

# Índice general

<b>1. Descripción y Planificación del Proyecto</b>	<b>1</b>
1.1. Descripción Funcional del Sistema . . . . .	1
1.2. Metodología de Desarrollo . . . . .	3
1.3. Requisitos de Alto Nivel del Sistema . . . . .	6
1.4. Iteraciones . . . . .	7
<b>2. Antecedentes</b>	<b>9</b>
2.1. Red local . . . . .	9
2.2. Demonio . . . . .	10
2.3. Cortafuegos : Netfilter/iptables . . . . .	10
2.4. MD5 . . . . .	11
<b>3. Definición Arquitectónica y Diseño Software</b>	<b>13</b>
3.1. Arquitectura física del Sistema . . . . .	13
3.2. Diseño Software . . . . .	14
3.3. Seguridad del software . . . . .	15
<b>4. Descripción de una iteración</b>	<b>19</b>
4.1. Inicio de la prueba . . . . .	20
4.2. Examen temporizado . . . . .	21
4.3. Denegar acceso a red . . . . .	21
4.4. Pruebas . . . . .	22
<b>5. Construcción, Implementación, Pruebas y Despliegue</b>	<b>25</b>
5.1. Construcción . . . . .	25
5.2. Creación de instaladores . . . . .	26
5.3. Pruebas . . . . .	29
<b>6. Discusión, Conclusiones y Trabajos Futuros</b>	<b>31</b>
6.1. Conclusiones . . . . .	31
6.2. Trabajos futuros . . . . .	32



# Índice de figuras

1.1. Diferencia entre iteración e incremento. . . . .	4
1.2. Carga de trabajo en las diferentes fases. . . . .	5
1.3. Diagrama de casos de uso del sistema . . . . .	6
3.1. Diagrama de despliegue del sistema . . . . .	14
3.2. Diagrama de actividad del sistema . . . . .	15
3.3. Diagrama de estados de la aplicación del alumno . . . . .	16
4.1. Aspecto de la GUI del alumno con una prueba temporizada. .	21
5.1. Plug-in Google Web Toolkit . . . . .	26
5.2. Cambios en el repositorio Subversion. . . . .	26



# Índice de cuadros

1.1. Requisitos de alto nivel del sistema . . . . .	7
4.1. Objetivos de la iteración . . . . .	20
4.2. Requisitos ya implementados del sistema . . . . .	20



# Capítulo 1

## Descripción y Planificación

El presente capítulo describe en líneas generales el ámbito funcional del proyecto, para delimitar su alcance, la metodología a usar durante el desarrollo del sistema y los requisitos de alto nivel del mismo, extraídos del ámbito funcional descrito.

«a completar»

### Contents

<b>1.1. Descripción Funcional del Sistema . . . . .</b>	<b>1</b>
<b>1.2. Metodología de Desarrollo . . . . .</b>	<b>3</b>
<b>1.3. Requisitos de Alto Nivel del Sistema . . . . .</b>	<b>6</b>
<b>1.4. Iteraciones . . . . .</b>	<b>7</b>

### 1.1. Descripción Funcional del Sistema

La presente sección describe el ámbito funcional del sistema software que deseamos construir, es decir, del *DrManhattan*, que es un *Sistema de Control de Accesos a Red para los Laboratorios de la Facultad de Ciencias*.

El objetivo del *DrManhattan* es controlar el acceso a la red local, existente en cada laboratorio de la Facultad de Ciencias de la Universidad de Cantabria, de los computadores conectados a ella. Se desea realizar dicho control sobre todo durante la realización de pruebas evaluables, de cara a evitar que se realicen accesos a contenidos no autorizados (e.g., páginas de internet con posibles soluciones a los problemas planteados, el directorio de trabajo de algún compañero, etc.) durante la realización de dichas pruebas. El sistema que se venía utilizando hasta ahora para evitar el mal uso de la red local consistía simplemente en desconectar la alimentación del concentrador de interconexión, deshabilitando la red. Es decir, se aplicaba el principio de muerto el perro, se acabó la rabia.

No obstante, el tener los diversos computadores interconectados mediante una red local, no sólo tiene inconvenientes, sino que también posee varias ventajas. Por ejemplo, ayuda a facilitar la distribución de material electrónico que resulte necesario para la realización del examen. También puede ser de gran utilidad para recoger los ejercicios realizados por los alumnos de una manera rápida y cómoda, ya que el alumno sólo tendría que enviar el material producido durante la prueba al computador o dirección que le indique el profesor. El método utilizado actualmente para realizar esta tarea consiste en que el profesor acude al computador del alumno cuando este desea entregar los ejercicios realizados y copia tales ejercicios en una memoria USB. Este proceso es lento, tedioso y en muchas ocasiones las memorias USB no son reconocidas por los computadores del laboratorio, lo que exige recurrir a otras técnicas. El tiempo empleado por los docentes para recopilar los ejercicios realizados por los alumnas suele oscilar entre la media hora y la hora completa, mientras que usando la red local dicho proceso podría realizarse en cuestión de minutos (cuando no menos).

Por tanto, el acceso de los computadores a la red local (e internet), se deberá controlar desde un computador distinguido, al que llamaremos *Watchman*, que será normalmente el computador asignado al docente. Dicho computador será el encargado de conceder y denegar el acceso a la red al resto de computadores conectados a la red local.

Se desea que el acceso a la red esté habilitado en las siguientes circunstancias:

1. Al comienzo de la realización de las pruebas, de forma que el docente pueda distribuir material electrónico necesario o de interés para la realización de la prueba (e.g., manuales, el enunciado de la prueba, etc.) entre los diferentes computadores de forma cómoda y eficiente.
2. Cuando el alumno haya finalizado la prueba, de forma que se puedan enviar los resultados a través de la red, evitando el tedioso proceso de tener que recogerlos de forma individual mediante copia en un memoria USB o dispositivo similar. En estos casos, sería además deseable, con objeto de evitar posteriores problemas, que el sistema comprobase la integridad de los archivos recibidos, es decir, que comprobase que no se han sufrido alteración alguna durante la transmisión.

Durante el periodo de tiempo que un alumno esté realizando una prueba, se debe denegar el acceso a la red del computador que esté utilizando para la realización de la prueba.

Por tanto, la secuencia de realización de una prueba evaluable sería tal como sigue:



1. El profesor y los alumnos acceden al aula y encienden sus correspondientes computadores. El acceso a la red está habilitado para todo el mundo.
2. El profesor envía el material necesario para la realización de la prueba a los computadores de los alumnos.
3. A continuación, una vez que un alumno comienza la prueba, se le deniega el acceso a la red al computador que esté utilizando.
4. Si durante la realización de la prueba un alumno considera que ha acabado y está satisfecho con los resultados producidos, indicará al sistema que ha concluido la prueba. Los ficheros producidos como material evaluable se enviarán al *Watchman*. Se comprobará que se han recibido correctamente y no están corruptos, por lo que se pueden abrir y leer sin problema alguno. Obviamente, para el envío de los ficheros de resultados habrá que habilitar de nuevo la red, pero se ha de evitar que el alumno pueda modificar dichos ficheros.
5. Debe existir también la posibilidad de que el docente decida que el tiempo de realización de la prueba ha concluido, por lo que deberá realizarse todo el proceso descrito en el punto anterior, pero para todos los computadores que se encuentren activos en ese momento y siendo el proceso iniciado desde el *Watchman*, en lugar de desde el propio computador del alumno.

Además, se guardarán registros de los eventos ocurridos en cada prueba, con el objetivo de tener información que permita auditar el transcurso de la misma. Dicho registro debe mantener constancia de eventos tales como hora de inicio de la prueba, hora de entrega de cada ejercicio, número de ficheros que se entregan, etc.

Para comodidad del alumno, en las pruebas con hora de finalización pre-determinada se mostrará en la interfaz de la aplicación los minutos restantes hasta el final.

La siguiente sección detalla la metodología de desarrollo que usaremos para la construcción de este sistema.

## 1.2. Metodología de Desarrollo

En esta sección se describe la metodología utilizada para desarrollar el sistema.

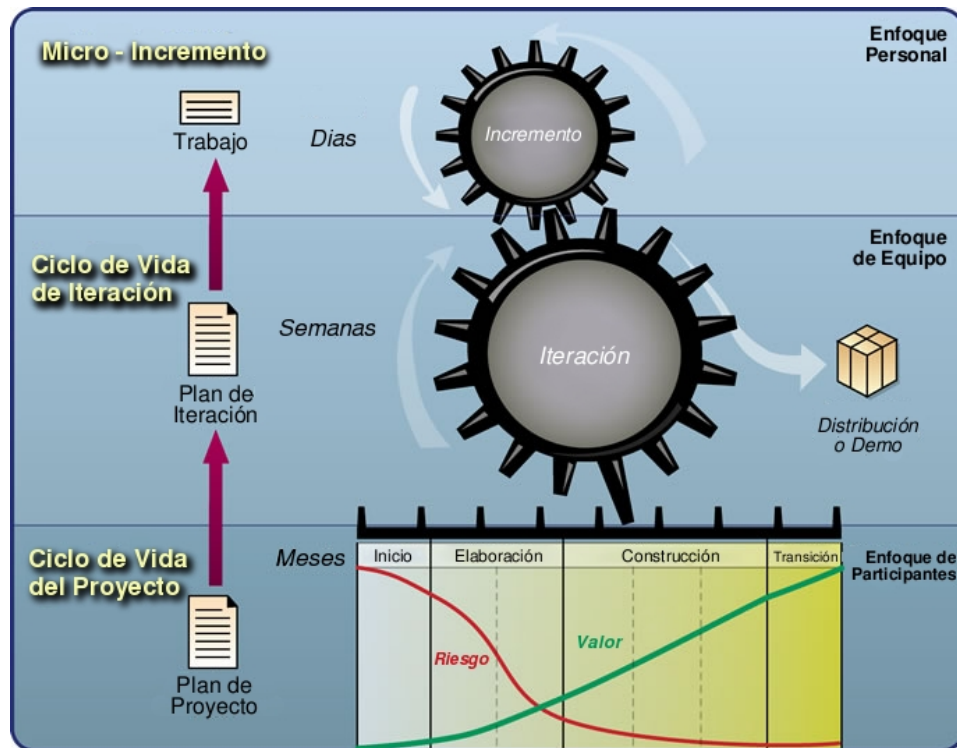


Figura 1.1: Diferencia entre iteración e incremento.

Se ha escogido el proceso de desarrollo *iterativo e incremental* que suele ser parte esencial en las metodologías de desarrollo ágiles.

La idea principal de este proceso se basa, como su nombre, en iteraciones e incrementos, que no son lo mismo. El software se construye mediante tareas repetitivas, *iteraciones*, en las que se añaden nuevas funcionalidades progresivamente, *incrementos*, para crear una versión del sistema que cumple más requisitos que la anterior. Se realizan tantas iteraciones sean necesarias hasta que todos los requisitos se hayan implementado y, por tanto, el sistema este finalizado.

Es decir, una iteración es un mini-proyecto en el que se obtiene una versión de cada una de las piezas del sistema, sean código o no, y un incremento se puede medir como la diferencia entre una iteración y la anterior.

El proceso se divide en cuatro partes fundamentales:

1. **Iniciación:** en esta fase se describe el ámbito funcional del sistema, los requisitos de alto nivel y se identifican posibles riesgos. Hay que comprender el sistema y sus límites.

2. **Elaboración:** el objetivo principal de esta fase es el de crear la arquitectura básica, para tener una visión de cómo será el sistema completo y además, proponer soluciones a los riesgos identificados.
3. **Construcción:** cómo su nombre indica, esta es la fase dónde se construye y prueba la aplicación. Se realiza un pequeño proceso en cascada en cada iteración de esta fase.
4. **Transición:** se despliega el sistema y finaliza el proceso.

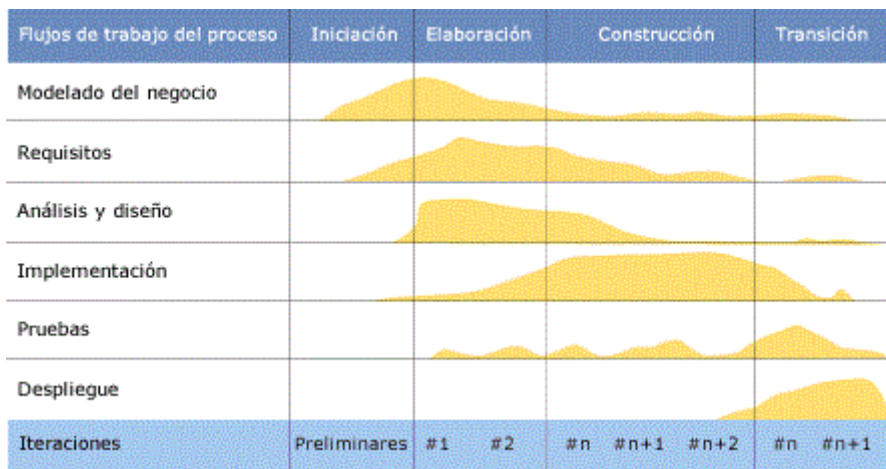


Figura 1.2: Carga de trabajo en las diferentes fases.

Cada una de estas fases, como es lógico, puede tener más de una iteración, en función del tamaño del proyecto, con tareas propias de la etapa en que se esté, así por ejemplo, en la etapa de construcción, en cada iteración se escoge un grupo de requisitos de alto nivel, se refinan, y partiendo de la versión del sistema obtenida en la iteración anterior, se diseñan, implementan y prueban esos requisitos, creando la versión que se utilizará en la siguiente iteración. En la figura 1.2 se muestra una aproximación de lo que sería la carga de trabajo que requieren las distintas tareas en función de la etapa en que se encuentre el proyecto.

Al finalizar el proceso obtenemos tanto el código de la aplicación, como modelos y documentación de la misma que se van creando a lo largo de las iteraciones.

Las principales razones por las que se ha escogido este tipo de proceso y no otro son:

- ♦ Poca incertidumbre a la hora de diseñar y planificar, ya que al realizarse en cada iteración y no del sistema completo, se tiene mayor

comprensión de los riesgos y de las tareas a realizar.

- ◆ Se adapta bien a cambios de requisitos.
- ◆ Si se priorizan los requisitos para implementarlos en las primeras iteraciones de la construcción, se obtiene una aplicación con funcionalidades clave en la que el cliente puede decidir si está o no correcto.
- ◆ Al trabajar con un subconjunto de requisitos en cada iteración la complejidad se reduce, lo que facilita que se reduzca también el número de errores producidos.

De la primera fase, *iniciación*, la descripción funcional detallando el alcance del sistema está descrita en la sección 1.1, a continuación se muestra un diagrama de casos de uso, basado en esa descripción y en la siguiente sección se describen los requisitos de alto nivel extraídos.

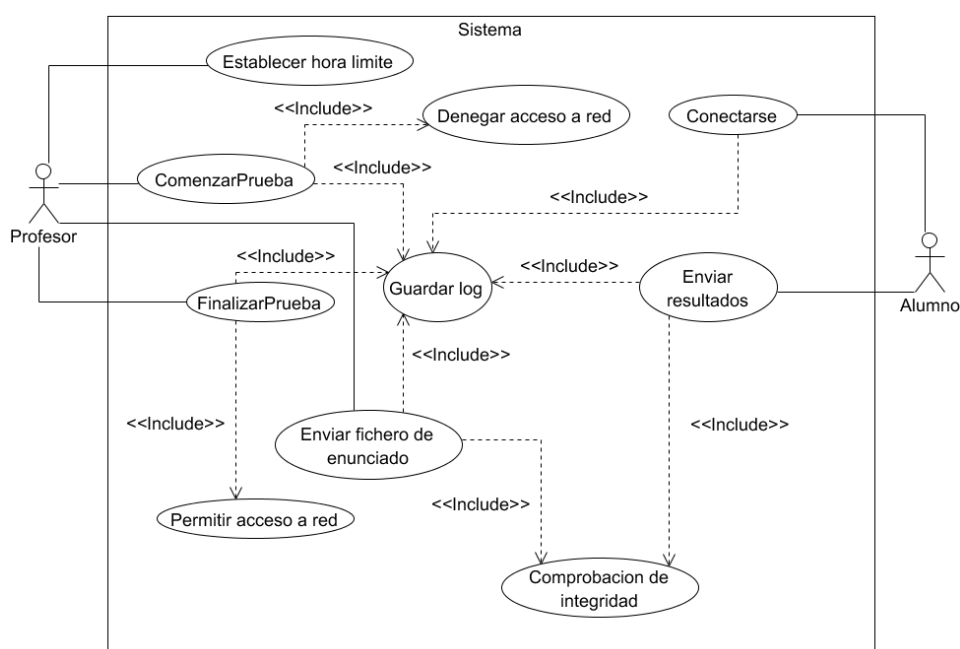


Figura 1.3: Diagrama de casos de uso del sistema

### 1.3. Requisitos de Alto Nivel del Sistema

Esta sección describe el segundo paso en nuestro proceso de desarrollo, de acuerdo a la metodología descrita en la sección anterior, que es la identificación de los requisitos de alto nivel que ha de satisfacer nuestro sistema software, de acuerdo a la descripción del ámbito funcional proporcionada en

Identificador	Descripción
R01	Un computador de la red debe poder ser designado como <i>Watchman</i> .
R02	Todos los computadores que no sean <i>Watchman</i> serán computadores normales, y podrán ser utilizados para la realización de las pruebas evaluables.
R03	El profesor debe ser capaz desde el <i>Watchman</i> de indicar el inicio de la prueba.
R04	El profesor debe ser capaz desde el <i>Watchman</i> de indicar el fin de la prueba.
R05	El profesor ha de ser capaz desde el <i>Watchman</i> de establecer una hora límite para la duración de la prueba.
R06	El profesor debe ser capaz desde el <i>Watchman</i> de enviar el fichero de enunciado al resto de computadores.
R07	El alumno desde su computador debe ser capaz de conectarse al <i>Watchman</i> .
R08	El alumno desde su computador debe ser capaz de enviar sus resultados al profesor.
R09	El alumno desde su computador debe ser capaz de indicar que da por finalizada la prueba.
R10	El alumno desde su computador debe tener la posibilidad de ver el tiempo restante el pruebas de duración prefijada.
R11	La aplicación del alumno ha de ser capaz de denegar el acceso a la red al empezar la prueba.
R12	La aplicación del alumno ha de ser capaz de permitir el acceso a la red al finalizar la prueba.
R13	La aplicación ha de ser capaz de comprobar que los archivos se han enviado correctamente.
R14	La aplicación del profesor ha de ser capaz de guardar registros de actividad.

Cuadro 1.1: Requisitos de alto nivel del sistema

la Sección 1.1.

En concreto, se han identificado los requisitos de alto nivel para nuestro sistema expuestos en la tabla 1.3.

## 1.4. Iteraciones

En esta sección se describen las iteraciones previstas para el desarrollo de la aplicación.

■ **Iteración 1: Crear un sistema distribuido.**

- Requisitos implementados: R06, R07
- Descripción: En la primera iteración la aplicación del alumno se podrá conectar al watchman para esperar el inicio de la prueba, envío de ficheros etc... y el profesor es capaz de enviar el enunciado.

■ **Iteración 2: Denegar acceso a red una vez comenzado el examen.**

- Requisitos implementados: R03, R05, R10, R11
- Descripción: En la segunda iteración el profesor puede establecer una hora límite e indicar el inicio de la prueba. La aplicación del alumno es capaz de denegar el acceso a la red cuando se inicia la prueba y en caso de que la duración este prefijada, consultar el tiempo restante

■ **Iteración 3: Habilitar red y enviar resultados**

- Requisitos implementados: R04, R08, R09, R12
- Descripción: En la tercera iteración se permite al profesor finalizar la prueba de modo general. Del mismo modo, cada alumno puede finalizar la prueba por su cuenta y enviar su archivo de resultados al computador del profesor. La aplicación del alumno, cuando este finaliza la prueba, habilita el acceso a la red.

■ **Iteración 4: Comprobación de integridad**

- Requisitos implementados: R13
- Descripción: En la cuarta iteración se implementa el sistema de comprobación de integridad de los ficheros transmitidos por red para garantizar la correcta recepción de los mismos.

■ **Iteración 5: Sistema de logs**

- Requisitos implementados: R14
- Descripción: Definir los mensajes necesarios para cuando en la aplicación ocurre algo que se desee guardar.

En cuanto a los requisitos R01 y R02 están implementados en arquitectura del sistema, no son funcionalidades.

En este capítulo se ha descrito el ámbito funcional del sistema y la metodología a utilizar para desarrollarlo. De acuerdo a este plan de desarrollo, se han obtenido los requisitos y planteado una serie de iteraciones a seguir para la construcción de la aplicación.

## Capítulo 2

# Antecedentes

El siguiente capítulo describe brevemente las tecnologías sobre las que se fundamenta el presente proyecto. Más concretamente, se explica el funcionamiento de las redes locales, del framework Netfilter para el filtrado de paquetes, y de los demonios en los sistemas Linux, tres aspectos fuertemente relacionados con el proyecto.

La aplicación a desarrollar está pensado que se utilice en los laboratorios de la Facultad de Ciencias de la Universidad de Cantabria, en estos laboratorios los computadores están conectado mediante una *red local* y utilizan sistemas Linux. Para gestionar los permisos de acceso a la red se va a utilizar un *demonio* que interactúe con *Netfilter* por medio de iptables.

### Contents

<b>2.1. Red local</b>	<b>9</b>
<b>2.2. Demonio</b>	<b>10</b>
<b>2.3. Cortafuegos : Netfilter/iptables</b>	<b>10</b>
<b>2.4. MD5</b>	<b>11</b>

### 2.1. Red local

Una *Red local* o LAN, siglas en inglés de Local Area Network, es un conjunto de computadoras conectadas entre sí en un área relativamente pequeña, como los laboratorios de la Facultad.

Cada uno de estos equipos interconectados en la red se conoce como nodo. Estos nodos son capaces de enviar, recibir y procesar comandos con el fin de transportar datos, así como compartir información y recursos a través de la red.

El funcionamiento de la red está estandarizado siendo el protocolo TCP/IP el más extendido.

## 2.2. Demonio

Un *demonio* (del inglés, *daemon*) es un tipo de proceso que posee la siguientes características:

1. Se ejecuta en segundo plano;
2. Generalmente se inicia en tiempo de arranque;
3. No usa los sistemas de entrada/salida estándar;
4. Mantienen la información que necesitan en ficheros especiales bien identificados.

Normalmente están cargados en memoria esperando una señal para ser ejecutados, por lo que su gasto de recursos no suele ser significativo.

Un ejemplo claro de demonio es *httpd*, que se ejecuta en los servidores web. El nombre viene de *HTTP Daemon* y es utilizado para aceptar peticiones, una vez que las acepta, crea otros procesos se encargan de atenderlas.

En el proyecto se implementa un demonio para interactuar con Netfilter, descrito en la siguiente sección.

## 2.3. Cortafuegos : Netfilter/iptables

Netfilter es un framework que permite filtrar de paquetes, traducción de direcciones y puertos de red y varias funcionalidades más para el manejo de paquetes. Es parte del núcleo de linux desde la versión 2.4 del mismo, sustituyendo a ipchains, bastante limitado en comparación con Netfilter.

Para interactuar con Netfilter una de las aplicaciones más usadas es *iptables*, siendo necesarios permisos de administrador para ello.

Ejemplo de uso de iptables:

```
# iptables -A INPUT -s 195.65.34.234 -j ACCEPT
```

El parámetro -A indica que se va a añadir una regla, el objetivo de la mismo es aceptar todos los paquetes entrantes provenientes del host indicado<sup>1</sup>. Del mismo modo, si lo que queremos es no aceptar las peticiones se cambiaría ACCEPT por DROP y si nos queremos referir a los paquetes salientes OUTPUT por INPUT. Si tenemos iptables con la configuración por defecto, se aceptan todos los paquetes entrantes y salientes, no hay ninguna restricción.

---

<sup>1</sup>en vez de la IP podemos poner su FQDN (*Fully Qualified Domain Name*) si lo deseáramos



## 2.4. MD5

MD5 son las siglas de Message-Digest Algorithm 5, algoritmo resumen del mensaje, que genera un número, conocido como firma, de 128 bits a partir del contenido de ese mensaje, de tal modo que, si por cualquier razón, el contenido cambia, el número resultante de aplicar el algoritmo al nuevo contenido cambiará también.

Es utilizado, por ejemplo, para realizar firmas a los paquetes que nos descargamos de los repositorios para instalar aplicaciones en Linux, de este modo, si vemos que la firma MD5 del fichero que hemos descargado, no concuerda con el que debería, puede ser que se haya corrompido en el proceso de descarga, o que lo descargado no sea lo que aparenta.

En la aplicación es utilizado para comprobar la integridad de los archivos enviados por la red, se genera el MD5 en el computador local del fichero a transferir, se transfieren el fichero y el resultado del MD5, al recibir, se reconstruye el fichero y se genera un nuevo MD5 a partir de él, si coinciden, tenemos la garantía de que se ha transferido correctamente.



## Capítulo 3

# Arquitectura y diseño

En el presente capítulo se presentan, mediante diagramas UML, la arquitectura física del sistema, los módulos y las comunicaciones entre ellos existentes en el sistema, y el funcionamiento del mismo.

«finalizar breve introducción»

### Contents

<b>3.1. Arquitectura física del Sistema . . . . .</b>	<b>13</b>
<b>3.2. Diseño Software . . . . .</b>	<b>14</b>
<b>3.3. Seguridad del software . . . . .</b>	<b>15</b>

### 3.1. Arquitectura física del Sistema

En esta sección se describe la arquitectura física del sistema mediante un diagrama de despliegue.

“LanzadorProfesor.jar”contiene lo necesario para mostrar la interfaz de usuario en la aplicación del profesor, así como la lógica de manejo de eventos y comunicación con las aplicaciones de los alumnos.

“LanzadorAlumno.jar”contiene las funciones para mostrar la interfaz de usuario de la aplicación del alumno, el manejo de eventos producidos por su uso y la lógica de comunicación con la aplicación del profesor y el demonio local.

“LanzadorDaemon.jar”contiene la lógica de comunicación con la aplicación del alumno y lo necesario para interactuar con iptables.

“iptables”es parte del núcleo del sistema Linux sobre el que se ejecutará la aplicación.

Cómo se puede observar, sólo se permite la existencia de una aplicación en el computador del profesor, a la que se conectan mediante TCP/IP un número indefinido de aplicaciones alumno, lógico, ya que en cada prue-

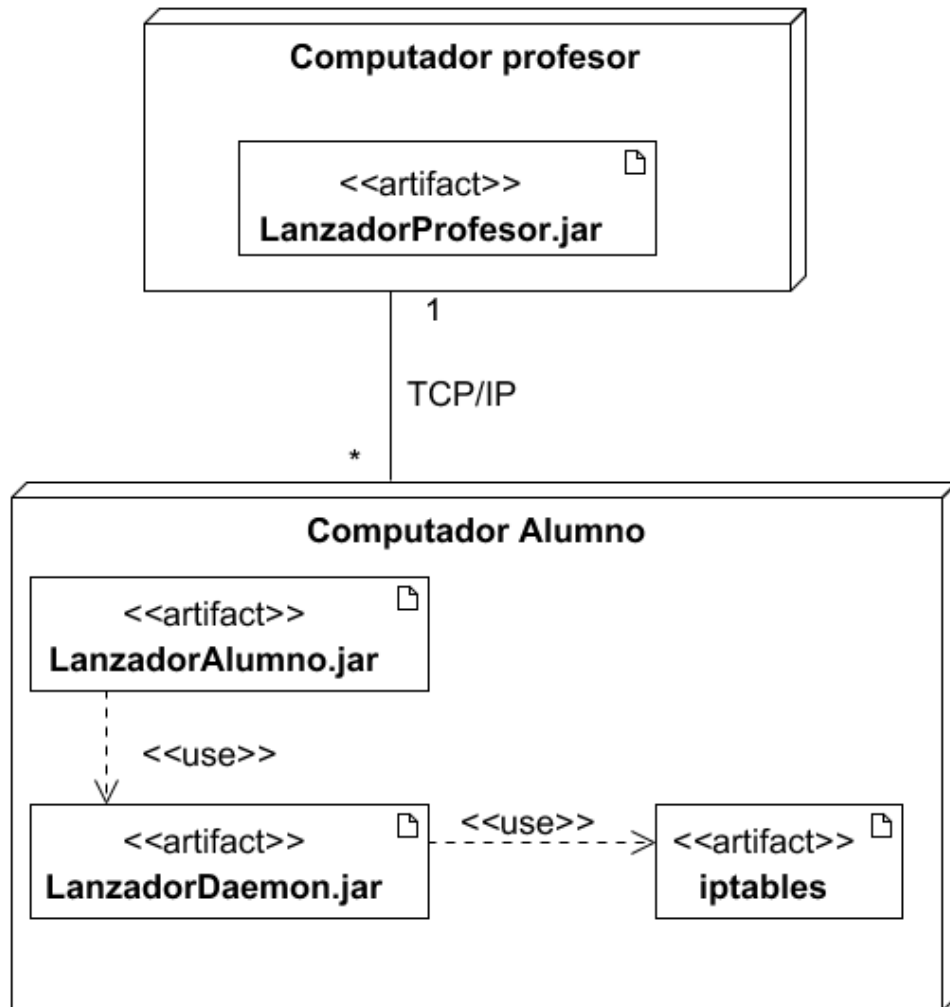


Figura 3.1: Diagrama de despliegue del sistema

ba suele haber varios los alumnos realizándola y uno el número de profesores.

A su vez, la aplicación del alumno interactúa con un único demonio del sistema.

### 3.2. Diseño Software

En esta sección se muestra un diagrama de actividad que resume las posibles interacciones que tanto el alumno como el profesor realizan con sus respectivas aplicaciones durante el transcurso normal de una prueba y un diagrama de estados para clarificar la aplicación del alumno.

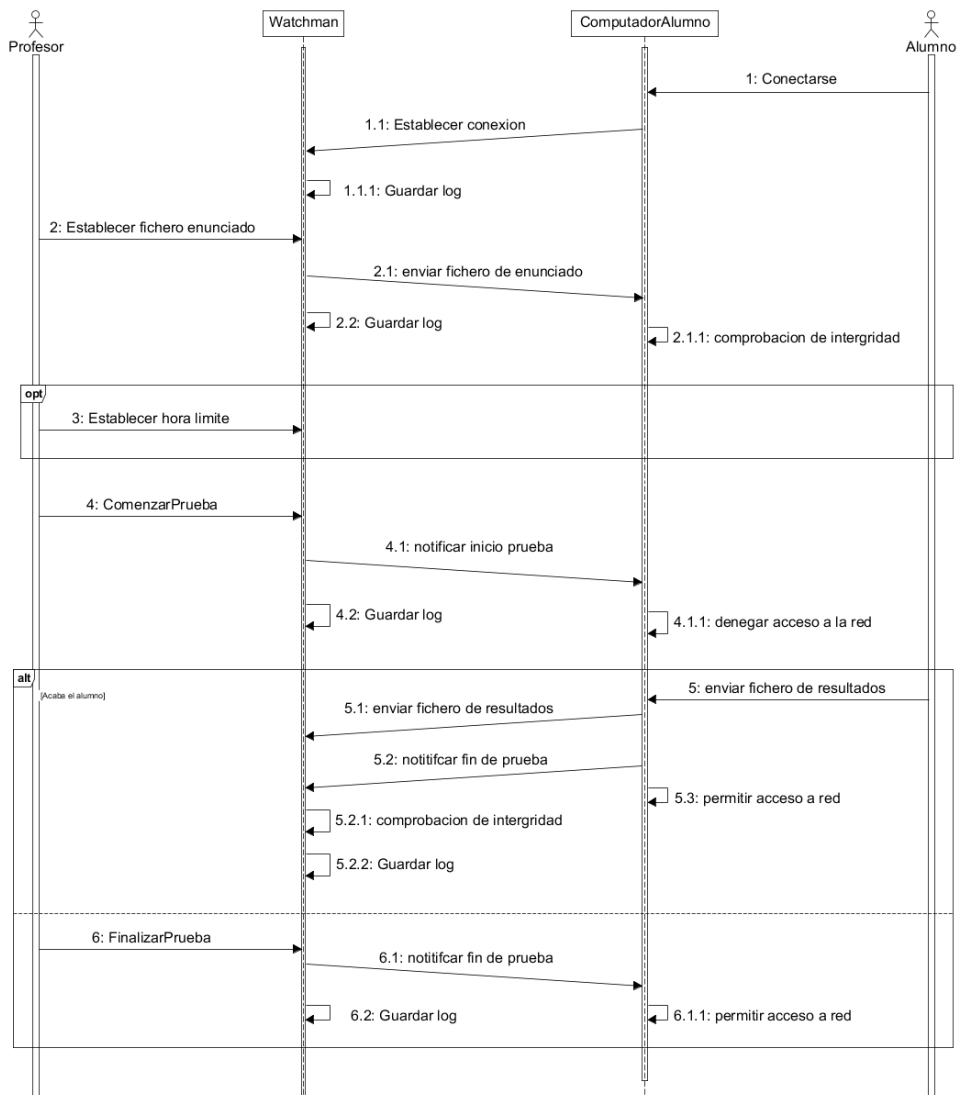


Figura 3.2: Diagrama de actividad del sistema

Vemos en la figura 3.2 que se siguen los pasos descritos en la sección 1.1

En la figura 3.3 se describen los estados por los que puede pasar la aplicación del alumno. Se corresponden directamente con el estado en que se encuentra la prueba.

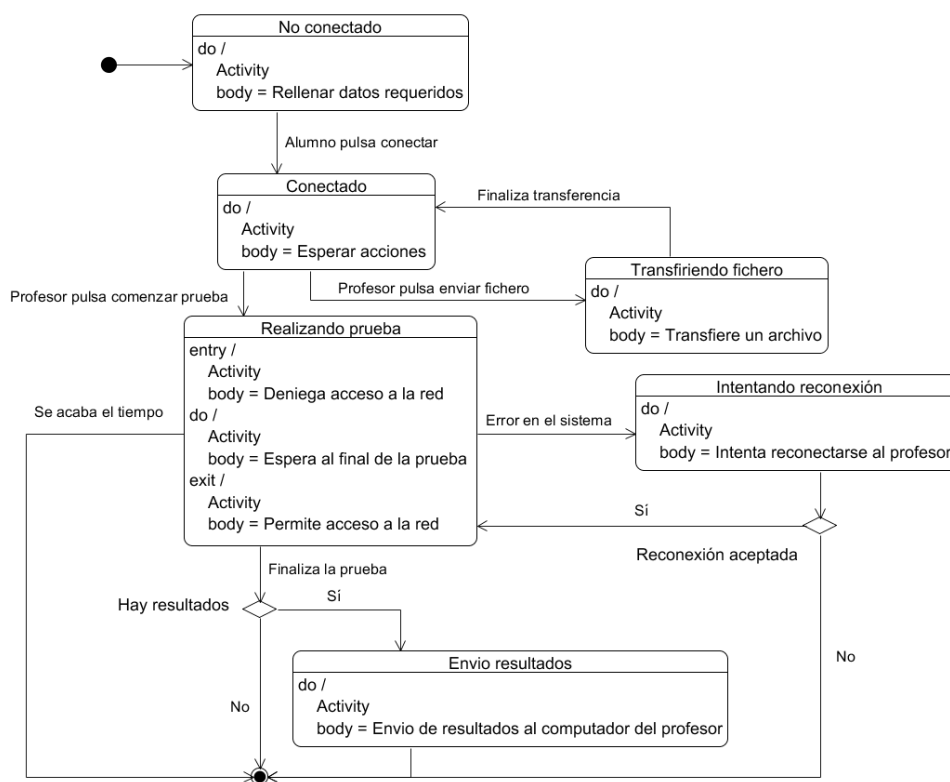


Figura 3.3: Diagrama de estados de la aplicación del alumno

### 3.3. Seguridad del software

La principal funcionalidad de la aplicación es la de denegar el acceso a la red en los computadores de los alumnos que están realizando la prueba, mientras la prueba esté en marcha, sin necesidad de apagar el router o switch del laboratorio. Esto se consigue por medio de iptables, sección 2.3, cambiando la política de los paquetes salientes una vez que empieza la prueba y hasta que acaba.

Por defecto iptables permite todo el tráfico que entre y salga del equipo, modificando las reglas para que deseche cualquier paquete destinado a cualquier equipo, conseguimos que no se pueda realizar ninguna petición a ningún nodo de la red, y, por tanto, tampoco recibiremos el contenido de la posible respuesta, el equipo del alumno queda aislado del resto.

A la hora de volver a permitir el acceso a la red, basta con modificar la política para el tratamiento de los paquetes salientes y restaurarla al estado anterior. Un alumno corriente no puede realizar esta operación ya que son necesarios privilegios de administrador para ello.

---

Para garantizar la imposibilidad de acceder a la red por parte del alumno, aunque se haya reiniciado el sistema, a propósito o por algún error, la aplicación es capaz de guardar el estado en el que se encontraba y cuándo se vuelva a iniciar, lo recuperará, como estaría realizando la prueba, bloquearía el acceso a la red, por tanto, incluso reiniciando el computador durante el transcurso de la prueba no se conseguiría acceso a la red.

Otro punto importante es el de la recogida de resultados automática, se ha de garantizar que el archivo no se ha corrompido a lo largo de la transferencia, para ello se realizan, tanto en el equipo que envía el fichero como en el que lo recibe, firmas MD5 al mismo para comprobar que no ha habido errores en la transferencia.





## Capítulo 4

# Segunda iteración

### Contents

---

<b>4.1. Inicio de la prueba . . . . .</b>	<b>20</b>
<b>4.2. Examen temporizado . . . . .</b>	<b>21</b>
<b>4.3. Denegar acceso a red . . . . .</b>	<b>21</b>
<b>4.4. Pruebas . . . . .</b>	<b>22</b>

---

En el siguiente capítulo se describe lo realizado en la segunda iteración de construcción. Se parte de lo obtenido en las iteraciones previas, que es, la descripción del sistema, la arquitectura del mismo y los diagramas de secuencia y estados, todo esto expuesto a lo largo de los capítulos 1 y 3.

En cuanto al estado de la aplicación, en la primera iteración de construcción se ha desarrollado un sistema distribuido, en el que desde la aplicación del alumno se puede conectar con la aplicación del profesor y transmitir ficheros.

El objetivo de esta iteración es que el profesor pueda establecer el inicio y de la prueba, poder prefijar una hora de finalización de la misma y que la aplicación del alumno sea capaz de denegar el acceso a la red mientras dure la prueba.

En la tabla 4 se pueden ver los requisitos a implementar durante esta iteración y en la 4 la base, o lo que es lo mismo, lo implementado en la iteración anterior. La lista global de requisitos se encuentra en la tabla 1.3.

En las siguientes secciones se describen los incrementos realizados dentro de esta iteración, generalmente coincide el número de incrementos en cada iteración con el número de requisitos a implementar.

Identificador	Descripción
R03	El profesor debe ser capaz desde el <i>Watchman</i> de indicar el inicio de la prueba.
R05	El profesor ha de ser capaz desde el <i>Watchman</i> de establecer una hora límite para la duración de la prueba.
R10	El alumno desde su computador debe tener la posibilidad de ver el tiempo restante el pruebas de duración prefijada.
R11	La aplicación del alumno ha de ser capaz de denegar el acceso a la red al empezar la prueba.

Cuadro 4.1: Objetivos de la iteración

Identificador	Descripción
R01	Un computador de la red debe poder ser designado como <i>Watchman</i> .
R02	Todos los computadores que no sean <i>Watchman</i> serán computadores normales, y podrán ser utilizados para la realización de las pruebas evaluables.
R06	El profesor debe ser capaz desde el <i>Watchman</i> de enviar el fichero de enunciado al resto de computadores.
R07	El alumno desde su computador debe ser capaz de conectarse al <i>Watchman</i> .
R13	La aplicación ha de ser capaz de comprobar que los archivos se han enviado correctamente.

Cuadro 4.2: Requisitos ya implementados del sistema

## 4.1. Inicio de la prueba

En esta sección se comenta cómo se consigue que el profesor pueda notificar a los alumnos el inicio de la prueba. Como ya hemos comentado, partimos de la iteración anterior en la cual los alumnos eran capaces de conectar a la aplicación del profesor. En la aplicación del alumno, cuando se crea una conexión, se inicia a su vez un hilo de ejecución que se mantiene a la espera de recibir distintas órdenes provenientes del computador del profesor. De este modo se hace muy simple mantener desde la aplicación del profesor una lista de las conexiones abiertas con cada alumno y, al presionar el botón de iniciar la prueba, recorrer esa lista enviando la orden por cada conexión. Cuando cada una de las aplicaciones del alumno recibe esa orden, actúa en consecuencia.

## 4.2. Examen temporizado

El profesor puede establecer una hora límite, llegada la cual, la prueba terminará automáticamente. Cuando el profesor define este límite en su aplicación y decide comenzar la prueba se envía también si hay una hora de fin y, en caso afirmativo, cual es. De este modo la aplicación del alumno puede mostrar una cuenta atrás con los minutos restantes para la finalización, para facilitar la referencia temporal. Se puede ver el resultado en la siguiente imagen.

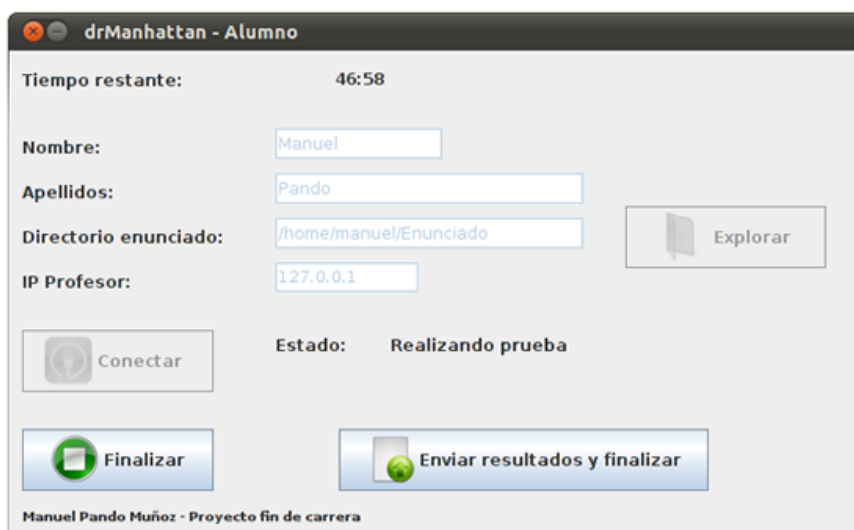


Figura 4.1: Aspecto de la GUI del alumno con una prueba temporizada.

## 4.3. Denegar acceso a red

Cómo ya hemos visto en las secciones anteriores, una vez que la aplicación del alumno recibe la orden de comenzar la prueba, se ha de denegar el acceso a la red. Para ello se utiliza iptables, por medio de un demonio.

El demonio creado es muy simple, se ejecuta en segundo plano esperando a que la aplicación del alumno conecte, y en función de lo requerido en ese momento, permitir o no el acceso a la red interactuando con iptables. Este demonio se inicia en tiempo de arranque y con los permisos necesarios para poder usar el cortafuegos.

Las órdenes concretas que ejecuta el demonio son, cuándo se desea denegar la red son:

```
iptables -P OUTPUT DROP
```

*iptables -A OUTPUT -s 127.0.0.1 -j ACCEPT*

Con la primera de ellas se cambia la política del tratamiento a los paquetes salientes del equipo, de modo que los deseche, dicho de otro modo, no se permiten las comunicaciones salientes con otros computadores, de esta forma se evita que se hagan peticiones, por ejemplo, a un servidor web.

La segunda es para permitir los paquetes salientes dirigidos al propio ordenador. Esto es necesario para la comunicación entre la aplicación del alumno y el demonio.

Puede parecer que el equipo queda totalmente aislado, incluso del computador del profesor, pero esto no es así, puesto que ya hay una conexión establecida y dado que la aplicación del alumno espera órdenes de la aplicación del profesor, puede seguir funcionando perfectamente, puesto que los paquetes de entrada sí que están permitidos, lo que estas instrucciones imposibilitan es la creación de nuevas conexiones y el envío de mensajes a cualquier computador que no sea el propio.

#### 4.4. Pruebas

En esta sección se describen las pruebas realizadas en la iteración con el objetivo de determinar si el funcionamiento de lo construido es correcto.

Debido a la naturaleza del sistema, automatizar pruebas con una herramienta estilo *JUnit* es prácticamente imposible, ya que funciona a base de eventos y es distribuido, por eso, para probar la aplicación utilizo el equipo de sobremesa y un portátil, para intentar recrear el entorno del laboratorio y ejecutar secuencias de acciones para desencadenar los eventos requeridos en cada funcionalidad.

- **R03** - El profesor debe ser capaz desde el *Watchman* de indicar el inicio de la prueba.
  - Pulsar el botón y ver que efectivamente llega la señal al computador del alumno.
- **R05** - El profesor ha de ser capaz desde el *Watchman* de establecer una hora límite para la duración de la prueba.
- **R10** - El alumno desde su computador debe tener la posibilidad de ver el tiempo restante en las pruebas de duración prefijada.
  - Establecer horas correctas futuras y comprobar que los minutos calculados restantes son correctos.

- 
- Establecer horas pasadas y comprobar que no se temporiza la prueba.
  - Establecer horas incorrectas y comprobar que es detectado.
- **R11** - La aplicación del alumno ha de ser capaz de denegar el acceso a la red al empezar la prueba.
- Después de que de inicio la prueba, en el computador del alumno, intentar abrir páginas web en el navegador, pings a equipos.



## Capítulo 5

# Construcción, Implementación, Pruebas y Despliegue

### Contents

5.1. Construcción . . . . .	25
5.2. Creación de instaladores . . . . .	26
5.3. Pruebas . . . . .	29

«introducción»

### 5.1. Construcción

Para la construcción del sistema se ha utilizado el entorno de desarrollo Eclipse 3.5.2, en su versión para java. Eclipse es un IDE de código abierto, multiplataforma, en el que se pueden instalar multitud de plug-ins para aumentar sus funcionalidades. Concretamente, para la realización de este proyecto se han usado el Google Web Toolkit en su versión para eclipse 3.5, que facilita mucho la creación de interfaces gráficas de modo visual, y Subclipse, para simplificar el trabajo con el repositorio Subversion.

En la figura 5.1 se muestra la interfaz del plug-in Google Web Toolkit. En la columna de la izquierda están los componentes listados en la parte superior, y características y atributos del seleccionado en la parte inferior. En el centro tenemos la paleta de componentes y en la derecha una vista previa de la interfaz que estemos construyendo. El código generado de la interfaz utiliza los componentes de la librería Swing.

«revisar las imágenes, sale muy mal»

«comentar lo del google code»

Google Code es un servicio de alojamiento de proyectos ofrecido por Google, que permite alojar código de Subversion de modo que se puede crear

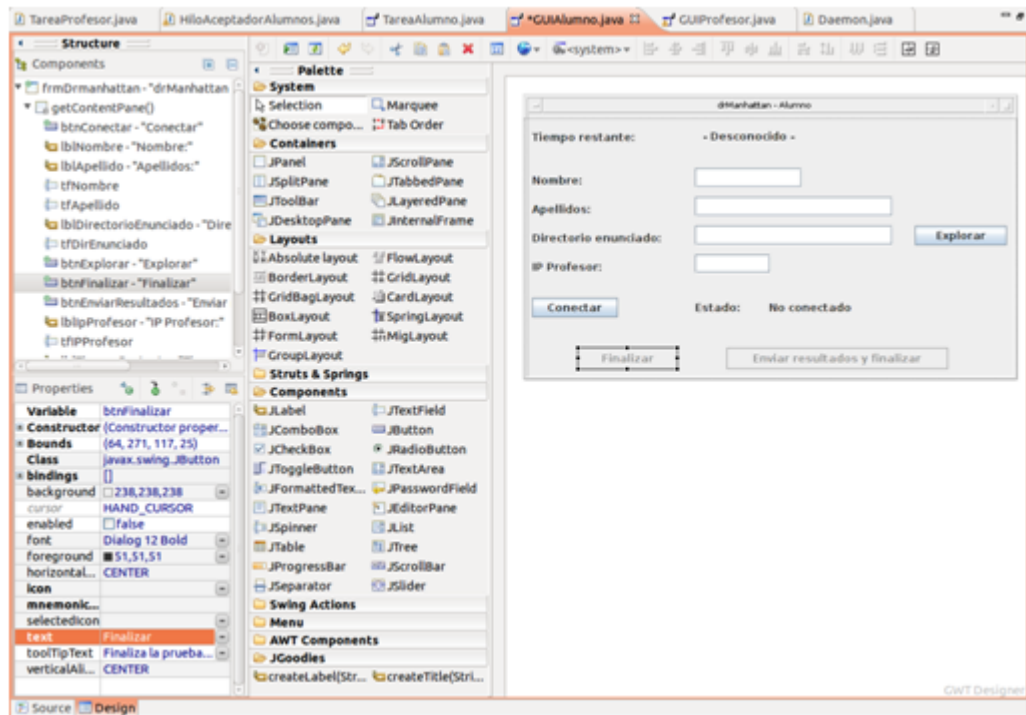


Figura 5.1: Plug-in Google Web Toolkit

```

7 min ago r129 ([No log message]) committed by manuel.p...@gmail.com - [No log message]
13 hours ago r128 ([No log message]) committed by manuel.p...@gmail.com - [No log message]
14 hours ago r127 ([No log message]) committed by manuel.p...@gmail.com - [No log message]
14 hours ago r126 ([No log message]) committed by manuel.p...@gmail.com - [No log message]
14 hours ago r125 ([No log message]) committed by manuel.p...@gmail.com - [No log message]
14 hours ago r124 ([No log message]) committed by manuel.p...@gmail.com - [No log message]
14 hours ago r123 ([No log message]) committed by manuel.p...@gmail.com - [No log message]

```

Figura 5.2: Cambios en el repositorio Subversion.

fácilmente un sistema de control de cambios, muy útil si se quiere regresar a un estado anterior del programa, recuperar algún objeto borrado, etc.

Todo ello sobre el sistema operativo Ubuntu 9.10 de 32 bits y la versión de kernel 2.6.35.

## 5.2. Creación de instaladores

Para que la aplicación pueda ser probada de un modo cómodo fuera del ambiente de desarrollo se hace necesaria la creación de instaladores. Esto se puede hacer, por ejemplo, con el comando *dpkg* en sistemas Linux derivados



de Debian, ya que con el se crean archivos `.deb`. En nuestro caso se han de crear dos instaladores, uno para la aplicación del profesor, y otro para la aplicación de los alumnos.

En el sistema de archivos, cada carpeta tiene su utilidad, conocida por todos, de modo que si, por ejemplo, queremos conocer los logs generados por alguna aplicación sabemos que hemos de buscar en `/var/log/`, `/etc/` para los temas de configuraciones, etc.

Esto es interesante conocerlo para que los archivos que coloque nuestro instalador en el sistema, estén correctamente ubicados, de modo que se puedan localizar fácilmente, en caso de querer modificarlos o eliminarlos.

Por ejemplo, la aplicación del alumno, creará en los siguientes directorios, los siguientes ficheros:

■ **`/etc/rc2.d:`**

- *S88drManhattanDaemon*

Este fichero contiene un script muy simple que se encarga de iniciar en segundo plano el demonio. Los ficheros en este directorio se ejecutan cuando se entra al segundo nivel de ejecución, en el arranque del sistema.

■ **`/usr/bin:`**

- *drManhattanAlumno.jar*
- *drManhattanAlumno.sh*
- *drManhattanDaemon.jar*

En este directorio se encuentran los ejecutables de las aplicaciones que pueden utilizar todos los usuarios.

■ **`/usr/share/applications:`**

- *drManhattanAlumno.desktop*

En nuestro caso se ha utilizado un sistema Ubuntu con el entorno de escritorio GNOME, en ese directorio se guardan las entradas de cada aplicación que se puede ejecutar en GNOME, en archivos de texto dónde se especifica la ruta del ejecutable, el icono del programa, su versión, comentarios, etc.

■ **`/usr/share/drManhattanAlumno/iconos:`**

- *icono.png*

- *envioFichero.png*
- *explorar.png*
- *fin.png*

Contiene los datos que no dependen de la arquitectura del sistema, imágenes, sonidos, etc.

■ **/usr/share/menu:**

- *drManhattanAlumno*

Cada archivo localizado en este directorio contiene la información necesaria para que GNOME pueda crear una entrada en el menú desplegable de aplicaciones.

«añadir los fichero de doc, licencia»

Una vez que están definidos los archivos y el directorio dónde los ha de colocar el instalador, creamos una carpeta a modo de raíz en la que simulamos el árbol de directorios anterior. Por ejemplo, creamos la carpeta `/home/usuario/deb`, y dentro de ella creamos `/home/usuario/deb/etc/rc2.d/`, `/home/usuario/deb/usr/bin/`, etc y los archivos correspondientes.

Dentro de la carpeta raíz se ha de crear además de lo anterior, un directorio llamado `DEBIAN` con tres ficheros, `control`, `postinst` y `postrm`. En el primero se especifican las características del paquete y los otros dos se ejecutan justo después de instalar el paquete y justo después de desinstalarlo, respectivamente, en nuestro caso actualizan los menús desplegables para que aparezca o desaparezca la aplicación.

Una vez que tenemos todo creado, cambiamos los permisos del directorio raíz y subdirectorios así:

```
chown root.root -R /home/usuario/deb/
```

Después de esto, la orden para empaquetarlo en un fichero `.deb` es:

```
dpkg -b /home/usuario/deb /home/usuario/paquete.deb
```

Para instalar, teniendo permisos de administrador:

```
dpkg -i /home/usuario/paquete.deb
```

### 5.3. Pruebas

Como se comentó en la sección 4.4 en la que se explicaban las pruebas realizadas durante una iteración, las pruebas han sido combinaciones de acciones, tanto de modo local mientras se desarrollaba, como en los laboratorios de la Facultad con el objetivo de imprimir mayor realismo en el entorno de ejecución como la carga de las aplicaciones.

1. Intentar conectar a una IP en la que no se está ejecutando la aplicación del profesor.
2. Sin alumnos conectados, enviar fichero de enunciado.
3. Con un alumno conectado, enviar ficheros de enunciado.
4. Con varios alumnos conectados, enviar ficheros de enunciado.
5. Iniciar prueba sin alumnos conectados.
6. Iniciar prueba con varios alumnos conectados.
7. Introducir hora errónea e iniciar prueba.
8. Iniciar prueba temporizada y esperar a que acabe el tiempo.
9. Iniciar prueba temporizada y finalizar manualmente.
10. Iniciar prueba normal y finalizar todos los alumnos sin resultados.
11. Iniciar prueba normal y finalizar alumnos sin resultados y alumnos con ellos.
12. Iniciar prueba normal y finalizar todos los alumnos con resultados.
13. Iniciar prueba normal y finalizar mientras hay alumnos realizando la prueba.
14. Iniciar prueba normal y reiniciar el computador de un alumno para intentar reconectar sin acabar la prueba.
15. Iniciar prueba normal, reiniciar el computador de un alumno, finalizar la prueba e intentar reconectar.
16. Iniciar prueba temporizada y reiniciar el computador de un alumno para reconectar antes de que se acabe la prueba.
17. Iniciar prueba temporizada y reiniciar el computador de un alumno para reconectar después de que se acabe la prueba.

Esta lista contiene las pruebas de integración y validación realizadas, esto es, cuándo se prueba el funcionamiento del sistema completo para comprobar que cumple los requisitos establecidos en la tabla 1.3.

## Capítulo 6

# Discusión, Conclusiones y Trabajos Futuros

### 6.1. Conclusiones

Como se comentaba al principio de este documento el objetivo del proyecto era el de crear una aplicación que facilitase la realización de pruebas evaluables a los docentes, automatizando ciertas tareas que sino han de realizarse manualmente, como la recogida de resultados, con el gasto de tiempo que ello conlleva, además de tener que ser realizado secuencialmente.

Para conseguir esto, se ha pasado por una fase de investigación, para tener conocimientos suficientes del problema a solucionar, las diferentes opciones posibles para ello y poder así tomar decisiones razonadas. Esto se transformó en una planificación y un diseño del proyecto, que se ha ido exponiendo a lo largo del documento.

La solución obtenida es una aplicación Open Source que permite, enviar archivos desde el computador del profesor a los alumnos, deshabilitar el acceso a la red durante el transcurso de la prueba, haciendo uso de iptables, software muy utilizado y probado, recoger automáticamente los resultados, comprobando la integridad de los ficheros, como principales funciones a través de interfaces visuales intuitivas y amigables.

Una de las lecciones que considero más importantes aprendidas a lo largo del desarrollo de este proyecto de fin de carrera, es la importancia de tener unos requisitos claros y definidos antes de planificar el diseño y construcción de la aplicación y tener un análisis y un diseño correcto y claro antes de codificar realmente el software y cómo el cambio en requisitos en fases avanzadas del proyecto puede suponer muchas horas de trabajo extra, en comparación con las primeras fases.

Merece la pena “perder” el tiempo revisando los requisitos y sus posibles incongruencias antes de empezar con la codificación.

Durante la realización de las pruebas descritas en la sección 5.3 se encontraba presente el administrador de sistemas de la Facultad, que propuso la funcionalidad de recuperar el estado frente a reinicios del sistema, parecía algo interesante añadirlo a la aplicación, así que se hizo. Esto realmente no significa un cambio en uno de los requisitos, sino la adición de uno nuevo, y gracias a la metodología usada para la construcción, que es flexible y prevé posibles cambios en los requisitos, el tiempo extra de codificación y diseño no fue mucho, comparado con lo que podría haber sido de utilizar una metodología en cascada.

En la siguiente sección se detallan posibles mejoras o adaptaciones a realizar en la aplicación para aumentar su funcionalidad, usabilidad y atractivo en general.

## 6.2. Trabajos futuros

Como continuación de este proyecto, en lo referido a su principal función, el mantenimiento de la integridad de las pruebas evaluables, o en el aumento de la comodidad a los docentes, pueden añadirse las siguientes líneas de desarrollo:

- **Deshabilitar USB:** Los dispositivos USB son otro posible aspecto que podría enturbar los resultados de una prueba, teniendo en cuenta que cada día son mas pequeños.
- **Acceso restringido:** En algunos momentos puede ser deseable que los alumnos accedan a ciertos contenidos en la red, apuntes de la asignatura en una plataforma como Moodle, por ejemplo, pero no al resto de internet, que el docente pudiese especificar qué contenidos son accesibles y durante cuánto tiempo sin duda puede resultar interesante.
- **Multiplataforma:** El proyecto ha sido construido íntegramente con el lenguaje de programación Java, pero el hecho de utilizar un cortafuegos propio del sistema Linux hace que no sea multiplataforma. Modificar esa parte del código y adaptarla para otros sistemas, haría que se pudiese utilizar independientemente de la plataforma, algo que es siempre atractivo.
- **Integración con otras aplicaciones:** Todos sabemos que en lo referente a la seguridad informática no se puede garantizar que una aplicación sea a prueba de todo, la integración con otros sistemas, de análisis

---

de tráfico por ejemplo, proporcionarían un aumento de la confianza en el proyecto.