro PAC[21] que lo componen.

Estas cuatro PAC y sus fechas son:

[20] *Peer To Peer* (de igual a igual).

[21] Prova d' Avaluació Continuada.

PAC #1 : se corresponde con el documento de definición, requerimientos y objetivos iniciales del PFM.

Fecha de entrega: 19 de marzo de 2009.

PAC #2 : propuesta formal del proyecto que se va a desarrollar.

Fecha de entrega: 2 de abril de 2009.

PAC #3 : entrega del documento que consta de título, objetivos, alcance, motivación/justificación, área de interés, marco de desarrollo, antecedentes, software que se utilizará, referencias bibliográficas, planificación temporal, pliego de condiciones y presupuesto.

Fecha de entrega: 23 de abril de 2009.

PAC #4 : Desarrollo íntegro del proyecto de acuerdo con la metodología de la asignatura y respetando los acuerdos establecidos.

Fecha de entrega: 29 de mayo de 2009.

Las fechas de las PAC marcarán los plazos máximos para la consecución de algunos hitos que se marcarán en el desarrollo del proyecto.

El proyecto constará de cuatro fases, que son:

1. **Preparación y planificación.**
2. **Diseño e implementación en modo no-distribuido.**
3. **Diseño e implementación en modo distribuido.**
4. **Experimentación y testeo.**

Esta es una previsión inicial, y es posible que algunas de las tareas se modifiquen, se añadan más o se cambien algunos plazos, según se avance en el desarrollo del proyecto. Sin embargo, es deseable que se cumplan los plazos establecidos.

Para planificar las tareas se utilizará la metodología WBS[22], con diagramas de Gantt en cada fase del proyecto. En la obra *How to Build a Work Breakdown Structure* [8] se explican los detalles de esta metodología de trabajo.

[22] *Work Breakdown Structure* (estructura de descomposición del trabajo).

Respecto al modelo de desarrollo que se utilizará, será el modelo incremental e iterativo, siguiendo en la medida de lo posible el RUP[23] [12].

Será incremental, ya que estará compuesto de varias fases (cuatro, en concreto), en las que se irá pasando de una fase a la siguiente a medida que se vayan añadiendo ciertas funcionalidades en forma de hitos a conseguir. Cada fase toma como base la aportación de las anteriores y añade más funcionalidades. Al llegar a la última fase, el proyecto cumple con todos los requisitos y es plenamente funcional.

Será iterativo, ya que la metodología que se sigue cumple con las fases que marca el RUP, que son:

- Modelado de negocio.
- Requisitos.
- Análisis y Diseño.
- Implementación.
- Pruebas.
- Despliegue.

El modelado de negocio y el despligue quedan fuera del ámbito de este proyecto.

4.3.1. Preparación y planificación

Es la primera fase, en la que se lleva a cabo toda la preparación necesaria para poder desarrollar el proyecto de forma efectiva en las siguientes fases.

En concreto, se llevan a cabo las siguientes tareas:

- (1) Instalar IDE[24]. En principio se instará NetBeans IDE, aunque se podría posteriormente sustituir o complementar con otros como, por ejemplo, Eclipse.
- (2) Instalar servidor de aplicaciones Jetty y comprobar que funciona correctamente.
- (3) Configurar Subversion para poder trabajar tanto en local como con el repositorio de la aplicación UniWiki en SourceForge.

[23] *Rational Unified Process* (proceso unificado de Rational).
[24] *Integrated Development Environment* (entorno integrado de desarrollo).

- (4) Obtener la última versión del prototipo UniWiki del repositorio SourceForge.
- (5) Compilar el prototipo y realizar un *deploy* en Jetty para comprobar que el entorno de desarrollo y pruebas es funcional.
- (6) Documentar algunas funciones del prototipo con *JavaDoc* y comprobar que la documentación se genera correctamente.
- (7) Crear algunos tests unitarios sencillos con la intención de comprobar que JUnit funciona correctamente y que es posible invocar rápidamente los tests desde el GUI[25] del IDE.
- (8) Codificar alguna función sencilla que haga uso de los web services, para comprobar que la librería Axis2 funciona correctamente.
- (9) Crear una interficie web muy sencilla para comprobar comunicación Ajax entre cliente y servidor web.
- (10) Integrar Ajax y XML en la interficie web anterior.

En la PAC #2 de deberá entregar la propuesta formal del proyecto que se va a desarrollar, y su fecha de entrega es el 2 de abril. Antes de terminar de formalizar los detalles de la propuesta, el entorno de desarrollo debe ser enteramente funcional, aunque se realizará la propuesta al mismo tiempo que se prepara el entorno.

Se fija como plazo máximo para finalizar esta fase el 30 de marzo.

En la figura (17) podemos ver el diagrama de Gantt de esta fase del proyecto.

4.3.2. Diseño e implementación en modo no distribuido

Es la segunda fase, que solo comenzará después de haber completado la primera, es decir, cuando se haya comprobado que el entorno de desarrollo es totalmente funcional y que no se van a presentar problemas relacionados con la configuración del sistema (IDE, Subversion, librerías, etc). La fecha prevista de inicio es el 3 de marzo y su finalización no está determinada, en principio, por la fecha de entrega de las restantes PAC.

En esta fase se desarrollarán los algoritmos básicos para conseguir la comunicación RMI[26], y leer y escribir objetos serializables en la DHT mediante EasyPastry, aunque

[25] *Graphical User Interface* (interfaz gráfica de usuario).
[26] *Remote Method Invocation* (invocación remota de métodos).

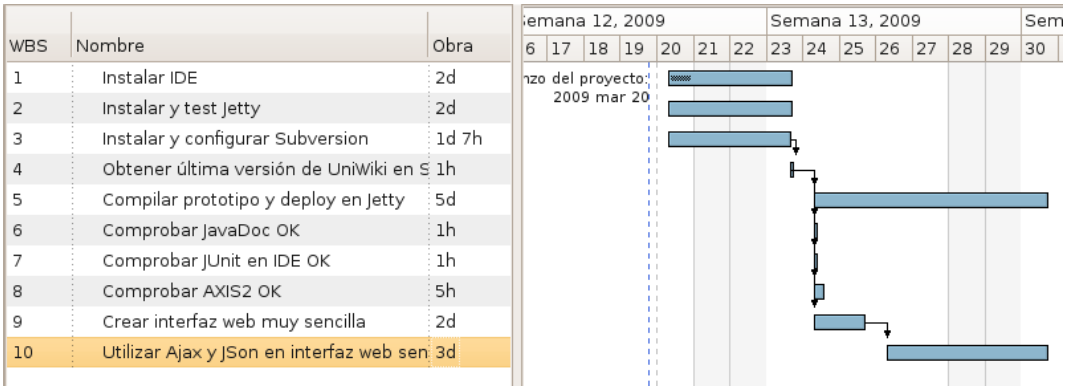

Figura 17: fase de preparación: diagrama de Gantt.

utilizando solamente uno o dos nodos de almacenamiento para las pruebas, y sin tener en cuenta la concurrencia.

Además, también se comenzará en esta fase el diseño de la interficie web con el lenguaje HTML y hojas de estilo CSS. El diseño inicial de la fase anterior se descartará, ya que era únicamente una versión muy inicial para poder probar el prototipo. Ahora se decidirán que funciones en concreto deberá ofrecer la interficie web y con qué diseño (por ejemplo, si se utiliza un área de texto para editar, botones de commit o update, información de estado como la versión que se edita, etc).

También se comenzará a utilizar Ajax e intercambio de mensajes mediante XML entre el cliente y el frontend.

Aún así, esta fase aún no se contempla que haya múltiples nodos DHT colaborando para almacenar la información, sino que este último paso se deja para la siguiente fase. Sin embargo, sí que se comenzará a escribir el código necesario para cuando existan varios nodos DHT, en previsión de la siguiente fase, aunque en esta fase simplemente se utilizará uno.

Pero aunque en esta fase solamente se considere un único nodo de almacenamiento en la DHT, los clientes acceden a CoLaTeX realizan updates y commits son múltiples y realizan estas operaciones de forma concurrente. Por lo tanto, es necesario el uso del algoritmo WOOT para asegurar la consistencia y unicidad de los ficheros almacenados que forman el *workspace* del proyecto.

En esta fase se llevan a cabo las siguientes tareas:

- (1) Diseñar las funcionalidades ofrecidas por la interficie web del frontend.
- (2) Diseñar la interficie web mediante lenguaje HTML y hojas de estilo CSS.
- (3) Integrar Ajax y XML en el diseño anterior.
- (4) Utilizar *web services* (Axis2) como una forma más de gestionar los proyectos.
- (5) Realizar tests para comprobar que el intercambio de mensajes en (4) es correcto.
- (6) Utilizar WOOT para asegurar la consistencia y la unicidad del documento final al que se accede desde el frontend.
- (7) Testear consistencia conseguida mediante WOOT.

Se fija como plazo máximo para finalizar esta fase el 24 de abril.

En la figura (18) podemos ver el diagrama de Gantt de esta fase del proyecto.

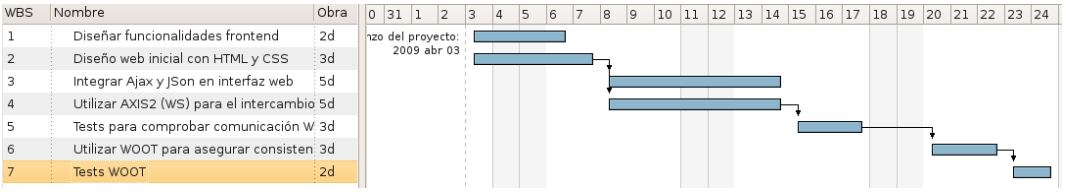

Figura 18: fase de desarrollo no distribuido: diagrama de Gantt.

4.3.3. Diseño e implementación en modo distribuido

En esta tercera tercera fase del proyecto se terminan de implementar las funcionalidades que faltan para poder cumplir con todos los requerimientos.

Comienza el 27 de abril, y se corresponde con la entrega de la PAC #4 (cuya entrega está fecha en el 29 de mayo). Como para finalizar el proyecto es necesario haber realizado la fase de testeo, se fija el plazo máximo de entrega el 21 de mayo. De esta forma, se puede realizar el testeo desde el día 22 al 29 de mayo.

En esta fase es necesario crear una red DHT con varios nodos (hasta el momento solo se había utilizado uno) y estudiar las modificaciones que puede requerir el código para hacer frente a la concurrencia de operaciones, si es que se requiere alguna.

Para tener varios nodos DHT, se crearán varias máquinas virtuales (por ejemplo, mediante VirtualBox), de forma que se conecten a la red local como si fueran máquinas independientes, con diferentes direcciones IP. Esto se conseguirá mediante *bridging*. También se utilizarán máquinas reales dentro de la red local e Internet, aparte de las virtualizadas.

Además de virtualizar los nodos de almacenamiento de la DHT, también se crearán diversos clientes que simularán usuarios reales realizando updates y commits de forma concurrente. Esto permite realizar un test de carga del sistema (ver cómo se comporta cuando muchos clientes solicitan operaciones), y sobre todo comprobar que el algoritmo WOOT está asegurando la consistencia de los documentos cuando se acceden y actualizan de forma concurrente.

Para comprobar que los datos que generan los usuarios simulados, se decidirá un método que permita saber si el orden en el que aparecen las aportaciones en el documento mantienen la coherencia. Por ejemplo, mediante un timestamp que los usuarios simulados escriban como parte del contenido que editan o crean.

Por lo tanto, las tareas que se llevarán a cabo en esta fase son:

- (1) Modificar el código fuente de los nodos DHT para que funcionen correctamente en un escenario distribuido y colaborativo cuando múltiples participantes realizan lecturas y escrituras concurrentemente.
- (2) Crear varias máquinas virtuales (VM) con diferentes IP (bridging) en la red local.
- (3) Diseñar un programa que simule la operación de un nodo DHT.
- (4) Diseñar un programa que simule la operación de un usuario de la wiki: periodica y aleatoriamente obtener una versión de algún documento y lo modifique.

Se fija como plazo máximo para finalizar esta fase el 21 de abril.

En la figura (19) podemos ver el diagrama de Gantt de esta fase del proyecto.

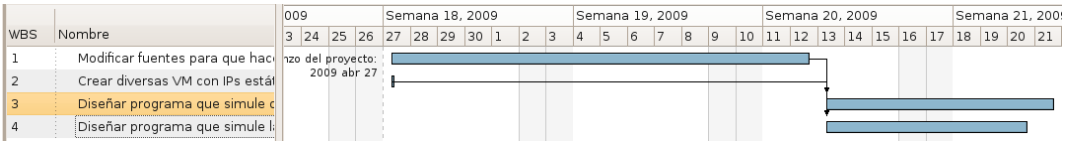

Figura 19: fase de desarrollo distribuido: diagrama de Gantt.

4.3.4. Experimentación y testeo

Es la última fase del proyecto, y se han previsto dos tipos de tests.

Uno para comprobar el funcionamiento de las clases del núcleo, para lo cual el test simula la actividad de un usuario que crea un nuevo proyecto desde cero, añade directorios y ficheros, los borra, edita el contenido de texto de los ficheros, sube contenido binario, etc.

El otro test comprueba que los mecanismos de consistencia funcionan correctamente en la edición concurrente de los documentos de texto. Para ello es necesario configurar un entorno con dos máquinas diferentes y ejecutar simultáneamente el test en cada una de ellas. El test hace que ambas máquinas realicen ediciones concurrentes, aunque esperan un tiempo aleatorio cada vez que terminan de editar el fichero y repiten la operación.

Luego, el test obtiene el contenido final del fichero de texto y comprueba si es coherente.

En la sección 7 se dan todos los detalles de estos dos tests y se muestran los resultados obtenidos para la versión actual de CoLaTeX.

Esta fase durará desde el día 22 hasta el día 27 de mayo. El día 29 es la entrega final del proyecto, así que se dispondrá de dos días para revisar el documento final.

4.4. Tareas comunes y trasversales

Durante el desarrollo del proyecto, y en cualquiera de sus fases, se llevarán a cabo las siguientes tareas:

- Ir documentando el código Java desarrollado con comentarios para JavaDoc.
- Ejecutar periódicamente los juegos de pruebas JUnit, especialmente después de implementar alguna nueva funcionalidad o modificación importante.
- Contribuir a la comunidad de SL con aportaciones en listas de correo, foros, ...

Estas tareas son transversales al proyecto de desarrollo como, por ejemplo la contribución a la comunidad de SL, mientras la ejecución de los juegos de pruebas JUnit se han de ir realizando periódicamente para asegurar que con la inclusión o modificación de funcionalidades no se introducen nuevos errores[27].

[27] Evidentemente, pasar los tests no garantiza que el programa esté libre de errores, pero sí que son una ayuda muy valiosa durante el desarrollo.

La documentación del código fuente mediante JavaDoc también es necesario realizarla de forma continuada, y nunca esperar a finalizar un determinado módulo para comenzar a documentarlo.

El motivo es que los demás desarrolladores del proyecto necesitan una forma rápida de poder entender para qué sirve cada clase y método del proyecto, a ser posible, sin necesidad de tener que analizar los algoritmos y estructuras del código fuente.

Una buena documentación, actualizada periódicamente, da al resto del equipo una visión global del proyecto, además de facilitar la integración e implementación de nuevas funcionalidades en módulos que han sido diseñados por personas diferentes.

4.5. Estimación del coste

Este es esencialmente un proyecto de desarrollo de software en el que se utilizan herramientas de software libre, así que no existe un coste económico asociado a licencias de uso. Sin embargo, es interesante calcular el TCO[28], que comprende las siguientes partidas:

- Coste total del hardware y software.
- Mejoras previstas en el hardware y software (actualizaciones).
- Mantenimiento.
- Soporte técnico.
- Formación a los usuarios.

Para realizar los cálculos, vamos a suponer un caso de estudio compuesto por 5 máquinas actuando como nodos DHT y una máquina como frontend. Cada una de estas máquinas será un PC estándar, los cuales tienen un coste medio de unos 1000 euros/unidad.

La fase de preparación tiene una duración de 7 días laborables (9 días), y en ella se realizan tareas de configuración. Se puede suponer que un técnico informático puede encargarse de ello dedicando 3 horas diarias. Si su salario es de 800 euros mensuales, tendríamos que sería necesario invertir 90 euros en esta primera fase.

La fase de desarrollo no distribuido requiere que uno o varios ingenieros o especialistas en computación distribuida creen un prototipo que integre Ajax, XML, Axis2 y WOOT.

[28] *Total Cost Ownership* (coste total de la propiedad).

Suponiendo que lo realizará una sola persona con un salario de 2000 euros mensuales y dedicando 8 horas diarias, tenemos que para los 16 días laborables (22 días) que dura el proyecto el coste sería de 1467 euros.

La fase de desarrollo distribuido debería realizarla el mismo equipo que se encargó de la fase anterior. Suponiendo el mismo salario y dedicación, los 19 días de trabajo (25 días) tendrían un coste de 1667 euros.

De igual forma, la fase de testeo de 6 días laborables (9 días) también la realizaría el mismo equipo y tendría un coste de 600 euros.

En definitiva, el coste económico de desarrollar (software y configuración) todo el proyecto serían 90 + 1467 + 1667 + 600 = 3824 euros.

Además, hay que añadir el coste en el hardware de las máquinas. Si llamamos D al número de máquinas en la DHT y P al coste de un PC, tenemos que el coste total es (1+D)P.

Sumando todas las cantidades, tenemos que el coste total es 3824 + (1+D)P.

Para este caso de estudio, con 5 máquinas en la DHT y un frontend, el coste final sería de 3824 + (1+5)*1000 = 9824 euros.

Respecto al coste del software, no existe coste asociado a licencias si se basa en SL, pero sí que sería conveniente contar con algún administrador para gestionar el sistema distribuido y ofrecer cierto soporte técnico.

Los usuarios no necesitan una formación especial, ya que el sistema se ha diseñado de forma que no tengan por qué saber que están utilizando un sistema distribuido, así que se espera que el tiempo que se tenga que dedicar a su formación sea escaso y que puedan editar sus proyectos LaTeX colaborativos sin mayor dificultad.

5. Fase de diseño

Una vez finalizada la fase de planificación, se prosigue con la de diseño. Aún así, es posible regresar a la fase de planificación si se requieren cambios en ella, al ser un proceso incremental e iterativo.

5.1. Arquitectura del sistema

En esta sección se mostrará cuál es la arquitectura global del sistema, describiendo los elementos que componen CoLaTeX, y cuál es la arquitectura interna de los clientes, la cual permite separar totalmente la lógica de negocio de las posibles capas de presentación.

5.1.1. Arquitectura global

La arquitectura global de este sistema la podemos ver representada en la figura (20). Esta imagen ha sido tomada del documento *UniWiki: A Reliable and Scalable Peer-to-Peer System for Distributing Wiki Applications* [1], en el que se explica con mayor detalle la arquitectura, el modelo de datos y algoritmos que utiliza UniWiki.

La arquitectura en CoLaTeX es la misma que en UniWiki, aunque con algunas pequeñas diferencias funcionales, como se explica a continuación.

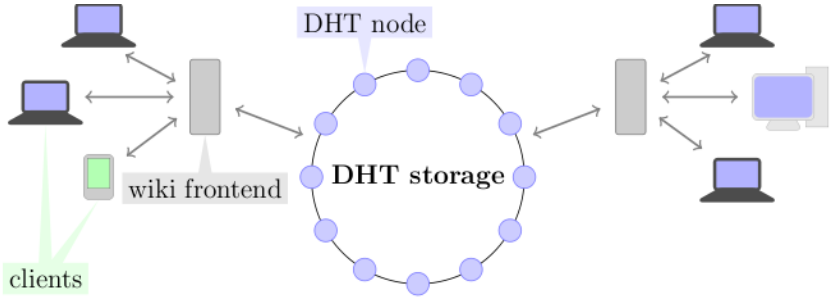

Figura 20: arquitectura global del sistema.

Es posible diferenciar varios tipos de entidades según su función:

Nodos de almacenamiento (DHT storage) : son los nodos de la red que almacenan la información de las páginas Wiki en Uniwiki, y los diferentes proyectos y sus *workspace* LaTeX, en CoLaTeX. Colaboran para asegurar la coherencia de los datos, al mismo tiempo que se distribuyen la información entre ellos.

Wiki frontend : es la parte visible de la red UniWiki, que se encarga de proveer un punto de conexión HTTP para que los clientes puedan acceder a la aplicación UniWiki con su navegador web. Oculta y abstrae a los clientes finales de la complejidad y las interacciones que se producen entre los nodos de almacenamiento y el propio frontend. En CoLaTeX su equivalente sería la aplicación web, la cual provee de un interficie gráfica con la cual el usuario accede a los proyectos.

Clientes : son los usuarios finales de la UniWiki o CoLaTeX, equiparables a los usuarios que consultan páginas de Wikipedia, por ejemplo. Se conectan al frontend para consultar y editar los ficheros de su proyecto LaTeX de forma colaborativa mediante una interficie web. Idealmente, no deberían poder determinar si están utilizando un sistema centralizado o distribuido, porque el frontend les abstrae de estos detalles, situándoles en una capa de nivel superior.

Los nodos que almacenan la información lo hacen en base a una DHT, un mecanismo basado en el funcionamiento de las estructuras de datos conocidas como tablas de dispersión[29]. La DHT permite seleccionar un nodo de la red en base al resultado de aplicar una función de *hash* a una determinada clave. Esto permite implementar las funciones de *put(key, value)* y *get(key)* y repartir de forma homogénea la información entre los diversos nodos de la red.

Para interactuar con la DHT y hacer uso de servicios adicionales, como la replicación de datos y caché (aportada por Bunshin), se utiliza EasyPastry. Su arquitectura la podemos ver en la figura[30] (21).

5.1.2. Arquitectura interna de los clientes

Inicialmente se pensó en crear una aplicación web tradicional, basada en servlets y el modelo JSP[31] 2, en la que los servlets reciben las peticiones de los clientes, las procesan, y finalmente redirigen al cliente a la página JSP correspondiente.

En este caso, los servlets procesarían las peticiones mediante llamadas a la librerías

[29] Estructura de datos que utiliza una función de hash para obtener directamente la posición de un elemento en una tabla a partir de su clave. En inglés se conoce también como *hash table* o *hash map*.

[30] La imagen ha sido obtenido de la web oficial de EasyPastry, en *http://ast-deim.urv.cat/easypastry/*

[31] *JavaServer Pages* (páginas de servidor en Java).

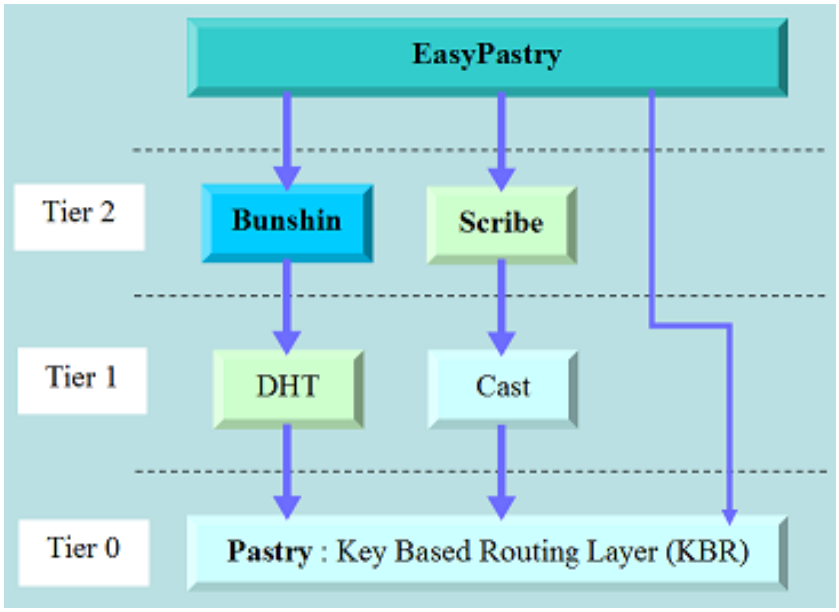

Figura 21: arquitectura de capas de EasyPastry.

EasyPastry para leer y escribir en la DHT y también a WOOT, para gestionar la consistencia de los ficheros de texto.

Durante la fase de desarrollo iterativo e incremental se vio que aunque esta arquitectura era funcional y típico, era al mismo tiempo poco flexible en cuanto a la depuración y el desarrollo incremental de nuevas funcionalidades. Era preciso depurar una aplicación web completa, lo cual es posible, pero mucho más lento que utilizar un debugger en una aplicación Java *standalone*. Además, cuando las modificaciones implicaban cambios en el código fuente (no así en las JSP o los servlets) era necesario volver a hacer un *deploy* de la aplicación, perdiendo datos de sesión.

Para evitar estos problemas y hacer más rápido el desarrollo y la depuración, se optó por modificar esta arquitectura y separar la aplicación cliente en dos partes bien diferenciadas:

- Aplicación standalone con la lógica de negocio.
- Aplicación web, con la capa de presentación.

La primera es una aplicación Java standalone tradicional, el cual implementa toda la lógica de negocio del sistema. El separarlo de la aplicación web presenta varias ventajas,

como ya se había adelantado, como la facilidad de depuración, el desarrollo incremental e iterativo del software y la separación total de la lógica de negocio respecto a la capa de presentación.

La otra parte es la aplicación web, la cual no tiene ninguna información ni responsabilidad en cuanto a la lógica de negocio, y únicamente se encarga de interaccionar con el usuario (recibir los *clicks* realizados en los controles de la web, presentar páginas de resultados, ...), de comunicarse con la parte de lógica de negocio para invocar funciones y recoger resultados, y presentar estos resultados o los correspondientes mensajes de error al usuario. Esta separación también beneficia a la capa de presentación, ya que facilita su depuración (toda la lógica de negocio y las librerías asociadas están en una aplicación separada, lo cual simplifica mucho la aplicación web) y al mismo tiempo que consigue desacoplar las dos capas.

Técnicamente, esta arquitectura es un ejemplo del patrón MVC[32], en la que el modelo está en la aplicación Java standalone, y la vista y el controlador en la aplicación web. En la figura (22) podemos verlo esquematizado.

Aunque las dos aplicaciones son diferentes y funcionan de forma independiente, es necesario que se puedan comunicar. Para ello, se han utilizado los mecanismos RMI[33] de Java.

Para utilizar RMI, la aplicación standalone selecciona qué clases contienen métodos que van a ser accesibles desde el exterior, y crea dos diferentes: una interfaz que extiende a *java.rmi.Remote*, y otra que contiene una implementación de la interfaz. Finalmente, se instancia el registro RMI y se publican todas estas clases.

Por su parte, la aplicación web solamente necesita conocer la interfaz de las clases para invocar sus métodos, así que mediante una llamada a RMI crea un objeto funcionalmente equivalente al objeto presente en la aplicación standalone.

Desde el punto de vista del programador, no hay diferencia entre los objetos locales y los RMI. Queda totalmente abstraído de la comunicación RMI y puede utilizar los objetos como si se ejecutaran localmente.

Esta separación de la lógica de negocio de la presentación puede ser incluso física, permitiendo distribuir la carga del sistema en diferentes máquinas si fuera necesario. La

[32] *Model-View-Controller.* (Modelo-Vista-Controlador)
[33] *Remote Method Invocation* (invocación de método remoto).

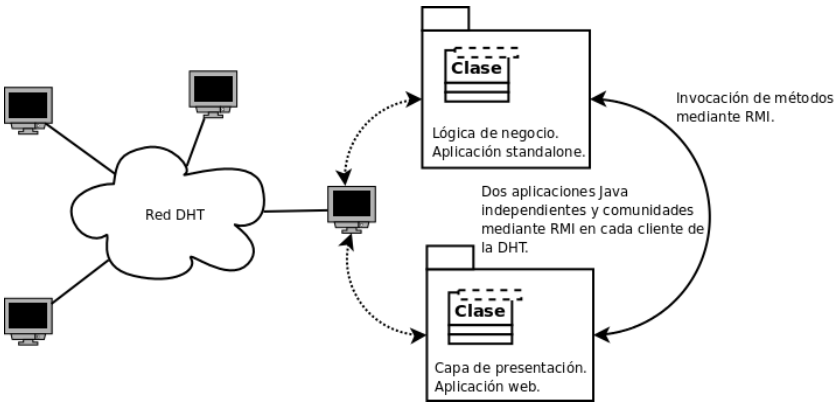

Figura 22: arquitectura interna de los clientes.

aplicación web encargada de la presentación simplemente tendría que ser configurada de manera que instanciara cada objeto desde la máquina correspondiente.

En la forma más simple, que será la que se utilizará normalmente, la aplicación asume que todos los objetos que se han de instanciar residen en la misma máquina, si bien se puede cambiar esta comportamiento fácilmente y se deja abierta la posibilidad de distribuir la carga entre varias máquinas.

Es importante dejar claro que aunque la carga quede distribuida entre varias máquinas o se utilicen los mecanismos RMI dentro de la misma máquina para comunicar las dos aplicaciones, estos cambios no afectan a la arquitectura global del sistema, sino únicamente a la arquitectura interna de cada cliente. Ni los demás clientes en la DHT ni el sistema global se ven afectados por ello, ya que los clientes siguen interactuando con el resto de participantes de la misma forma.

5.2. Registro de actividad (logging)

La aplicación de lógica de negocio guarda un *log* de nombre colatex.log con la actividad que lleva a cabo, la cual es muy útil especialmente cuando se producen errores, ya que se incluye el *stacktrace* de las llamadas, entre otros datos relevantes.

El *logger* está basado en log4j y es configurable de forma estándar mediante el fichero *log4j.properties*, el cual permite ajustar el formato y canales de salida, así como el nivel (DEBUG, INFO, ...), capacidad, etc.

A continuación se muestra un ejemplo de la información registrada en el fichero:

```
2009-04-13 04:53:33,194 [main] INFO  colatex.Main  - Colatex iniciado.
2009-04-13 04:53:33,197 [main] DEBUG colatex.Main  - PastryKernel init.
2009-04-13 04:53:33,785 [main] DEBUG colatex.Main  - Obteniendo conexion Pastry.
2009-04-13 04:53:33,785 [main] DEBUG colatex.Main  - Creando boot node.
2009-04-13 04:53:36,789 [main] DEBUG colatex.Main  - Publicando servicios RMI.
2009-04-13 04:53:37,063 [main] INFO  colatex.Main  - Colatex durmiendo...
2009-04-13 04:53:50,190 [RMI]  DEBUG colatex.Main  - Obteniendo proyectos.
2009-04-13 04:53:51,494 [RMI]  DEBUG colatex.Main  - Se han obtenido 6 proyectos.
...
```

5.3. Control de errores

Todas las respuestas dadas por cualquiera de los servlets de la aplicación web incluyen siempre una sección XML de nombre *status* donde se indica lo siguiente:

- **code**: código numérico asociado al fallo. El código de valor 0 indica que la operación se completó correctamente.
- **codeDesc**: literal asociado al código anterior, que facilita su identificación rápida por un humano.
- **message**: mensaje que describe el error.

Actualmente están previstos los siguientes errores:

- **OK (0)**: sin errores, operación completada correctamente.
- **ERR_EXCEPCION_GENERICA (1)**: se produjo una excepción no controlada durante la ejecución del módulo de negocio.
- **ERR_NOMBRE_VACIO (2)**: no se especificó el nombre del proyecto en una operación que lo requiere obligatoriamente.
- **ERR_PROYECTO_YA_EXISTE (3)**: el proyecto especificado ya existe.
- **ERR_PROYECTO_NO_EXISTE (4)**: el proyecto especificado no existe.
- **ERR_PROYECTO_YA_ABIERTO (5)**: se ha intentado abrir un proyecto que ya está abierto.

- **ERR_PROYECTO_CERRADO (6)**: no se puede realizar la operación sobre un proyecto cerrado.
- **ERR_DIRECTORIO_YA_EXISTE (7)**: el directorio especificado ya existe.
- **ERR_DIRECTORIO_NO_EXISTE (8)**: el directorio especificado no existe.
- **ERR_FICHERO_YA_EXISTE (9)**: el fichero especificado ya existe.
- **ERR_FICHERO_NO_EXISTE (10)**: el fichero especificado no existe.
- **ERR_NOMBRE_NO_VALIDO (11)**: el nombre especificado no es válido, ya que contiene caracteres o secuencias prohibidas en nombres de fichero y directorios como, por ejemplo, «*», «..», ...

Por ejemplo, esta es la respuesta que se obtiene al solicitar la lista de proyectos dados de alta en CoLaTeX cuando la aplicación de lógica de negocio no se está ejecutando:

```
<?xml version="1.0"?>
<projects>
    <status>
        <code>1</code>
        <codeDesc>ERR_EXCEPCION_GENERICA</codeDesc>
        <message><![CDATA[Connection refused to host: localhost; nested exception is:
         java.net.ConnectException: Connection refused]]</message>
    </status>
</projects>
```

5.4. Consistencia de datos bajo edición concurrente

En esta sección se va a explicar cómo se asegura la consistencia de los datos aún cuando la edición la realizan concurrentemente varios usuarios. Se mostrarán ciertas partes claves del código fuente para ayudar a entenderlo.

Si se analizan los atributos y métodos del objeto InfoFichero, se puede ver que no hay ninguno que haga referencia al texto que almacena el fichero.

Esto es así porque InfoFichero es simplemente una clase contenedora de información, en la que se pretende evitar que ejecute operaciones. Los datos binarios son una característica propia del objeto, así que los puede almacenar directamente.

Sin embargo, el texto del fichero no es exclusivo de una instancia concreta de InfoFichero de un determinado usuario, sino que es el resultado de la edición concurrente de todos ellos. Por lo tanto, en cuanto al contenido de texto, los objetos InfoFichero se utilizan únicamente como referencia.

La gestión del texto almacenado en los ficheros la realiza la clase especializada *CtrlDatosFichero*, la cual se construye con una referencia al fichero sobre la cual opera, y permite obtener y editar el contenido del fichero fácilmente.

La clase CtrlDatosFichero tiene solamente tres métodos públicos, que son:

- public void borrar() throws DHTException;
- public void editar(String nuevoContenido) throws Exception;
- public String getData() throws Exception;

Estos métodos que permiten borrar, editar y obtener los datos de texto del fichero encapsulan los mecanismos de consistencia y la interacción de varias clases del algoritmo WOOT involucradas en ellos, como Patch, WootEngine, WootOp, WootPage, etc. Técnicamente, la solución consistente en ofrece una interfaz más sencilla que se encargue de encapsular, gestionar y ocultar una interacción compleja entre varios elementos se conoce como *facade pattern* [7].

Para poder explicar convenientemente el mecanismo de consistencia, es necesario describir algunos conceptos propios del algoritmo WOOT.

WootEngine (motor): se trata del motor principal del algoritmo WOOT, el cual se encarga de almacenar y devolver las páginas que gestiona.

WootPage (página): una porción de texto gestionada por el algoritmo WOOT, el contenido de la cual se puede consultar y editar. Aunque se pueden crear independientemente, en el funcionamiento normal del algoritmo las páginas se han de almacenar y obtener del motor.

WootEditor (editor): es una clase especializada en la edición de páginas del motor. Permite obtener un parche asociado a la edición (ver punto siguiente).

Patch (parche): representa una modificación en una página, indicando qué elementos son nuevos y cuales se han eliminado. Solo contienen información de los *cambios*, con lo cual aquellas partes del texto que no hayan cambiado no contribuirán al parche

WootOp (operación): son los componentes de los parches e indican una acción (*ins*: añadir, *del*: borrar).

deliverPatch (enviar un parche): método del motor WOOT que sirve para solicitar la aplicación de un parche sobre alguna de las páginas que gestiona. El motor extrae las operaciones que contiene el parche y acorde con ellas va modificando el texto contenido en la página.

Los parches tienen dos propiedades muy importantes, que son:

1. un documento de texto se puede recuperar a partir del conjunto de todos sus parches.
2. el documento recuperado es invariante respecto al orden de aplicación de los parches [3].

En CoLaTeX, la solución adoptada se aprovecha de estas dos propiedades y considera los ficheros como un conjunto de patches que se pueden aplicar en cualquier orden.

Gracias a esto, es posible extraer los parches correspondientes de la edición de cada usuario y ponerlos en común. Luego, para recuperar el contenido final del fichero, basta con aplicar el conjunto de parches al motor de cada usuario.

En las secciones 6.3.1 y 6.3.2 se explica exactamente cómo la clase CtrlDatosFichero gestiona la consistencia del contenido del texto de los ficheros, mediante las clases del modelo WOOT. Concretamente se detalla de qué manera se utilizan estas clases cuando el usuario solicita el contenido del texto del fichero, luego lo edita y finalmente guarda los cambios.

6. Fase de implementación

En esta fase, posterior al diseño, se define la forma de las interfaces necesarias para interaccionar con la aplicación desde el exterior (aplicación web, servlets, conexión RMI, ...), las clases que las implementan, las que tendrán el rol de contenedores de información (InfoFichero, InfoDirectorio, InfoProyecto, ...) y las que se encargarán de ciertas funciones especializadas (como el control de consistencia en la edición concurrente con la clase *CtrlDatosFichero*).

6.1. Gestión de proyectos

La gestión de proyectos se refiere a aquellas operaciones del núcleo que tienen que ver con los proyectos dados de alta en CoLaTeX como, por ejemplo, la creación de nuevos proyectos o la comprobacion de su existencia.

La ejecución de estas operaciones las lleva a cabo la clase de control *ProyectosImpl*, la cual implementa la interfaz *Proyectos*:

```
public interface Proyectos extends java.rmi.Remote {
    boolean addProyecto(InfoProyecto proyecto) throws DHTException, RemoteException;
    Vector<InfoProyecto> getProyectos() throws DHTException, RemoteException;
    boolean existeProyecto(String nombre) throws DHTException, RemoteException;
    InfoProyecto getProyecto(String nombre) throws DHTException, RemoteException;
    boolean deleteProyecto(String nombre) throws DHTException, RemoteException;
    boolean updateProyecto(InfoProyecto datos) throws DHTException, RemoteException;
}
```

La clase de control abstrae al programador de los accesos a la DHT, y le permiten ejecutar un conjunto de operaciones relativas al control de los proyectos de CoLaTeX. De forma transparente, acceden al contexto *p2p://colatex/proyectos*, donde se guarda un vector de objetos InfoProyecto (información de cada uno de los proyectos).

Las operaciones disponibles en esta clase de control son:

- **addProyecto**: añade un nuevo proyecto a CoLaTeX.
- **getProyectos**: devuelve un vector con todos los proyectos dados de alta.

- **existeProyecto**: indica si el proyecto especificado mediante su nombre está dado de alta en CoLaTeX.
- **getProyecto**: obtiene el proyecto cuyo nombre coincide con el especificado, o *null* si no se encuentra.
- **deleteProyecto**: borra el proyecto especificado mediante su nombre.
- **updateProyecto**: actualiza los datos del proyecto especificado mediante su nombre (por ejemplo, su descripción).

6.1.1. Clase InfoProyecto

La clase InfoProyecto es una clase de datos que representa un proyecto de CoLaTeX, y contiene los siguientes atributos:

- **nombre**: el nombre del proyecto. Es un atributo de texto que identifica cada proyecto de forma unívoca.
- **descripción**: una cadena de texto que describe adecuadamente el proyecto.
- **dirTemporal**: una cadena de texto que identifica el directorio en el cual se debe descargar el proyecto para compilarlo en el ordenador del usuario local y generar el documento final.
- **comandoCompilacion**: la orden que se debe ejecutar en el directorio temporal para generar el documento final. Por ejemplo: pdflatex freedom.latex

6.1.2. Interacción desde el exterior

Las operaciones para la gestión de proyectos son públicas en la clase de control *ProyectosImpl*, la cual se instancia en el núcleo. Por lo tanto, es necesario algún mecanismo para poder acceder a esta clase desde el exterior.

Para hacer posible la interacción desde el exterior, se han previsto cuatro formas diferentes:

- Invocando directamente las operaciones del servlet *proyectos* mediante el protocolo HTTP y decodificando las respuestas XML.

- Utilizando una aplicación web que realice de forma transparente la invocación de las llamadas del servlet *proyectos*. Como parte de este PFC se entrega una aplicación web completa que utiliza Ajax como mecanismo de comunicación asíncrona en la invocación de las operaciones de los servlet.
- Invocando las operaciones de la clase de control *ProyectosImpl* mediante el webservice correspondiente.
- Utilizando la interfaz *Proyectos* e instanciando la clase de control remota mediante RMI.

En previsión de ampliar CoLaTeX con nuevas formas de interacción con el exterior, se ha hecho que los servlet no ejecuten las operaciones directamente, sino que las deleguen en un objeto de tipo *Executor*.

De esta forma, si en el futuro se diseñan nuevas formas de comunicacioń con el exterior, bastará que utilicen el objeto de tipo Executor para delegar sobre el la ejecución de las operaciones y leer las respuestas XML.

Concretamente, para el control de proyectos se utiliza la subclase *ProyectosExecutor*. En la figura (23) podemos ver el esquema de la relación entre los diferentes elementos involucrados en la interacción.

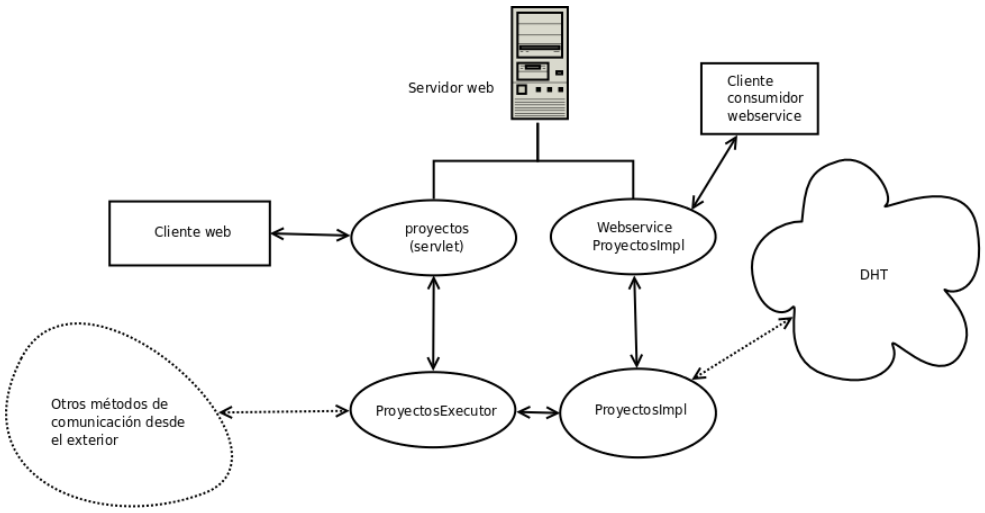

Figura 23: diagrama de interacción en la gestión de proyectos.

Desde el servidor web la aplicación accede a los servlets, los cuales utilizan a *Proyec-*

tosExecutor y para su procesamiento y éstas a su vez internamente hacen uso de la clase ProyectosImpl, la cual finalmente realiza la gestión de las lecturas y escrituras en la DHT.

Si se utiliza un cliente de webservice, se accede al servicio web en el servidor, que a su vez utiliza internamente la clase ProyectosImpl para ofrecer el servicio.

Finalmente, el cliente también puede acceder directamente al núcleo mediante llamadas RMI, con lo cual accede remotamente a la clase *ProyectosImpl*.

A continuación se describirán las operaciones *get*, *add*, *delete* y *update* que están disponibles en *ProyectosExecutor*. Se explicarán referidas al servlet, ya que el acceso mediante la interficie web es el método más intuitivo, aunque sería muy similar para otros métodos de acceso.

La aplicación web utiliza un servlet situado en */servlets/proyectos* que obtiene órdenes en la cadena de parámetros GET y devuelve los resultados en formato XML.

6.1.3. Operación get

Devuelve una lista con todos los proyectos dados de alta. Si se especifica además el parámetro *name*, solamente devolverá información para ese proyecto.

Ejemplo: petición para obtener todos los proyectos dados de alta y respuesta del servet:

http://localhost:8080/colatexWeb/servlets/proyectos?action=get

```
<?xml version="1.0"?>
<projects>
   <project>
       <name>Variable compleja y aplicaciones</name>
       <description>Ensayo sobre matemáticas: análisis complejo</description>
       <tempDir>/tmp/variable/</tempDir>
       <compCommand>pdflatex variable.latex</compCommand>
   </project>
   <project>
       <name>Novela colaborativa</name>
       <description>Primer concurso de novela colaborativa</description>
       <tempDir>/tmp/novela/</tempDir>
       <compCommand>pdflatex novelaConcurso.latex</compCommand>
```

```
   </project>
   <status>
      <code>0</code>
      <codeDesc>OK</codeDesc>
      <message><![CDATA[Ok]]</message>
   </status>
</projects>
```

En el caso de que se solicite un proyecto que no existe, no se devolverá ningún error, sino una lista vacía.

http://localhost:8080/colatexWeb/servlets/proyectos?action=get&name=noExiste

```
<?xml version="1.0"?>
<projects>
<status>
   <code>0</code>
   <codeDesc>OK</codeDesc>
   <message><![CDATA[Ok]]</message>
</status>
</projects>
```

6.1.4. Operación add

Crea un nuevo proyecto. La descripción se obtiene del parámetro obligatorio *description*.

Ejemplo: petición de adición de un nuevo proyecto de nombre *arboria*, y respuesta del servlet.

http://localhost:8080/colatexWeb/servlets/proyectos?action=add&name=arboria&

description=El mundo de las plantas salvajes

```
<?xml version="1.0"?>
<status>
   <code>0</code>
   <codeDesc>OK</codeDesc>
```

```
   <message><![CDATA[Ok, new proyect created: arboria]]</message>
</status>
```

Ejemplo de error: intentar dar de alta un proyecto que ya existe.

http://localhost:8080/colatexWeb/servlets/proyectos?action=add&name=arboria

```
<?xml version="1.0"?>
<status>
   <code>3</code>
   <codeDesc>ERR_PROYECTO_YA_EXISTE</codeDesc>
   <message><![CDATA[Proyect already exists: arboria]]</message>
</status>
```

6.1.5. Operación delete

Borra el proyecto cuyo nombre está especificado en el parámetro obligatorio *name*.

Ejemplo: petición de borrado del proyecto *arboria*, y respuesta del servlet.

http://localhost:8080/colatexWeb/servlets/proyectos?action=delete&name=arboria

```
<?xml version="1.0"?>
<status>
   <code>0</code>
   <codeDesc>OK</codeDesc>
   <message><![CDATA[Ok, proyect deleted: arboria]]</message>
</status>
```

Ejemplo de error: intentar borrar un proyecto que no existe.

http://localhost:8080/colatexWeb/servlets/proyectos?action=delete&name=arboria

```
<?xml version="1.0"?>
<status>
   <code>4</code>
   <codeDesc>ERR_PROYECTO_NO_EXISTE</codeDesc>
   <message><![CDATA[Unknown proyect: arboria]]</message>
</status>
```

6.1.6. Operación update

Actualiza los datos del proyecto cuyo nombre está especificado en el parámetro obligatorio *name*. Es posible

Ejemplo: petición de actualización de la descripción a *Canciones de piratas* en el proyecto *arboria* y respuesta del servlet.

http://localhost:8080/colatexWeb/servlets/proyectos?action=update&name=arboria&

description=Canciones de piratas

```
<?xml version="1.0"?>
<status>
   <code>0</code>
   <codeDesc>OK</codeDesc>
   <message><![CDATA[Ok, proyect updated: arboria]]</message>
</status>
```

Ejemplo de error: intentar actualizar un proyecto que no existe.

http://localhost:8080/colatexWeb/servlets/proyectos?action=update&name=noExiste&

description=Canciones de piratas

```
<?xml version="1.0"?>
<status>
   <code>4</code>
   <codeDesc>ERR_PROYECTO_NO_EXISTE</codeDesc>
   <message><![CDATA[Unknown proyect: noExiste]]</message>
</status>
```

6.2. Gestión del proyecto LaTeX

Una vez que un proyecto ha sido dado de alta, es necesario disponer de mecanismos que permitan gestionar el contenido del proyecto en sí mismo. Son necesarias operaciones como, por ejemplo, añadir y borrar ficheros binarios y de texto, añadir y borrar directorios, obtener listados de ficheros y directorios, etc.

La gestión del proyecto LaTeX se realiza en el núcleo mediante la clase del núcleo *ProyectoLatexImpl*, la cual implementa la interfaz *ProyectoLatex*:

```
public interface ProyectoLatex extends java.rmi.Remote {
    int addDirectorio(String nombre)
        throws DHTException, RemoteException;
    int addFichero(InfoFichero fichero)
        throws DHTException, RemoteException;
    int borrarSubDirectorio(InfoDirectorio directorio)
        throws DHTException, RemoteException;
    int borrarSubDirectorio(String subDirectorio)
        throws DHTException, RemoteException;
    int borrarFichero(InfoFichero fichero)
        throws DHTException, RemoteException;
    InfoFichero getFichero(String pathAndFilename)
        throws DHTException, RemoteException;
    String getTextoFichero(InfoFichero fichero)
        throws Exception, RemoteException;
    void editarFichero(InfoFichero fichero, String nuevoContenido)
        throws Exception, RemoteException;
    int compilarProyecto()
        throws Exception, RemoteException;
    Vector<String> getListaSubDirectorios()
        throws DHTException, RemoteException;
    Vector<String> getListaFicheros()
        throws DHTException, RemoteException;
    ListaDirectoriosYFicheros getListaSubDirectoriosYFicheros()
        throws DHTException, RemoteException;
    boolean isOpen()
        throws RemoteException;
    int open(String proyecto)
        throws RemoteException;
    void close()
        throws RemoteException;
    String getProyectName()
        throws RemoteException;
    int actualizarFicheroEnDHT(InfoFichero fichero)
```

```
        throws DHTException, RemoteException;
    InfoDirectorio getDirectorioActual()
        throws RemoteException;
    int chDir(InfoDirectorio directorio)
        throws DHTException, RemoteException;
    int chDir(String subDirectorio)
        throws DHTException, RemoteException;
}
```

Las operaciones disponibles en esta clase de control son:

- **addDirectorio**: añade un nuevo directorio al proyecto, dentro del directorio actual.
- **addFichero**: añade el fichero especificado dentro del directorio actual.
- **borrarSubDirectorio**: borra un subdirectorio del proyecto y todos los ficheros y subdirectorios que contiene.
- **borrarFichero**: elimina del proyecto el fichero especificado.
- **getFichero**: obtiene un objeto del tipo InfoFichero con todos los datos del fichero especificado a partir de su path completo (por ejemplo, */capitulo2/resumen/info.latex*).
- **getTextoFichero**: dado el path completo de un fichero de texto, devuelve el texto que contiene en ese momento.
- **editarFichero**: dado un fichero identificado por su objeto InfoFichero permite editar su contenido especificando cual es el resultado de la edición del usuario. Debido a los mecanismos de consistencia, el contenido final del fichero no será el especificado, sino que contendrá la contribución de todas las ediciones concurrentes.
- **compilarProyecto**: guarda en el directorio temporal configurado para ese proyecto sus los ficheros y directorios y ejecuta la orden que se configuró para la compilación. La compilación se realiza de forma local en el directorio temporal del usuario.
- **getListaSubDirectorios**: obtiene una lista con los subdirectorios del directorio actual. Solamente se considera un nivel, es decir, se listan los subdirectorios cuyo padre es el directorio actual.

- **getListaFicheros**: obtiene una lista de los ficheros contenidos en el directorio actual.
- **isOpen**: indica si el proyecto actual está abierto, o no.
- **open**: abre el proyecto actual.
- **close**: cierra el proyecto actual.
- **getProyectName**: obtiene el nombre del proyecto abierto actualmente.
- **actualizarFicheroEnDHT**: dado un objeto InfoFichero que representa un determinado fichero, solicita que se haga persistente en la DHT. Esta funcionalidad existe como función explícita porque los objetos de tipo InfoFichero son simplemente contenedores de información para el intercambio de información[34] y los cambios que se efectúan en ellos mediante sus *setters* no son persistentes. Lo mismo ocurre con los objetos de tipo InfoDirectorio.
- **getDirectorioActual**: obtiene el path completo del directorio actual (por ejemplo, */capitulo2/resumen/*).
- **chDir**: cambia el directorio actual al especificado. Si el parámetro es de tipo String, se entiende que el cambio es a un directorio localizado dentro del directorio actual, así que solo se especifica su nombre. Si el parámetro es del tipo InfoDirectorio, se entiende que el cambio es a cualquier subdirectorio del proyecto.

A continuación se explicarán brevemente las clases InfoDirectorio, InfoFichero y ListaDirectoriosYFicheros, involucradas en la gestión del proyecto LaTeX.

6.2.1. Clase InfoDirectorio

Esta clase representa un directorio del proyecto LaTeX, el cual puede contener ficheros y otros directorios. Los ficheros (clase InfoFichero) guardan una referencia del tipo InfoDirectorio indicando a qué directorio pertenecen.

6.2.2. Clase InfoFichero

InfoFichero representa un fichero del proyecto LaTeX, el cual está dentro del un determinado directorio. Consta de los siguientes atributos:

[34] Este tipo de objetos también son conocidos como *POJO*s (Plain Old Java Object).

- **nombre**: el nombre del fichero, sin información del directorio. Por ejemplo, *fichero.latex*.
- **directorio**: referencia del tipo InfoDirectorio que indica el directorio en el cual reside el fichero.
- **binario**: indica si el fichero contiene datos binarios, o por el contrario, es de texto.
- **dataBin**: array de bytes con el contenido del fichero binario. Si es de texto, su valor será *null*.

Es importante hacer notar que cuando el fichero es binario, los datos están almacenados directamente en el atributo **dataBin** del objeto, pero no existe ningún campo para almacenar el contenido cuando el fichero es de texto.

Para garantizar la consistencia de los datos en la edición concurrente, los ficheros no guardan los datos de texto, sino que sirven como referencia a una clase de control especializada en recuperar y almacenar la información textual en la DHT, de una forma consistente. El funcionamiento de esta clase (*CtrlDatosFichero*) y los mecanismos de consistencia se explican detalladamente en la sección 5.4.

6.2.3. Clase ListaDirectoriosYFicheros

ListaDirectoriosYFicheros encapsula el contenido (ficheros y subdirectorios) de un directorio del proyecto.

Esta estructura es devuelva por *getListaSubDirectoriosYFicheros()* cada vez que el usuario cambia el directorio de trabajo.

6.2.4. Interacción desde el exterior

Al igual que ocurre en el caso de la gestión de proyectos, también es necesario disponer de una clase de control en el núcleo para el control del contenido de los proyectos LaTeX, accesible desde el exterior. Esta clase de control es *ProyectosImpl*.

Las cuatro formas de interacción con el exterior son las que se explican en la sección 6.1.2. De igual forma, también se hace uso de un objeto de clase clase *Executor*.

En previsión de ampliar CoLaTeX con nuevas formas de interacción con el exterior, se ha hecho que los servlet no ejecuten las operaciones directamente, sino que las deleguen en

un objeto de cuyo tipo es una subclase *Executor*. En concreto, es del tipo *ProyectoLatex-Executor*.

Como ya se explicó, se utilizan estos objetos de tipo *Executor* es previsión de que en el futuro se añadan nuevas formas de interacción con el exterior, de manera que estos nuevos mecanismos puedan delegar el funcionamiento sobre los objetos *Executor* e implementen solamente la comunicación.

Podemos ver esquematizada la relación entre los diferentes elementos en la figura (24). La estructura es muy similar al caso de la gestión de proyectos que se vio en la sección 6.1.2.

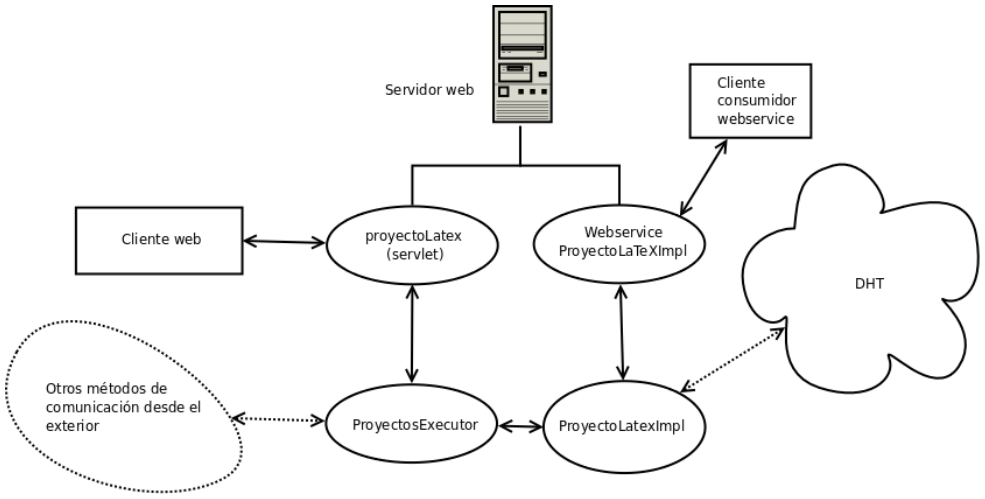

Figura 24: diagrama de interacción en la gestión de proyectos LaTeX.

A continuación se describirán las operaciones *open*, *close*, *getName*, *getCurrentDir*, *chDir*, *addSubDir*, *deleteFile*, *deleteSubDir*, *getFiles*, *getSubDirs* y *getDirsAndFiles* que están disponibles en *ProyectoLatexExecutor*. Se explicarán referidas al servlet, ya que el acceso mediante la interficie web es el método más intuitivo, aunque sería exactamente igual para cualquier otro métodos de acceso que utilice *ProyectoLatexExecutor*.

La aplicación web utiliza un servlet situado en */servlets/proyectoLatex* que obtiene órdenes en la cadena de parámetros GET y devuelve los resultados en formato XML.

Este servlet es el encargado de gestionar todas las acciones del usuarios relativas a un proyecto en concreto como, por ejemplo, la creación de nuevos ficheros o directorios, cambiar del directorio de trabajo, etc.

De igual forma que el servlet *proyectos*, recibe los parámetros mediante el protocolo

GET y su respuesta está codificada en XML, incluyendo siempre información de estado de la ejecución de la operación.

La aplicación invoca las operaciones del servlet, recoge la respuesta, comprueba si se han producido errores, y finalmente la hace visible en la página de edición, dinámicamente mediante Ajax.

Las órdenes a las que puede responder el servlet (o en general, a las que puede responder *ProyectoLatexExecutor*) son *open*, *openAndRedirect*, *close*, *getName*, *addSubDir*, *addFile*, *deleteFile*, *deleteDir*, *chDir*, *getCurrentDir*, *getFiles*, *getDirsAndFiles*, *getSubDirs*, *getFileText*, *generate*, *editFileText* y *uploadFile*.

6.2.5. Operación open

Abre el proyecto especificado para editar sus ficheros y directorios.

Ejemplo: petición para abrir el proyecto *arboria*, y respuesta del servlet.

http://localhost:8080/colatexWeb/servlets/proyectos?action=open&name=arboria

```
<?xml version="1.0"?>
<status>
    <code>0</code>
    <codeDesc>OK</codeDesc>
    <message><![CDATA[Ok]]</message>
</status>
```

6.2.6. Operación openAndRedirect

Es una operación exclusiva para aquellos métodos basados en web, como por ejemplo el acceso mediante un servlet que utiliza la aplicación web.

Comprueba si el proyecto especificado está abierto y realiza alguna de estas acciones:

- Si no se especificó ningún proyecto, redirige la petición a la página elegirProyecto.jsp, para que el usuario pueda elegir alguno.
- Si el proyecto especificado está abierto, redirige la petición a la página editar.jsp.
- Si el proyecto especificado está cerrado, lo abre y redirige la petición a la página editar.jsp.

6.2.7. Operación close

Cierra el proyecto actual, si es que está abierto.

Ejemplo: petición para cerrar el proyecto actual, y respuesta del servlet.

```
<?xml version="1.0"?>
<status>
    <code>0</code>
    <codeDesc>OK</codeDesc>
    <message><![CDATA[Ok]]</message>
</status>
```

6.2.8. Operación getName

Devuelve el nombre del proyecto LaTeX sobre el cual se está trabajando actualmente.

Ejemplo: petición para obtener el nombre del proyecto actual, y respuesta del servlet.

http://localhost:8080/colatexWeb/servlets/proyectos?action=get

```
<?xml version="1.0"?>
<project>
    <name>Novela colaborativa</name>
    <status>
        <code>0</code>
        <codeDesc>OK</codeDesc>
        <message><![CDATA[Ok]]</message>
    </status>
</project>
```

6.2.9. Operación addSubDir

Crea un nuevo directorio, dentro del actual.

Ejemplo: petición para crear el directorio *mercury*, y respuesta del servlet (un error, debido a que el directorio ya existe).

http://localhost:8080/colatexWeb/servlets/proyectos?action=addSubDir&name=mercury

```
<?xml version="1.0"?>
<status>
    <code>7</code>
    <codeDesc>ERR_DIRECTORIO_YA_EXISTE</codeDesc>
    <message><![CDATA[No se pudo crear el subdirectorio mercury]]</message>
</status>
```

6.2.10. Operación addFile

Crea un nuevo fichero dentro del directorio actual

Ejemplo: creación del fichero de texto *informe.latex* y respuesta del servlet (un error, debido a que el fichero ya existe).

http://localhost:8080/colatexWeb/servlets/proyectos?

action=addFile&name=informe.latex&bin=0

```
<?xml version="1.0"?>
<status>
    <code>9</code>
    <codeDesc>ERR_FICHERO_YA_EXISTE</codeDesc>
    <message><![CDATA[No se pudo crear el fichero local informe.latex]]</message>
</status>
```

6.2.11. Operación deleteFile

Borra el fichero especificado a partir de su path completo.

Ejemplo: borrar el fichero */incidencias/seguridad/informe.latex* y respuesta del servlet (un error, debido a que el fichero no existe).

http://localhost:8080/colatexWeb/servlets/proyectos?action=deleteFile&

path=/incidencias/seguridad/informe.latex

```
<?xml version="1.0"?>
<status>
    <code>10</code>
    <codeDesc>ERR_FICHERO_NO_EXISTE</codeDesc>
```

```
    <message><![CDATA[No se pudo borrar el fichero
/incidencias/seguridad/informe.latex]]</message>
</status>
```

6.2.12. Operación deleteDir

Elimina el subdirectorio especificado mediante su path completo. El subdirectorio se entiende localizado dentro del directorio actual.

Todos los subdirectorios que cuelguen directa o indirectamente del subdirectorio también se borrarán. Si el directorio especificado es el actual, se producirá un cambio de directorio a la raíz (/) despúes del borrado.

Ejemplo: petición para eliminar el directorio *musica/rock/* supuestamente localizado en el directorio actual, y respuesta del servlet (un error, debido a que el subdirectorio no existe dentro del directorioa actual).

http://localhost:8080/colatexWeb/servlets/proyectos?action=deleteSubDir&path=musica/rock/

```
<?xml version="1.0"?>
<status>
    <code>8</code>
    <codeDesc>ERR_DIRECTORIO_NO_EXISTE</codeDesc>
    <message><![CDATA[No se pudo borrar el directorio musica/rock/]]</message>
</status>
```

6.2.13. Operación chDir

Cambia el directorio de trabajo. Está permitido especificar el path completo (por ejemplo, */autores/miguel/opinion/*, y también solamente el nombre de un directorio (se entenderá que está dentro del directorio de trabajo actual). El sistema es capaz de reconocer automáticamente entre los dos casos.

Para regresar al directorio inmediatamente superior respecto al actual, se puede especificar la cadena "..". Si el directorio de trabajo ya era la raíz (/), entonces la operación no tiene ningún efecto.

Ejemplo: petición para cambiar el directorio de trabajo a *opinion*, y respuesta del servlet (un error, debido a que el directorio destino no existe).

http://localhost:8080/colatexWeb/servlets/proyectos?action=chDir&path=opinion

```
<?xml version="1.0"?>
<status>
    <code>8</code>
    <codeDesc>ERR_DIRECTORIO_NO_EXISTE</codeDesc>
    <message><![CDATA[No se pudo cambiar al directorio opinion]]></message>
</status>
```

6.2.14. Operación getCurrentDir

Obtiene el directorio de trabajo actual.

Las operaciones que afectan a directorios y ficheros no requieren que se especifique el path completo hasta el recurso al cual se refieren, sino que permiten que el usuario especifique únicamente el nombre del fichero o directorio.

El path que se utilizará dentro de la estructura de directorios del proyecto a la hora de efectuar la operación (borrar fichero, crear directorio, etc) se entiende referido al directorio de trabajo.

Ejemplo: petición para obtener el directorio de trabajo actual, y respuesta del servlet.

http://localhost:8080/colatexWeb/servlets/proyectos?action=getCurrentDir

```
<?xml version="1.0"?>
<directory>
    <path>/resumen/criticas/</path>
    <status>
        <code>0</code>
        <codeDesc>OK</codeDesc>
        <message><![CDATA[Ok]]></message>
    </status>
</directory>
```

6.2.15. Operación getFiles

Obtiene una lista de los ficheros que residen en el directorio actual.

Para cada fichero se devuelve su path completo.

Ejemplo: petición para obtener los ficheros del directorio actual (/dir1/dir2/), y respuesta del servlet.

http://localhost:8080/colatexWeb/servlets/proyectos?action=getFiles

```
<?xml version="1.0"?>
<files>
    <file>/dir1/dir2/documento.latex</file>
    <file>/dir1/dir2/imagen.png</file>
    <file>/dir1/dir2/bibliografia.bib</file>
    <status>
        <code>0</code>
        <codeDesc>OK</codeDesc>
        <message><![CDATA[Ok]]</message>
    </status>
</files>
```

6.2.16. Operación getDirsAndFiles

Obtiene una lista de ficheros y subdirectorios que residen en el directorio actual.

La información que aporta esta operación es la unión de las operaciones getFiles y getSubDirs. El motivo por el cual existe como una función aparte es porque obtener todos los ficheros y subdirectorios del directorio actual es una operación muy utilizada, por lo que con una única llamada a esta función es posible conocerlos.

Ejemplo: petición para obtener los ficheros y subdirectorios del directorio actual (/mates/riemann-z/), y respuesta del servlet.

http://localhost:8080/colatexWeb/servlets/proyectos?action=getDirsAndFiles

```
<?xml version="1.0"?>
<DirsAndFiles>
    <subdirectories>
        <path>/mates/riemann-z/abstract/</file>
        <path>/mates/riemann-z/introduccion/</file>
        <path>/mates/riemann-z/estudio/</file>
```

```
        <path>/mates/riemann-z/conclusiones/</file>
    </subdirectories>
    <files>
        <file>/mates/riemann-z/inicio.latex</file>
        <file>/mates/riemann-z/portada.jpg</file>
        <file>/mates/riemann-z/biblio.bib</file>
    </files>
    <status>
        <code>0</code>
        <codeDesc>OK</codeDesc>
        <message><![CDATA[Ok]]</message>
    </status>
</DirsAndFiles>
```

6.2.17. Operación getSubDirs

Obtiene una lista de los subdirectorios que residen en el directorio actual.

Para cada directorio se devuelve su path completo.

Ejemplo: petición para obtener los subdirectorios del directorio actual (/mates/riemann-z/), y respuesta del servlet.

http://localhost:8080/colatexWeb/servlets/proyectos?action=getSubDirs

```
<?xml version="1.0"?>
<subdirectories>
    <path>/mates/riemann-z/abstract/</file>
    <path>/mates/riemann-z/introduccion/</file>
    <path>/mates/riemann-z/estudio/</file>
    <path>/mates/riemann-z/conclusiones/</file>
    <status>
        <code>0</code>
        <codeDesc>OK</codeDesc>
        <message><![CDATA[Ok]]</message>
    </status>
</subdirectories>
```

6.2.18. Operación getFileText

Obtiene el texto actual del fichero de texto del cual se indica el path completo.

El texto que se obtiene es el resultado de haber conciliado todos los cambios que todos y cada uno de los usuarios han ido realizando concurrentemente sobre ese fichero. En la sección 5.4 se explica exactamente cómo funciona el mecanismo de consistencia basado en el algoritmo WOOT.

Ejemplo: obtención del texto actual para el fichero cuyo path es /informe.latex.

http://localhost:8080/colatexWeb/servlets/proyectoLatex?

action=getFileText&path=/informe.latex

```
<?xml version="1.0"?>
<fileText>
    <text><![CDATA[\documentclass{article}
\title{Informe hola mundo}
\author{Miguel Colom}
\date{Fecha actual}
\begin{document}
\maketitle
Hola mundo!
\end{document}
]]></text>
    <status>
       <code>0</code>
       <codeDesc>OK</codeDesc>
       <message><![CDATA[Ok</message>
    </status>
</fileText>
```

Como se puede observar, se protege el texto devuelto dentro de un campo CDATA.

6.2.19. Operación generate

Compila el proyecto LaTeX abierto. Para hacerlo, descarga todos los ficheros y directorios del proyecto en el directorio temporal de ese proyecto y luego ejecuta la orden de

compilación. Tanto el directorio temporal como la orden de compilación se configuran en las propiedades del proyecto.

Ejemplo: generación del fichero PDF final.

http://localhost:8080/colatexWeb/servlets/proyectoLatex?action=generate

```
<?xml version="1.0"?>
<status>
   <code>0</code>
   <codeDesc>OK</codeDesc>
   <message><![CDATA[Ok]]></message>
</status>
```

6.2.20. Operación editFileText

Edita los datos de texto del fichero especificado mediante su path completo.

Esta operación es exclusiva de servlets u otros métodos capaces de enviar datos mediante el protocolo POST de HTTP.

Para realizar la operación se ha de especificar en el parámetro *path* de la cadena GET la ruta completa del fichero y en los datos POST la cadena de texto que resulta de la edición del fichero por parte del usuario.

Los mecanismos de consistencia extraerán las aportaciones del usuario y las tendrán en cuenta a la hora de generar el contenido final del fichero. Si la operación finaliza correctamente, se enviará la siguiente respuesta:

```
<?xml version="1.0"?>
<status>
   <code>0</code>
   <codeDesc>OK</codeDesc>
   <message><![CDATA[Ok]]></message>
</status>
```

6.2.21. Operación uploadFile

Actualiza los datos binarios del fichero especificado mediante su path completo.

Al igual que *editFileText*, esta operación es exclusiva de servlets u otros métodos capaces de enviar datos mediante el protocolo POST de HTTP.

Esta operación no devuelve ninguna respuesta XML, sino que realiza una redirección hacia la página editar.jsp si la actualización del contenido del fichero se realizó correctamente.

En el parámetro *path* de la cadena GET se ha de indicar la ruta completa del fichero y en los datos POST el contenido binario del fichero.

6.3. Gestión de la consistencia con la clase *CtrlDatosFichero*

En la sección 5.4 se explicó cómo se diseñó el control de consistencia en CoLaTeX.

En esta sección se explica cómo se ha implementado mediante la clase *CtrlDatosFichero*, a partir del diseño inicial.

6.3.1. Recuperación del texto del fichero

En esta operación se recupera el contenido de texto asociado al fichero, a partir del total de todos sus parches. Estos parches son la contribución de los diferentes usuarios que lo han ido editando.

Para entenderlo mejor, se expondrá el código fuente y se irá comentando entre líneas cada parte.

```
public String getData() throws Exception {
    WootEngine myEngine = this.getEngine();

    Vector<Patch> vPatches = this.recuperarPatches();
    for (Patch p: vPatches)
        this.deliverPatch(myEngine, p);
```

En primer lugar, se obtiene una referencia al motor WOOT.

El siguiente paso consiste en recuperar de la DHT un vector conteniendo todos los parches que se han enviado hasta ese momento.

Una vez obtenido el vector, se envían los parches al motor, para que vaya componiendo el texto actual de la página.

```
    CtrlDatosFichero.vPatchesEdicion = new Vector<Patch>();
    CtrlDatosFichero.vPatchesEdicion.addAll(vPatches);
```

El contenido del vector de parches que se acaba de obtener se guarda en el vector *vPatchesEdicion*. Este último vector, miembro de la clase, representa el estado actual del fichero, es decir, el total de parches **anterior** a cualquier modificación que pueda realizar el usuario de aquí en adelante.

Es muy importante guardar esta información en el vector *vPatchesEdicion*, ya que si el fichero se edita posteriormente, será necesario recuperar el contenido de vector de parches que se ha guardado en este paso. El contenido de *vPatchesEdicion* no tiene por qué coincidir con el que se recuperaría de la DHT en el momento de la edición, ya que podría darse el caso de que otros usuarios la hubieran estado modificando.

```
    // Obtener página a partir de los patches recuperados
    WootPage page = myEngine.getWootPage(this.fichero.getPathAndFilename());
    if (page == null)
        page = new WootPage(this.fichero.getPathAndFilename(),
            new WootPageLogImplMock());

    return page.toHumanString();
}
```

Finalmente, se solicita al motor que recupere la página (su nombre coincide con el path completo del fichero) y se devuelve su contenido de texto.

6.3.2. Edición del texto de un fichero

Cuando el usuario quiere modificar el contenido de un fichero, el primer paso siempre es recuperar el texto que el fichero está almacenando en ese momento. Para ello se realiza una llamada al método *getData()* que se ha explicado en la sección 6.3.1.

Una vez que ha recuperado la cadena de texto asociada al fichero, se invoca el método *editar* de esta clase, donde se especifica como parámetro el nuevo texto que el usuario quiere establecer para el fichero.

A continuación se expondrá este método, intercalando algunas explicaciones entre líneas.

```
public void editar(String nuevoContenido) throws Exception {
    // Recuperar estado a partir de todos los patches de edición
    WootEngine myEngine = this.getEngine();
    if (CtrlDatosFichero.vPatchesEdicion == null)
        CtrlDatosFichero.vPatchesEdicion = new Vector<Patch>();

    for (Patch p: CtrlDatosFichero.vPatchesEdicion)
        this.deliverPatch(myEngine, p);
```

En primer lugar, se obtiene una referencia al motor WOOT y se le envían todos los parches del vector *vPatchesEdicion*.

El contenido de este vector se estableció al recuperar el texto del fichero, y es precisamente el estado del fichero en el momento en el que el usuario lo recuperó. Es decir, es el conjunto de patches que aplicados al motor WOOT permiten reconstruir el texto que el usuario está a punto de modificar.

Es importante hacer notar que el contenido de este vector no tiene por qué coincidir con el mismo vector almacenado en la DHT en ese momento. Si otros usuarios están realizando ediciones concurrentes, el contenido de dos vectores será diferente.

```
    // Obtener página a partir de los patches edición
    WootPage page = myEngine.getWootPage(this.fichero.getPathAndFilename());
    if (page == null)
        page = new WootPage(this.fichero.getPathAndFilename(),
            new WootPageLogImplMock());
```

El siguiente paso consiste en recuperar del motor la página resultante de aplicar el vector de parches de edición. El texto de esta página es precisamente el contenido que el usuario pretende editar.

```
    // Editar página y obtener nuevoPatch
    WootEditor myEditor = this.getEditor();
    WootSession ws = myEditor.edit(page);
    ws.setContent(nuevoContenido);
    Patch nuevoPatch = ws.save();
```

Seguidamente, se obtiene un editor WOOT, se edita la página y se establece el nuevo contenido.

Como el contenido de la página es el texto previo a la edición y se establece el contenido posterior a la modificación del usuario, tenemos que al obtener el patch, este solamente contiene las modificaciones para este usuario en concreto.

Este es el motivo por el cual se guardó el vector *vPatchesEdicion* al recuperar el texto de la página: para aislar los cambios efectuados por este usuario en su edición. De esta manera, el parche obtenido solamente contiene información de los cambios de este usuario.

```
    if (nuevoPatch != null) {
        Vector<Patch> vPatches  = this.recuperarPatches();
        vPatches.add(nuevoPatch);

        // Añadir nuevo patch y almacenar el vector de patches en la DHT
        DHTHandler dht = this.getDHTPatches();
        dht.put(this.getKeyFichero(this.fichero), vPatches);
    }
}
```

Finalmente, se recupera el vector de patches de la DHT, el cual contiene los aportados por los usuarios que han editado y enviado sus cambios mientras nuestro usuario realizaba su propia edición, y se añade el nuevo parche.

Cuando cualquier otro usuario obtenga el texto del fichero, recuperará el vector de parches de la DHT, los aplicará a su motor WOOT y la aportación de este usuario se verá reflejada junto con las de los demás participantes.

El contenido de texto de un fichero no tiene por qué coincidir con el estableció ninguno de los usuarios en sus respectivas ediciones, sino que será una combinación consistente de la aportacioń de todos y cada uno de ellos.

7. Fase de experimentación y testeo

Para comprobar el funcionamiento de CoLaTeX se ha utilizado JUnit [9].

Se ha diseñado un test de funcionamiento general y otro más específico para comprobar la edición concurrente.

7.1. Test de funcionamiento general

Antes de realizar este test, es necesario eliminar (o renombrar) el directorio *storage*, para que la aplicación pueda testear convenientemente los mecanismos de comprobación de existencia de proyectos, ficheros y directorios, partiendo de un estado conocido.

El test creará una estructura de ficheros como la siguiente e irá realizando diversas acciones, como si fuera un usuario utilizando la aplicación.

```
/ (raíz)
|-f1.jpg
|-f2.txt
|-f3.latex
|-dir1/
|-dir2/
|   |
|   |-f4.latex
|   |-f5.png
|   |-dir3/
|   |   |-f6.latex
|   |-dir4/
|   |-dir5/
|       |-f7.latex
|-dir7
```

El fichero */f1.jpg* es binario y el test realizará con el una prueba consistente en establecer su contenido binario, escribirlo en la DHT, obtenerlo de la DHT y comprobar una suma de control.

Para establecer el contenido, obtendrá los datos binarios del fichero *colatex.jpg* situado en el directorio /tmp/.

El test realiza las siguientes operaciones:

1. El proyecto Test no existe antes del test.
2. El proyecto Test no se puede actualizar (no existe).
3. El proyecto Test no se puede abrir (no existe).
4. No hay ningún proyecto abierto.
5. No se puede añadir ningún directorio (no hay ningún proyecto abierto).
6. No se puede añadir ningún fichero (no hay ningún proyecto abierto).
7. No se puede cambiar el directorio actual (no hay ningún proyecto abierto).
8. Se puede crear el proyecto Test correctamente.
9. El proyecto Test está cerrado.
10. No se puede abrir el proyecto Test2 (no existe).
11. Se puede abrir el proyecto Test.
12. Falla la apertura del proyecto Test (ya está abierto).
13. El nombre del proyecto Test es «Test».
14. El directorio actual es la raíz.
15. No hay ficheros en el directorio actual (raíz).
16. No hay subdirectorios en el directorio actual (raíz).
17. Se puede crear la jerarquía de directorios y ficheros del test.
18. El fichero f4.latex se puede editar y establecer su contenido.
19. El cambio de directorio (chDir) funciona correctamente.
20. Hay exactamente 3 ficheros en la raíz y son /f1.jpg, /f2.txt y /f3.latex.
21. Hay exactamente 3 subdirectorios en la raíz y son /dir1/, /dir2/ y /dir7/.
22. El directorio /dir2/dir3/ no contiene subdirectorios.

23. El directorio /dir2/dir3/ contiene el fichero f6.latex y ningún otro.
24. Se pudo borrar el directorio /dir3/
25. El fichero /dir3/f6.latex no existe.
26. El directorio /dir3/ no existe.
27. El fichero /dir2/f4.latex no es binario y contiene los datos de texto esperados.
28. El fichero /dir2/f1.jpg es binario y contiene los datos binarios esperados.

Se considera que se ha pasado el test si, y solo si, todas las comprobaciones han resultado positivas.

7.2. Test de edición concurrente

Este segundo test tiene como objetivo evaluar el mecanismo de edición concurrente y comprobar si los resultados son consistentes.

Se han utilizado dos máquinas diferentes, aunque se puede modificar la prueba para ampliarla y añadir más participantes.

Tal y como se tiene configurada la red local en la que se realizan los test, hay dos máquinas. Una con la IP 10.0.0.1 de nombre «nenux» y la otra con la IP 10.0.0.2, de nombre «xergon». La máquina «nenux» crea una red DHT especificandose a sí misma como nodo inicial, mientras que «xergon» utiliza a «nenux» como nodo inicial al que conectarse.

La prueba consiste en que ambas máquinas editan el mismo fichero simultáneamente. Cada una de ellas escribe un total de 10 líneas, cada una de las cuales incluye su identificador de nodo. Después de cada edición, cada nodo realiza una pausa aleatoria de entre 2 y 45 segundos, para simular un comportamiento impredecible y no sincronizado.

Al final de la edición, ambos participantes realizan una pausa de duración igual al producto del tiempo máximo de las pausas de edición por el número de iteraciones realizadas. Este valor asegura que después de la pausa todos los participantes habrán terminado la edición.

Entonces se hace un recuento de líneas. El número total de líneas ha de ser igual al doble del número de iteraciones y la mitad de ellas deben haber sido generadas por cada uno de los dos participantes. Si esto no es así, el test falla.

En el siguiente fragmento de código podemos ver cómo se realiza la edición, intercalando las pausas:

```
private void dormir(int minSegs, int maxSegs) throws InterruptedException {
    Random rnd = new Random();
    int segundos = rnd.nextInt(maxSegs);
    if (segundos < minSegs)
        segundos = minSegs;

    System.out.println("A las " + this.hora() +
        " comienzo a dormir durante " + segundos + " segundos...");
    Thread.sleep(segundos * 1000);
    System.out.println("A las " + this.hora() + " estoy despierto");
}

int ITERACIONES = 10;
int SEGUNDOS_MIN = 2;
int SEGUNDOS_MAX = 45;
for (int i = 0; i < ITERACIONES; i++) {
    this.dormir(SEGUNDOS_MIN, SEGUNDOS_MAX);
    String contenidoActual = ctrlProyectoLatex.getTextoFichero(infoFichero);
    String nuevaLinea = String.format("%s, nueva línea #%d de %s\n",
        this.hora(), i, ColatexConcTest.idStr);
    String contenidoNuevo = contenidoActual + nuevaLinea;
    System.out.println("Nueva linea:" + nuevaLinea);
    System.out.println("Nuevo contenido:\n" + contenidoNuevo);
    ctrlProyectoLatex.editarFichero(infoFichero, contenidoNuevo);
}
```

Los resultados completos del test se puede consultar en el apéndice A.

8. Conclusiones y mejoras futuras

Se ha diseñado e implementado una aplicación que cumple con todos los requisitos que se marcaron para este PFC, los tests realizados (ver apéndice A) indican que la aplicación funciona correctamente.

La arquitectura elegida, compuesta por un núcleo con el cual es posible interactuar de varias formas diferentes (interficie web, comunicación con los servlets, servicios web y conexión RMI), permite separar totalmente la capa responsable de la lógica de negocia de la capa de presentación.

El mecanismo de comunicación RMI que se utiliza internamente para comunicar el núcleo con los servlets se pensó inicialmente con el objetivo de facilitar el desarrollo y depuración del sistema, aunque rápidamente se vio que esta separación entre la capa de lógica de negocio y presentación abría la puerta a la posibilidad de distribuir la carga entre varias máquinas.

En el caso concreto de este PFM, no parece que la necesidad de cómputo de la aplicación CoLaTeX requiera una redistribución de la carga en diferentes máquinas, pero la forma modular en la que se ha diseñado lo permite.

Para ello habría que instanciar los diferentes objetos RMI en varias máquinas y publicar su existencia mediante el registro RMI, y en la aplicación web simplemente habría que configurar en qué máquinas están disponibles los servicios.

Del mismo modo, la redistribución de la carga en diferentes máquinas (incluso su forma más simple mediante comunicación RMI dentro del mismo cliente ejecutando las dos aplicaciones) no afecta en absoluto a los demás participantes en la red DHT, ya que es una característica de la arquitectura interna de los clientes.

Es posible crear diferentes clientes, incluso en otras plataformas o lenguajes de programación, siempre que sean conformes a las especificaciones. En concreto, se deberían crear controladores que implementen las interfaces presentadas en sección dedicada a la fase de diseño.

La decisión de construir la aplicación del núcleo de CoLaTeX como una aplicación Java estándar se ha ido demostrando como muy acertada a medida que se avanzaba en el desarrollo. Ha facilitado mucho la depuración del programa y sobre todo, ha acelerado mucho el desarrollo especiamente debido a la rapidez en la compilación y también por no ser necesario desplegar la aplicación en el servidor después de cada cambio.

En CoLaTeX los requerimientos en cuanto a recursos cálculo son muy moderados, y la mayor parte son debidos a la ejecución del algoritmo WOOT en el cliente para crear el contenido consistente de los ficheros de texto.

Aún así, la misma arquitectura se puede utilizar para crear otros sistemas distribuidos descentralizados con objetivos diferentes a la edición concurrente de ficheros, y con necesidades de recursos computacionales muy diferentes.

La tecnología de *cloud computing* se podría utilizar para complementar la arquitectura propuesta.

Un ejemplo de otra aplicación posible sería el procesamiento en paralelo de ficheros de vídeo por grupos de usuarios geográficamente distantes. Este tipo de aplicaciones se caracterizan por demandar todos los recursos de computación disponibles (por ejemplo, ciclos de CPU) durante su ejecución.

Los ordenadores PC comerciales generalmente están infrautilizados, ya la mayoría de aplicaciones consumen una mínima parte de los recursos de computación durante su funcionamiento, exceptuando momentos puntuales. Por otra parte, su arquitectura generalista no los hace adecuados para aplicaciones muy intensivas de cálculo, a no ser que se amplíen con hardware especializado (por ejemplo, las tarjetas del fabricante NVIDIA para el procesamiento paralelo con múltiples núcleos). Aún contando con hardware especializado, se corre el riesgo de que quede infrautilizado la mayor parte del tiempo (normalmente el usuario ejecutará puntualmente las aplicaciones de cálculo intensivo, no durante todo el tiempo de funcionamiento de la máquina).

Por estos motivos, la tecnología cloud computing combinada con la arquitectura presentada en este proyecto es idónea para aplicaciones en las que se quieran combinar las capacidades de cálculo de varios nodos independientes colaborando de forma descentralizada. De esta forma, la capacidad de los ordenadores no queda infrautilizada, pero tampoco sobrecargada, ya que es posible contratar servicios de computación distribuidos en la nube para ampliar los recursos de los clientes. Existen varios proveedores de recursos en la nube, como los servicios EC2 de Amazon, u otros de Google, Yahoo, Microsoft, Zoho, etc.

El hecho de que CoLaTeX sea un sistema distribuido, descentralizado y colaborativo presenta claras ventajas, como ya se apuntaba en la introducción.

La más evidente, es que no existe un controlador explícito que se encargue de coordinar la actividad de los participantes. Esto elimina un punto crítico, en el que su fallo

puede significar la caída de todo el servicio o una interrupción hasta que se elige un nuevo coordinador, si lo hay. En un sistema descentralizado como CoLaTeX, los partipantes se encargan ellos mismos de su coordinación, lo cual le confiere una excelente tolerancia a fallos.

Además, esto hace que el sistema sea también muy resistente a la censura, que ya no existe un único punto que se pueda neutralizar para interrumpir el intercambio de información entre los participantes. En este proyecto no se ha considerado la posibilidad de cifrar los mensajes enviados y recibidos por los nodos, pero se podría considerar hacerlo si uno de los objetivos es evitar la censura o toda clase de ataques explícitos a la red.

Como se ha comentado anteriormente, otra ventaja es que los recursos de computación se aprovechan mejor en un sistema distribuido como el presentado, ya que se reparten entre todos los participantes. Aún en el caso de que fuera necesario obtener aún más recursos, se puede considerar la posibilidad de contratarlos a proveedores de servicios de cloud computing. El cliente solamente paga por los recursos consumidos, lo cual económicamente es mucho más ventajoso que disponer de un hardware especializado y posiblemente también infrautilizado la mayor parte del tiempo.

Finalmente, el hecho de que sea posible la edición colaborativa y simultánea de ficheros sin necesidad de bloquearlos durante su edición hace posible que los usuarios puedan modificar el contenido sin tener que preocuparse de si otros usuarios están editando el mismo fichero, o si los cambios de unos usuarios afectarán a los de otros. El algoritmo de consistencia utilizado asegura que los usuarios puedan trabajar de la misma forma que si estuvieran en un sistema centralizado y sobre todo, sin tener en cuenta la actividad de los demás usuarios, ya que el sistema se encarga de garantizar la consistencia y hacer presentes las contribuciones de todos y cada uno de los usuarios, todo de forma descentraliza.

Respecto a los aspectos técnicos, una posible mejora que se podría considerar para versiones futuras es la forma en la que se recuperan los patches de cambios para un determinado fichero durante su edición. En la versión actual se obtiene un vector de parches desde la DHT, aunque hubiera sido preferible que la aplicación fuese capaz de detectar la existencia de todos los parches de un fichero y recuperarlos, en lugar de obtener un único vector que los contiene.

La aplicación nunca elimina parches del historial de ediciones de un fichero, sino que únicamente se van añadiendo nuevos. Si, por ejemplo, se borra un fragmento, se añadirá un

parche con la operación *del*, pero no se eliminará el parche original con el *ins* del contenido).

Pero no se ha encontrado ninguna forma de obtener el conjunto de todas las claves almacenadas en un contexto determinado de la DHT, así que se ha tenido que recurrir a almacenar un vector con una clave conocida. Si en el futuro se encuentra la forma de hacerlo, o se añade esta operación en EasyPastry, sería conveniente realizar este cambio.

Otras mejoras menores que se podrían estudiar en el futuro para mejorar la aplicación podrían ser la posibilidad de importar múltiples ficheros mediante un solo botón desde la interficie web. Esto sería de utilidad si el grupo de usuarios ya cuenta con un proyecto LaTeX creado anteriormente y quiere importarlo en CoLaTeX. En la versión actual deberían crear los directorios y subir los ficheros uno a uno.

También se podría estudiar la posibilidad de que los usuarios pudieran elegir el idioma en el que se muestra la interficie y crear una web multi-idioma.

9. Glosario

Ajax : acrónimo de *Asynchronous JavaScript And XML* (JavaScript asíncrono y XML). Es una técnica de desarrollo web en la el cliente envía peticiones en segundo plano al servidor, que envía su respuesta en formato XML. Posibilita la existencia de interficies de usuario más dinámicas y rápidas, al no ser necesaria la recarga completa de la página en cada petición.

API : acrónimo de *Application Programming Interface* (interfaz de programación de la aplicación). Es el conjunto de funciones que permiten interactuar un determinado módulo o programa, las cuales constituyen una capa de abstracción que oculta los detalles de la implementación.

Axis2 : versión 2 de la implementación del protocolo SOAP[35] por parte de la Apache Foundation, que abstrae al programador de los detalles referentes a la utilización de SOAP y WSDL. Permite codificar servicios web en la parte de servidor y así como programas clientes.

Bunshin : es un módulo que opera en la capa 2 de la Common API y que ofrece servicios de replicación y caché para la DHT. La palabra *Bunshin* además de significar *«el espíritu/mente que detiene la espada»* en japonés, también se puede traducir como *clon* o *réplica*.

Concurrencia : en el contexto de este proyecto, la concurrencia es la situación que se da cuando dos usuarios editan al mismo tiempo el mismo fichero de texto y luego guardan sus cambios en la DHT. Si no se toman medidas para asegurar la concurrencia, en principio el último usuario que guarda el fichero destruye los cambios de los anteriores usuarios que no están presentes en su documento. Técnicamente, esto se conoce como una condición de carrera (*race condition*).

Para evitarlo, se utilizan mecanismos para asegurar la consistencia, como el algoritmo WOOT, en este caso.

Consistencia : en el contexto de este proyecto, es la propiedad de que el contenido final de un fichero editado concurrentemente por múltiples usuarios contenga las aportaciones

[35] SOAP es un protocolo para el intercambio de información estructurada en la implementación de servicios web.

de todos ellos, sin que la edición simultánea provoque la pérdida de información. Esta propiedad también se conoce como *coherencia*.

Desplegar : instalar una aplicación web en el servidor. Generalmente esta operación consiste en copiar un fichero con extensión *.war (Web ARchive)* en una carpeta concreta del servidor. En inglés, esta acción se traduce como *to deploy* o *deployment*.

DHT : siglas de *Distributed Hash Table* (tabla de dispersión distribuida). Es una tabla de dispersión que funciona sobre un sistema distribuido, de forma que cada nodo es responsable de un subconjunto de claves y sus correspondientes valores.

EasyPastry : capa de abstracción multicapa situada en lo alto de Pastry, que tiene como objetivo facilitar el desarrollo de aplicaciones P2P. Utiliza Bunshin para el control de réplicas y caché, y Scribe para el envío de mensajes entre nodos de la DHT. En la figura (21) se puede ver representada su arquitectura.

GUI : acrónimo de *Graphical User Interface* (interficie gráfica de usuario). Es una representación gráfica de un entorno que permite al humano interaccionar con la máquina, buscando la máxima usabilidad y ergonomía.

HTTP : siglas de *HyperText Transfer Protocol* (protocolo de transferencia de hipertexto). Es un protocolo desarrollado por la W3C[36] y la IETF[37] utilizado principalmente para transferir documentos de hipertexto (páginas web) al navegador del cliente. También es usado por los servicios web como vía de transporte, entre otros. El RFC 2616 define la primera versión estable, la v1.1.

JUnit : *framework* de clases Java creadas por Erich Gamma y Kent Beck para el desarrollo de tests unitarios.

JSP : siglas de *JavaServer Pages* (páginas de servidor en Java). Tecnología que permite crear contenido dinámico en respuesta a la solicitud HTTP de un cliente. Internamente se traducen y pre-compilan de forma que se genera un servlet para cada una de las páginas.

KBR : siglas de *Key-Based Routing* (encaminamiento basado en clave). Es la capa 0 en la Common API, que define el encaminamiento y la topología de la red. Es una

[36] World Wide Web Consortium.
[37] Internet Engineering Task Force.

capa utilizada en la gran mayoría de aplicaciones distribuidas que utilizan una red superpuesta.

LaTeX : lenguaje de marcado para documentos y sistema de generación, compuesto por las macros de TeX. Creado por Leslie Lamport en 1984 con la intención de facilitar el uso del sistema de tipografía TeX creado por Donald Knuth en 1978.

Log4j : sistema de registro (log) de mensajes desarrollado por la Apache Software Foundation. Permite seleccionar diferentes dispositivos de salida para los mensajes (pantalla, fichero, memoria, ...), así como filtrarlos por su nivel de prioridad.

Middleware : en los sistemas distribuidos, es una capa que abstrae al programador de los detalles de la red de comunicaciones subyacente, de los diferentes sistemas operativos presentes en ella y, en general, de la heterogeneidad de los componentes del sistema. Para ello proveen una API lo más sencilla posible. Por lo tanto, son un buen ejemplo del patrón de diseño *Facade*.

MVC (patrón) : siglas de *Model-View-Controller* (Modelo-Vista-Controlador). Es un patrón de diseño el que se separa la lógica de negocio de la capa representación. Se utilizan tres componentes: el modelo (representación consistente de los datos con los que opera la aplicación), la vista (interficie con la cual interactúa el usuario) y el controlador (se encarga de responder a eventos, obtener información de estado de el modelo y la vista, y de invocar cambios en ellos).

P2P : siglas de *Peer to Peer* (de igual a igual). Son sistemas descentralizados en los que la comunicación se realiza entre nodos jerárquicamente iguales, sin que exista explícitamente ningún coordinador.

Pastry : capa genérica para el desarrollo de aplicaciones P2P. Crea una red superpuesta y tolerante a fallos con nodos descentralizados y auto-organizados. También ofrece servicios de encaminamiento de mensajes, localización de objetos y balanceo de carga.

PFM : siglas de *Proyecto de Final de Máster*.

RMI : siglas de *Remote Method Invocation* (invocación remota de métodos). Sistema de comunicación de las máquinas virtuales Java que permite publicar las interfaces de las clases y remotamente invocar los métodos, desde una máquina virtual diferente y remota.

RUP : siglas de *Rational Unified Process* (proceso unificado de Rational). Proceso de desarrollo iterativo de software creado por la empresa Rational Software. Consta de las siguientes actividades y pasos: toma de requerimientos, especificación de la arquitectura, diseño de la implementación, testeo y mantenimiento.

Servlet : objetos del lenguaje Java que permiten leer una petición HTTP y dinámicamente generar la respuesta. Se puede implementar directamente su código, o generar automáticamente a partir de páginas JSP.

SOAP : originalmente un acrónimo de *Simple Object Access Protocol* (protocolo simple para el acceso a objetos). Es un protocolo para el intercambio de información estructurada en la implementación de servicios web.

Superpuesta, red : red a nivel de aplicación en la que se encaminan los mensajes de los nodos de un sistema P2P. Se la denomina red superpuesta (o *overlay network* en inglés) porque la red de comunicaciones subyacente normalmente dispone ya de una capa de red que realiza el encaminamiento de paquetes a nivel de red como, por ejemplo, la capa IP.

Tabla de dispersión : estructura de datos que utiliza una función de hash para obtener directamente la posición de un elemento en una tabla a partir de su clave. En inglés se conoce también como *hash table* o *hash map*.

TCO : siglas de *Total Cost of Ownership* (coste total de la propiedad). Es una estimación del coste económico que supone un determinado producto, incluyendo todos los gastos asociados. Por ejemplo, si el administrador de sistemas de una organización ha de calcular el TCO de sistema de copias de seguridad, no solamente tendría que tener en cuenta el coste de la licencia, sino además tambień el mantenimiento, costes debidos a la adaptación de los sistemas (transición desde un sistema diferente), testeo, etc.

TEX sistema tipográfico creado por Donald Knuth el año 1978. Sus objetivos principales son, por una parte, producir documentos de alta calidad en formato de libro, y por la otra, producir los mismos ordenadores en todos los ordenadores en los que se ejecuta.

Test unitario : es un test que comprueba el funcionamiento de la unidad testeable más pequeña de un sistema. En los programas basados en lenguajes orientados a objetos, esta unidad mínima es la clase.

WBS : siglas de *Work Breakdown Structure* (estructura de descomposición del trabajo). Es una herramienta utilizada en la gestión de proyectos y en la ingeniería para dividir el proyecto en múltiples tareas discretas. Para una explicación mucho más detallada, es recomendable consultar la bibliografía especializada [8].

Servicio web : conjunto de protocolos y estándares para el intercambio de información entre diferentes plataformas de hardware y software, con el objetivo de asegurar la interoperatividad. Las organizaciones OASIS y W3C son las responsables de la arquitectura y reglamentación de los servicios web. La *Organización para la Interoperabilidad de Servicios Web* (Web Services Interoperability Organization), WS-I vela por que el desarrollo de las especificaciones de servicios web asegure la interoperatividad.

Sistema distribuido : conjunto de ordenadores autónomos comunicados por una red de comunicaciones que gracias a un middleware específico forman una red homogénea de servicios integrados.

WOOT : modelo para la edición colaborativa consistente de textos en entornos P2P. La consistencia que consigue el algoritmo asegura la convergencia, la preservación de intención y la preservación de causalidad.

Los detalles del modelo y el algoritmo se pueden consultar en el artículo *Data Consistency for P2P Collaborative Editing* [3]

XML : siglas de *Extensible Markup Language* (lenguaje de marcas extensible). Especificación de propósito general para la creación de lenguajes de marcas, desarrollado por el W3C.

A. Apéndice: resultados del test de concurrencia

Aquí se presentan los resultados completos del test de concurrencia, ejecutado simultáneamente en dos máquinas diferentes, de nombre «nenux» y «xergon».

El ordenador «nenux» es un PC de sobremesa con las siguientes características:

- Placa base 8I945P Dual Graphic de Gigabyte Technology Co., Ltd.
- Procesador Intel(R) Pentium(R) 4 CPU 3.00GHz.
- Arquitectura de 32 bits.
- 2 Gb RAM.
- Interfaz de red utilizada para comunicar los nodos durante la prueba: NetLink BCM5789 Gigabit Ethernet PCI Express de Broadcom Corporation.
- Disco duro ATA MAXTOR STM332082 (320 GB).
- Sistema operativo de la distribución Ubuntu 8.10.
- Núcleo Linux 2.6.27-11-generic #1 SMP Wed Apr 1 20:57:48 UTC 2009 i686.
- Sistema de ficheros reiserfs.

El ordenador «xergon» es un PC portátil con las siguientes características:

- Ordenador modelo Aspire 7720G.
- Placa base Poyang de Acer.
- Procesador Intel(R) Core(TM)2 Duo CPU T5250 @ 1.50GHz, con dos núcleos.
- Arquitectura de 64 bits.
- 2 Gb RAM
- Interfaz de red utilizada para comunicar los nodos durante la prueba: NetLink BCM5787M Gigabit Ethernet PCI Express de Broadcom Corporation.
- Disco duro WDC WD1600BEVS-2 de Western Digital (160 GB).

- Sistema operativo de la distribución Ubuntu 8.10.
- Núcleo Linux 2.6.27-14-generic #1 SMP Wed Apr 15 19:29:46 UTC 2009 x86_64 GNU/Linux.
- Sistema de ficheros ext3.

El identificador de nodo de «nenux» en esta prueba es 831C592BD05D872B3D3F697F0DBA73A719031F5E, y en los tests aparece abreviado como 831C(...).

El identificador de nodo de «xergon» es 180E432C3634E55FE10C6B2E8B67C303CC1F19A2, y en los tests aparece abreviado como 180E(...).

A continuación podemos ver resultados[38] del test de concurrencia en la máquina **«nenux»**:

```
[junit] A las 04:55:47 comienzo a dormir durante 18 segundos...
[junit] A las 04:56:05 estoy despierto
[junit] Nueva linea:Mon Jun 08 04:56:06, nueva línea #0 de 831C(...)
[junit]
[junit] Nuevo contenido:
[junit] 04:56:06, nueva línea #0 de 831C(...)
[junit]
[junit] A las 04:56:08 comienzo a dormir durante 9 segundos...
[junit] A las 04:56:18 estoy despierto
[junit] Nueva linea:Mon Jun 08 04:56:19, nueva línea #1 de 831C(...)
[junit]
[junit] Nuevo contenido:
[junit] 04:56:18, nueva línea #0 de 180E(...)
[junit] 04:56:06, nueva línea #0 de 831C(...)
[junit] 04:56:19, nueva línea #1 de 831C(...)
[junit]
[junit] A las 04:56:21 comienzo a dormir durante 31 segundos...
[junit] A las 04:56:53 estoy despierto
```

[38] Se han editado debido a su gran longitud, al igual que los de «xergon».

```
[junit] Nueva linea:Mon Jun 08 04:56:55, nueva línea #2 de 831C(...)
[junit]
[junit] Nuevo contenido:
[junit] 04:56:18, nueva línea #0 de 180E(...)
[junit] 04:56:06, nueva línea #0 de 831C(...)
[junit] 04:56:19, nueva línea #1 de 831C(...)
[junit] 04:56:53, nueva línea #1 de 180E(...)
[junit] 04:56:55, nueva línea #2 de 831C(...)
[junit]
[junit] A las 04:56:57 comienzo a dormir durante 8 segundos...
[junit] A las 04:57:06 estoy despierto
[junit] Nueva linea:Mon Jun 08 04:57:07, nueva línea #3 de 831C(...)
[junit]
[junit] Nuevo contenido:
[junit] 04:56:18, nueva línea #0 de 180E(...)
[junit] 04:56:06, nueva línea #0 de 831C(...)
[junit] 04:56:19, nueva línea #1 de 831C(...)
[junit] 04:56:53, nueva línea #1 de 180E(...)
[junit] 04:56:55, nueva línea #2 de 831C(...)
[junit] 04:57:13, nueva línea #2 de 180E(...)
[junit] 04:57:07, nueva línea #3 de 831C(...)
[junit]
[junit] A las 04:57:10 comienzo a dormir durante 5 segundos...
[junit] A las 04:57:15 estoy despierto
[junit] Nueva linea:Mon Jun 08 04:57:17, nueva línea #4 de 831C(...)
[junit]
[junit] Nuevo contenido:
[junit] 04:56:18, nueva línea #0 de 180E(...)
[junit] 04:56:06, nueva línea #0 de 831C(...)
[junit] 04:56:19, nueva línea #1 de 831C(...)
[junit] 04:56:53, nueva línea #1 de 180E(...)
[junit] 04:56:55, nueva línea #2 de 831C(...)
[junit] 04:57:13, nueva línea #2 de 180E(...)
[junit] 04:57:07, nueva línea #3 de 831C(...)
```

```
[junit] 04:57:17, nueva línea #4 de 831C(...)
[junit]
[junit] A las 04:57:19 comienzo a dormir durante 18 segundos...
[junit] A las 04:57:38 estoy despierto
[junit] Nueva linea:Mon Jun 08 04:57:39, nueva línea #5 de 831C(...)
[junit]
[junit] Nuevo contenido:
[junit] 04:56:18, nueva línea #0 de 180E(...)
[junit] 04:56:06, nueva línea #0 de 831C(...)
[junit] 04:56:19, nueva línea #1 de 831C(...)
[junit] 04:56:53, nueva línea #1 de 180E(...)
[junit] 04:56:55, nueva línea #2 de 831C(...)
[junit] 04:57:13, nueva línea #2 de 180E(...)
[junit] 04:57:07, nueva línea #3 de 831C(...)
[junit] 04:57:17, nueva línea #4 de 831C(...)
[junit] 04:57:38, nueva línea #3 de 180E(...)
[junit] 04:57:39, nueva línea #5 de 831C(...)
[junit]
[junit] A las 04:57:41 comienzo a dormir durante 19 segundos...
[junit] A las 04:58:01 estoy despierto
[junit] Nueva linea:Mon Jun 08 04:58:02, nueva línea #6 de 831C(...)
[junit]
[junit] Nuevo contenido:
[junit] 04:56:18, nueva línea #0 de 180E(...)
[junit] 04:56:06, nueva línea #0 de 831C(...)
[junit] 04:56:19, nueva línea #1 de 831C(...)
[junit] 04:56:53, nueva línea #1 de 180E(...)
[junit] 04:56:55, nueva línea #2 de 831C(...)
[junit] 04:57:13, nueva línea #2 de 180E(...)
[junit] 04:57:07, nueva línea #3 de 831C(...)
[junit] 04:57:17, nueva línea #4 de 831C(...)
[junit] 04:57:38, nueva línea #3 de 180E(...)
[junit] 04:57:39, nueva línea #5 de 831C(...)
[junit] 04:58:08, nueva línea #4 de 180E(...)
```

```
[junit] 04:58:02, nueva línea #6 de 831C(...)
[junit]
[junit] A las 04:58:05 comienzo a dormir durante 27 segundos...
[junit] A las 04:58:32 estoy despierto
[junit] Nueva linea:Mon Jun 08 04:58:34, nueva línea #7 de 831C(...)
[junit]
[junit] Nuevo contenido:
[junit] 04:56:18, nueva línea #0 de 180E(...)
[junit] 04:56:06, nueva línea #0 de 831C(...)
[junit] 04:56:19, nueva línea #1 de 831C(...)
[junit] 04:56:53, nueva línea #1 de 180E(...)
[junit] 04:56:55, nueva línea #2 de 831C(...)
[junit] 04:57:13, nueva línea #2 de 180E(...)
[junit] 04:57:07, nueva línea #3 de 831C(...)
[junit] 04:57:17, nueva línea #4 de 831C(...)
[junit] 04:57:38, nueva línea #3 de 180E(...)
[junit] 04:57:39, nueva línea #5 de 831C(...)
[junit] 04:58:08, nueva línea #4 de 180E(...)
[junit] 04:58:02, nueva línea #6 de 831C(...)
[junit] 04:58:23, nueva línea #5 de 180E(...)
[junit] 04:58:34, nueva línea #7 de 831C(...)
[junit]
[junit] A las 04:58:36 comienzo a dormir durante 25 segundos...
[junit] A las 04:59:02 estoy despierto
[junit] Nueva linea:Mon Jun 08 04:59:03, nueva línea #8 de 831C(...)
[junit]
[junit] Nuevo contenido:
[junit] 04:56:18, nueva línea #0 de 180E(...)
[junit] 04:56:06, nueva línea #0 de 831C(...)
[junit] 04:56:19, nueva línea #1 de 831C(...)
[junit] 04:56:53, nueva línea #1 de 180E(...)
[junit] 04:56:55, nueva línea #2 de 831C(...)
[junit] 04:57:13, nueva línea #2 de 180E(...)
[junit] 04:57:07, nueva línea #3 de 831C(...)
```

```
[junit] 04:57:17, nueva línea #4 de 831C(...)
[junit] 04:57:38, nueva línea #3 de 180E(...)
[junit] 04:57:39, nueva línea #5 de 831C(...)
[junit] 04:58:08, nueva línea #4 de 180E(...)
[junit] 04:58:02, nueva línea #6 de 831C(...)
[junit] 04:58:23, nueva línea #5 de 180E(...)
[junit] 04:58:34, nueva línea #7 de 831C(...)
[junit] 04:58:47, nueva línea #6 de 180E(...)
[junit] 04:58:54, nueva línea #7 de 180E(...)
[junit] 04:59:03, nueva línea #8 de 831C(...)
[junit]
[junit] A las 04:59:05 comienzo a dormir durante 5 segundos...
[junit] A las 04:59:11 estoy despierto
[junit] Nueva linea:Mon Jun 08 04:59:12, nueva línea #9 de 831C(...)
[junit]
[junit] Nuevo contenido:
[junit] 04:56:18, nueva línea #0 de 180E(...)
[junit] 04:56:06, nueva línea #0 de 831C(...)
[junit] 04:56:19, nueva línea #1 de 831C(...)
[junit] 04:56:53, nueva línea #1 de 180E(...)
[junit] 04:56:55, nueva línea #2 de 831C(...)
[junit] 04:57:13, nueva línea #2 de 180E(...)
[junit] 04:57:07, nueva línea #3 de 831C(...)
[junit] 04:57:17, nueva línea #4 de 831C(...)
[junit] 04:57:38, nueva línea #3 de 180E(...)
[junit] 04:57:39, nueva línea #5 de 831C(...)
[junit] 04:58:08, nueva línea #4 de 180E(...)
[junit] 04:58:02, nueva línea #6 de 831C(...)
[junit] 04:58:23, nueva línea #5 de 180E(...)
[junit] 04:58:34, nueva línea #7 de 831C(...)
[junit] 04:58:47, nueva línea #6 de 180E(...)
[junit] 04:58:54, nueva línea #7 de 180E(...)
[junit] 04:59:03, nueva línea #8 de 831C(...)
[junit] 04:59:12, nueva línea #9 de 831C(...)
```

```
[junit]
[junit] A las 04:59:14 comienzo a dormir durante 360 segundos...
[junit] Test passed.
```

Estos son los resultados del test en la máquina «**xergon**», que se ejecutó al mismo tiempo que el test de «nenux»:

```
[junit] A las 04:55:56 comienzo a dormir durante 21 segundos...
[junit] A las 04:56:17 estoy despierto
[junit] Nueva linea:Mon Jun 08 04:56:18, nueva línea #0 de 180E(...)
[junit]
[junit] Nuevo contenido:
[junit] 04:56:18, nueva línea #0 de 180E(...)
[junit]
[junit] A las 04:56:21 comienzo a dormir durante 31 segundos...
[junit] A las 04:56:52 estoy despierto
[junit] Nueva linea:Mon Jun 08 04:56:53, nueva línea #1 de 180E(...)
[junit]
[junit] Nuevo contenido:
[junit] 04:56:18, nueva línea #0 de 180E(...)
[junit] 04:56:06, nueva línea #0 de 831C(...)
[junit] 04:56:19, nueva línea #1 de 831C(...)
[junit] 04:56:53, nueva línea #1 de 180E(...)
[junit]
[junit] A las 04:56:55 comienzo a dormir durante 15 segundos...
[junit] A las 04:57:11 estoy despierto
[junit] Nueva linea:Mon Jun 08 04:57:13, nueva línea #2 de 180E(...)
[junit]
[junit] Nuevo contenido:
[junit] 04:56:18, nueva línea #0 de 180E(...)
[junit] 04:56:06, nueva línea #0 de 831C(...)
[junit] 04:56:19, nueva línea #1 de 831C(...)
[junit] 04:56:53, nueva línea #1 de 180E(...)
[junit] 04:56:55, nueva línea #2 de 831C(...)
```

```
[junit] 04:57:13, nueva línea #2 de 180E(...)
[junit]
[junit] A las 04:57:15 comienzo a dormir durante 21 segundos...
[junit] A las 04:57:37 estoy despierto
[junit] Nueva linea:Mon Jun 08 04:57:38, nueva línea #3 de 180E(...)
[junit]
[junit] Nuevo contenido:
[junit] 04:56:18, nueva línea #0 de 180E(...)
[junit] 04:56:06, nueva línea #0 de 831C(...)
[junit] 04:56:19, nueva línea #1 de 831C(...)
[junit] 04:56:53, nueva línea #1 de 180E(...)
[junit] 04:56:55, nueva línea #2 de 831C(...)
[junit] 04:57:13, nueva línea #2 de 180E(...)
[junit] 04:57:07, nueva línea #3 de 831C(...)
[junit] 04:57:17, nueva línea #4 de 831C(...)
[junit] 04:57:38, nueva línea #3 de 180E(...)
[junit]
[junit] A las 04:57:40 comienzo a dormir durante 26 segundos...
[junit] A las 04:58:07 estoy despierto
[junit] Nueva linea:Mon Jun 08 04:58:08, nueva línea #4 de 180E(...)
[junit]
[junit] Nuevo contenido:
[junit] 04:56:18, nueva línea #0 de 180E(...)
[junit] 04:56:06, nueva línea #0 de 831C(...)
[junit] 04:56:19, nueva línea #1 de 831C(...)
[junit] 04:56:53, nueva línea #1 de 180E(...)
[junit] 04:56:55, nueva línea #2 de 831C(...)
[junit] 04:57:13, nueva línea #2 de 180E(...)
[junit] 04:57:07, nueva línea #3 de 831C(...)
[junit] 04:57:17, nueva línea #4 de 831C(...)
[junit] 04:57:38, nueva línea #3 de 180E(...)
[junit] 04:57:39, nueva línea #5 de 831C(...)
[junit] 04:58:08, nueva línea #4 de 180E(...)
[junit]
```

```
[junit] A las 04:58:11 comienzo a dormir durante 11 segundos...
[junit] A las 04:58:22 estoy despierto
[junit] Nueva linea:Mon Jun 08 04:58:23, nueva línea #5 de 180E(...)
[junit]
[junit] Nuevo contenido:
[junit] 04:56:18, nueva línea #0 de 180E(...)
[junit] 04:56:06, nueva línea #0 de 831C(...)
[junit] 04:56:19, nueva línea #1 de 831C(...)
[junit] 04:56:53, nueva línea #1 de 180E(...)
[junit] 04:56:55, nueva línea #2 de 831C(...)
[junit] 04:57:13, nueva línea #2 de 180E(...)
[junit] 04:57:07, nueva línea #3 de 831C(...)
[junit] 04:57:17, nueva línea #4 de 831C(...)
[junit] 04:57:38, nueva línea #3 de 180E(...)
[junit] 04:57:39, nueva línea #5 de 831C(...)
[junit] 04:58:08, nueva línea #4 de 180E(...)
[junit] 04:58:02, nueva línea #6 de 831C(...)
[junit] 04:58:23, nueva línea #5 de 180E(...)
[junit]
[junit] A las 04:58:26 comienzo a dormir durante 19 segundos...
[junit] A las 04:58:45 estoy despierto
[junit] Nueva linea:Mon Jun 08 04:58:47, nueva línea #6 de 180E(...)
[junit]
[junit] Nuevo contenido:
[junit] 04:56:18, nueva línea #0 de 180E(...)
[junit] 04:56:06, nueva línea #0 de 831C(...)
[junit] 04:56:19, nueva línea #1 de 831C(...)
[junit] 04:56:53, nueva línea #1 de 180E(...)
[junit] 04:56:55, nueva línea #2 de 831C(...)
[junit] 04:57:13, nueva línea #2 de 180E(...)
[junit] 04:57:07, nueva línea #3 de 831C(...)
[junit] 04:57:17, nueva línea #4 de 831C(...)
[junit] 04:57:38, nueva línea #3 de 180E(...)
[junit] 04:57:39, nueva línea #5 de 831C(...)
```

```
[junit] 04:58:08, nueva línea #4 de 180E(...)
[junit] 04:58:02, nueva línea #6 de 831C(...)
[junit] 04:58:23, nueva línea #5 de 180E(...)
[junit] 04:58:34, nueva línea #7 de 831C(...)
[junit] 04:58:47, nueva línea #6 de 180E(...)
[junit]
[junit] A las 04:58:49 comienzo a dormir durante 3 segundos...
[junit] A las 04:58:52 estoy despierto
[junit] Nueva linea:Mon Jun 08 04:58:54, nueva línea #7 de 180E(...)
[junit]
[junit] Nuevo contenido:
[junit] 04:56:18, nueva línea #0 de 180E(...)
[junit] 04:56:06, nueva línea #0 de 831C(...)
[junit] 04:56:19, nueva línea #1 de 831C(...)
[junit] 04:56:53, nueva línea #1 de 180E(...)
[junit] 04:56:55, nueva línea #2 de 831C(...)
[junit] 04:57:13, nueva línea #2 de 180E(...)
[junit] 04:57:07, nueva línea #3 de 831C(...)
[junit] 04:57:17, nueva línea #4 de 831C(...)
[junit] 04:57:38, nueva línea #3 de 180E(...)
[junit] 04:57:39, nueva línea #5 de 831C(...)
[junit] 04:58:08, nueva línea #4 de 180E(...)
[junit] 04:58:02, nueva línea #6 de 831C(...)
[junit] 04:58:23, nueva línea #5 de 180E(...)
[junit] 04:58:34, nueva línea #7 de 831C(...)
[junit] 04:58:47, nueva línea #6 de 180E(...)
[junit] 04:58:54, nueva línea #7 de 180E(...)
[junit]
[junit] A las 04:58:56 comienzo a dormir durante 22 segundos...
[junit] A las 04:59:19 estoy despierto
[junit] Nueva linea:Mon Jun 08 04:59:20, nueva línea #8 de 180E(...)
[junit]
[junit] Nuevo contenido:
[junit] 04:56:18, nueva línea #0 de 180E(...)
```

```
[junit] 04:56:06, nueva línea #0 de 831C(...)
[junit] 04:56:19, nueva línea #1 de 831C(...)
[junit] 04:56:53, nueva línea #1 de 180E(...)
[junit] 04:56:55, nueva línea #2 de 831C(...)
[junit] 04:57:13, nueva línea #2 de 180E(...)
[junit] 04:57:07, nueva línea #3 de 831C(...)
[junit] 04:57:17, nueva línea #4 de 831C(...)
[junit] 04:57:38, nueva línea #3 de 180E(...)
[junit] 04:57:39, nueva línea #5 de 831C(...)
[junit] 04:58:08, nueva línea #4 de 180E(...)
[junit] 04:58:02, nueva línea #6 de 831C(...)
[junit] 04:58:23, nueva línea #5 de 180E(...)
[junit] 04:58:34, nueva línea #7 de 831C(...)
[junit] 04:58:47, nueva línea #6 de 180E(...)
[junit] 04:58:54, nueva línea #7 de 180E(...)
[junit] 04:59:03, nueva línea #8 de 831C(...)
[junit] 04:59:20, nueva línea #8 de 180E(...)
[junit]
[junit] A las 04:59:23 comienzo a dormir durante 11 segundos...
[junit] A las 04:59:34 estoy despierto
[junit] Nueva linea:Mon Jun 08 04:59:36, nueva línea #9 de 180E(...)
[junit]
[junit] Nuevo contenido:
[junit] 04:56:18, nueva línea #0 de 180E(...)
[junit] 04:56:06, nueva línea #0 de 831C(...)
[junit] 04:56:19, nueva línea #1 de 831C(...)
[junit] 04:56:53, nueva línea #1 de 180E(...)
[junit] 04:56:55, nueva línea #2 de 831C(...)
[junit] 04:57:13, nueva línea #2 de 180E(...)
[junit] 04:57:07, nueva línea #3 de 831C(...)
[junit] 04:57:17, nueva línea #4 de 831C(...)
[junit] 04:57:38, nueva línea #3 de 180E(...)
[junit] 04:57:39, nueva línea #5 de 831C(...)
[junit] 04:58:08, nueva línea #4 de 180E(...)
```

```
[junit] 04:58:02, nueva línea #6 de 831C(...)
[junit] 04:58:23, nueva línea #5 de 180E(...)
[junit] 04:58:34, nueva línea #7 de 831C(...)
[junit] 04:58:47, nueva línea #6 de 180E(...)
[junit] 04:58:54, nueva línea #7 de 180E(...)
[junit] 04:59:03, nueva línea #8 de 831C(...)
[junit] 04:59:20, nueva línea #8 de 180E(...)
[junit] 04:59:12, nueva línea #9 de 831C(...)
[junit] 04:59:36, nueva línea #9 de 180E(...)
[junit]
[junit] A las 04:59:38 comienzo a dormir durante 360 segundos...
[junit] Test passed.
```

Índice de figuras

Referencias

[1] G. Oster, P. Molli, S. Dumitriu, and R. Mondejar. *UniWiki: A Reliable and Scalable Peer-to-Peer System for Distributing Wiki Applications.* Research Report RR-6848, LORIA – INRIA Nancy Grand Est (France), February 2009.
http://www.loria.fr/~oster/pmwiki/pub/papers/OsterRR09a.pdf

[2] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. *Towards a common API for Structured Peer-to-Peer Overlays.* In Proc. of IPTPS'03 Workshop, Berkeley, CA, Feb. 2003.
http://oceanstore.cs.berkeley.edu/publications/papers/pdf/iptps03-api.pdf

[3] G. Oster, P. Urso, P. Molli, and A. Imine. *Data Consistency for P2P Collaborative Editing.* In Proceedings of the ACM Conference on Computer-Supported Cooperative Work - CSCW 2006, pages 259-267, Banff, Alberta, Canada, Nov. 2006. ACM Press.

[4] *EasyPastry : Middleware Abstraction for Structured P2P Applications.*
http://ast-deim.urv.cat/easypastry/

[5] R. Mondéjar, P. García, C. Pairot. *Bunshin: DHT para aplicaciones distribuidas.*
http://planet.urv.es/bunshin/papers/Bunshin_CEDI2005.pdf

[6] Web del proyecto UniWiki en el repositorio SourceForge.
http://sourceforge.net/projects/uniwiki/

[7] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, Reading, Massachusetts, January 1995, 0201633612.

[8] Carl L. Pritchard. *Nuts and Bolts Series 1: How to Build a Work Breakdown Structure.* ISBN 1-890367-12-5.

[9] *Resources for Test Driven Development.*
http://www.junit.org/

[10] *Apache Axis2 Web services engine.*
http://ws.apache.org/axis2/

[11] *GNU General Public License.*
http://www.gnu.org/copyleft/gpl.html

[12] *Introducción a RUP*. Universidad Politécnica de Valencia.
https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Introducci%C3%B3n%20a%20RUP.doc

[13] *Resumen (deed) de la licencia Creative Commons by-sa.*
http://creativecommons.org/licenses/by-sa/3.0/deed.es

[14] *Texto legal completo de la licencia Creative Commons by-sa.*
http://creativecommons.org/licenses/by-sa/3.0/legalcode