Universidad Carlos III de Madrid

Trabajo de fin de grado

Sistema de ficheros para GNU/Linux con monitorización de operaciones

Autor: Jose María Hoya Quecedo

Tutor: Alejandro Calderón Mateos

Resumen

Hoy en día, los sistemas de ficheros son una de las estructuras lógicas peor optimizadas en un sistema operativo, debido a la necesidad de proveer una vista homogénea y simplificada del espacio de almacenamiento repartido entre soportes físicos muy diversos.

Uno de los pasos fundamentales que deben tomarse para mejorar esta situación es el de facilitar herramientas que nos permitan monitorizar un sistema de ficheros y extraer información que podamos entender directamente o procesar de forma automática para extraer conclusiones.

En este proyecto se propone un sistema con esta finalidad, y se realiza un estudio de las soluciones ya existentes o que se puedan utilizar, un análisis de requisitos y un diseño técnico del mismo, así como la implementación de un prototipo funcional como prueba de concepto.

Abstract

Nowadays, file systems are one of the worst optimised logical structures in an operating system due to the need of providing a simplified and homogeneous view of the storage space, which is distributed across different pieces of hardware.

One of the fundamental steps which must be taken so as to improve this situation is to provide tools which allow us to monitor a file system and recover information which can be understood directly or be processed automatically in order to draw conclusions from it.

The aim of this project is to propose such system, as well as making a study of existing solutions and existing tools which can be used, a requirements analysis and technical design of the system, and the implementation of a functional prototype as proof of concept.

Índice

1. Introducción 1

1.1. Motivación 1

1.2. Objetivos 2

1.3. Estructura del documento 2

2. Estado de la cuestión 4

2.1. Soluciones existentes 4

2.1.1. Inotify 4

2.1.2. LoggedFS 5

2.2. Herramientas utilizadas 6

2.2.1. FUSE 6

2.2.2. Debian GNU/Linux 8

2.2.3. Perl 9

2.2.4. SQLite 10

2.2.5. GTK 10

2.2.6. D3js 12

2.2.7. Eclipse 13

2.3. Marco regulador 15

3. Desarrollo de la solución 16

3.1 Metodología 16

3.2. Análisis 18

3.2.1. Casos de uso 18

3.2.1.1. Formato 18

3.2.1.2. Actores 19

3.2.1.3. Listado de casos de uso 19

3.2.2 Requisitos 22

3.2.2.1 Formato 22

3.2.2.2. Requisitos funcionales 23
3.2.2.3. Requisitos no funcionales 31
3.2.2.4 Matriz de trazabilidad 33
3.3. Diseño 35
3.3.1. Modelo de datos 36
3.3.1.1. Tabla operación 37
3.3.1.2. Tabla fichero 37
3.3.1.3. Tabla usuario 38
3.3.1.4. Tabla tipo_operación 38
3.3.1.5. Tabla tipo_fichero 39
3.3.2. Componentes 40
3.3.2.1. Sistema de ficheros 40
3.3.2.1.1. Operaciones 40
3.3.2.1.2. Formato del fichero de registro 41
3.3.2.1.3. Formato del fichero de configuración 41
3.3.2.2. Generador de informes 43
3.3.2.3. Controlador de base de datos 45
3.3.2.4. Matriz de trazabilidad 47
3.3.3. Interfaz gráfica 48
3.3.3.1. Montar y configurar sistema 48
3.3.3.2. Generar informe 51
3.3.3.3. Gestionar datos 52
4. Evaluación 55
4.1. Escritura y lectura de datos 55
4.2. Rendimiento en metadatos 57
5. Planificación y presupuesto 59
5.1. Planificación 59
5.2. Presupuesto 62
5.2.1. Costes 62

5.2.1.1. Personal 62
5.2.1.2. Material informático 63
5.2.1.3. Aplicaciones 63
5.2.1.4. Bienes fungibles 64
5.2.1.5. Costes indirectos 64
5.2.2. Resumen 65
6. Conclusiones y trabajos futuros 66
6.1. Conclusiones 66
6.2. Trabajos futuros 67
Anexo I. Acrónimos 68
Anexo II. Manual de instalación 69
Anexo III. Manual de usuario 71
Referencias 75

Índice de figuras

Ilustración 1. Ruta de una llamada en FUSE 8
Ilustración 2. Ejemplo de interfaz gráfica en GTK 12
Ilustración 3. Ejemplo de un gráfico interactivo en d3js 13
Ilustración 4. Ejemplo de una ventana de Eclipse 14
Ilustración 5. Fases de una metodología en cascada 16
Ilustración 6. Casos de uso de un usuario normal 19
Ilustración 7. Casos de uso de un administrador 21
Ilustración 8. Arquitectura general del sistema 36
Ilustración 9. Esquema relacional 36
Ilustración 10. Ejemplo de gráfico generado 44
Ilustración 11. Configurar sistema, sin montar 49
Ilustración 12. Mensaje de éxito 49
Ilustración 13. Mensaje de error 49
Ilustración 14. Configurar sistema, montado 50
Ilustración 15. Nueva regla de registro 50
Ilustración 16. Seleccionar punto de montaje 51
Ilustración 17. Generar informe 51
Ilustración 18. Nuevo fichero de resultados 52
Ilustración 19. Ejemplo de documento generado 52
Ilustración 20. Gestión de datos, sin BD seleccionada 53
Ilustración 21. Nueva base de datos 53
Ilustración 22. Seleccionar base de datos 54
Ilustración 23. Gestión de datos, con BD seleccionada 54
Ilustración 24. Mensaje de confirmación de guardado 54
Ilustración 25. Tiempos de lectura 56
Ilustración 26. Tiempos de escritura 57
Ilustración 27. Tasa de creación 58
Ilustración 28. Tasa de eliminación 58
Ilustración 29. Ejemplo de documento generado 73

Índice de tablas

Tabla 1. Evento de inotify 4
Tabla 2. CU-1 20
Tabla 3. CU-2 20
Tabla 4. CU-3 20
Tabla 5. CU-4 21
Tabla 6. CU-5 21
Tabla 7. RF-1 23
Tabla 8. RF-2 24
Tabla 9. RF-3 24
Tabla 10. RF-4 24
Tabla 11. RF-5 24
Tabla 12. RF-6 25
Tabla 13. RF-7 25
Tabla 14. RF-8 25
Tabla 15. RF-9 25
Tabla 16. RF-10 26
Tabla 17. RF-11 26
Tabla 18. RF-12 26
Tabla 19. RF-13 27
Tabla 20. RF-14 27
Tabla 21. RF-15 27
Tabla 22. RF-16 27
Tabla 23. RF-17 28
Tabla 24. RF-18 28
Tabla 25. RF-19 28
Tabla 26. RF-20 29
Tabla 27. RF-21 29
Tabla 28. RF-22 29
Tabla 29. RF-23 29
Tabla 30. RF-24 30
Tabla 31. RF-25 30
Tabla 32. RF-26 30
Tabla 33. RF-27 30

Tabla 34. RF-28 31
Tabla 35. RF-29 31
Tabla 36. RNF-1 31
Tabla 37. RNF-2 32
Tabla 38. RNF-3 32
Tabla 39. RNF-4 32
Tabla 40. RNF-5 32
Tabla 41. RNF-6 33
Tabla 42. RNF-7 33
Tabla 43. Matriz de trazabilidad de requisitos 34
Tabla 44. Operación 37
Tabla 45. Fichero 38
Tabla 46. Usuario 38
Tabla 47. Tipo_operación 39
Tabla 48. Tipo_fichero 39
Tabla 49. Funciones que deben interceptarse 40
Tabla 50. Estructura del fichero de configuración 42
Tabla 51. Atributos de un gráfico 45
Tabla 52. Esquema del método para crear base de datos 46
Tabla 53. Esquema del método para volcar datos de registro 46
Tabla 54. Esquema del método para recuperar información 46
Tabla 55. Matriz de trazabilidad de componentes 48
Tabla 56. Resultados de dd 56
Tabla 57. Resultados de fdtree 58
Tabla 58. Planificación del proyecto 60
Tabla 59. Diagrama de Gantt del proyecto 61
Tabla 60. Costes de personal 63
Tabla 61. Costes de material informático 63
Tabla 62. Costes de aplicaciones 64
Tabla 63. Costes de bienes fungibles 64
Tabla 64. Costes indirectos 65
Tabla 65. Resumen de costes 65

1. Introducción

1.1. Motivación

La tendencia actual en el diseño de sistemas de ficheros es la de intentar ocultar al usuario la organización interna de los datos para presentarle una visión lo más homogénea posible de su espacio de almacenamiento. De esta forma el usuario no tiene que preocuparse de cosas como tener varios discos duros en lugar de uno o de guardar los ficheros en sectores contiguos del disco para evitar la fragmentación, sólo tiene que establecer la estructura lógica que le interese y el sistema se encarga de adaptarlo a la organización física subyacente.

Sin embargo, en el proceso de facilitarle la vida al usuario se pierde información que puede ser útil para un administrador del sistema, aunque no se considere relevante o inteligible para el usuario final. Un caso podría ser el de tener varios soportes con distintas tasas de lectura y escritura que, por estar bajo la misma partición lógica, se consideran iguales. Por otra parte, los sistemas de ficheros suelen degradarse con el tiempo (en términos de rendimiento) y estar mal aprovechados. Además, los discos duros son el componente más lento de un ordenador, por lo que se hace imperativo optimizar el uso que hacemos de él. Para ello hay mucha información que podría sernos de utilidad:

- Conocer qué ficheros se usan más y qué ficheros se usan menos, para poder decidir cómo distribuirlos entre los distintos medios de almacenamiento.
- En un entorno multiusuario, el sistema se suele distribuir de forma equitativa entre todos los usuarios, pero el uso que realiza cada uno es distinto. Tener una idea más clara del perfil de uso de cada usuario puede ayudarnos por ejemplo a ajustar las cuotas de disco.
- Detectar qué procesos hacen un uso más intensivo del disco, a fin de ver si podemos programarlos para que se ejecuten en las horas de menos actividad por parte de los usuarios.

Por último, dejando de lado cuestiones de rendimiento y aprovechamiento del sistema, la creación de perfiles de usuario se podría utilizar con otros fines como el de detectar posibles intrusiones cuando se realiza un uso del sistema de ficheros distinto al habitual.

1.2. Objetivos

El objetivo fundamental de este proyecto es la elaboración de una solución al problema de la falta de información sobre el uso del sistema de ficheros. Para ello se propone la creación de un sistema de ficheros que registre el uso que se hace de él y del cual podamos extraer esta información para poder analizarla.

Por tanto, los principales objetivos que derivan de este son:

- Analizar qué tipo de información necesitamos conocer sobre el sistema de ficheros.
- Diseñar un sistema de ficheros que registre el uso que se hace del mismo y presente esta información al usuario.
- Crear un prototipo funcional de este sistema de ficheros.

1.3. Estructura del documento

El presente documento está dividido en las siguientes secciones:

1. Estado de la cuestión: En esta sección se hace un breve repaso del estado actual del área a tratar, en este caso los sistemas de ficheros. Además se describen las soluciones similares a la propuesta que ya existan, así como las herramientas de las se hace uso para el desarrollo de la misma.

2. Desarrollo de la solución: En esta sección se lleva a cabo un análisis de requisitos y un diseño técnico de la propuesta, siguiendo una metodología determinada.

3. Evaluación: En esta sección se prueba la solución y se registran los resultados obtenidos.

4. Planificación y presupuesto: En esta sección se detalla la planificación seguida para elaborar el proyecto, así como un desglose de todos los costes en los que se ha incurrido para ello.

5. Conclusiones y trabajos futuros: En esta sección se decide si la solución ha cumplido con los objetivos, se repasan dificultades que se hayan encontrado a lo largo de la elaboración del proyecto y se da una valoración personal. Además se proponen posibles futuras líneas de investigación o mejoras de la solución.

2. Estado de la cuestión

2.1. Soluciones existentes

En esta sección se describen las principales soluciones que pueden ser aplicables para la resolución del problema planteado, así como los principales motivos por los cuales no aportan una solución completa.

2.1.1. Inotify

Inotify es un componente de Linux que proporciona una interfaz para monitorizar cambios en el sistema de ficheros. Puede utilizarse para registrar las principales operaciones que se realizan sobre ficheros, como accesos, modificaciones, borrados, movimiento de ficheros entre directorios, etc. Para ello, debemos instanciar un observador (*watch*) asociado a un directorio, al cual le especificamos una máscara que determina el tipo de operaciones que va a observar.

La información contenida en cada evento que registra inotify es la siguiente [1]:

Atributo	**Descripción**
wd	Un descriptor que identifica el observador.
mask	La máscara que indica qué operaciones se han observado.
cookie	Un número que rastrea el movimiento de un fichero a otro directorio (igual a 0 si no se ha movido nada)
len	Marca de longitud para el siguiente campo.
name	Nombre del fichero sobre el cual se ha ejecutado la operación (opcional).

Tabla 1. Evento de inotify

Dada su simplicidad y eficiencia, al estar integrado directamente en el núcleo del sistema, inotify se suele utilizar para aplicaciones que requieran actuación inmediata al haber cambios en el sistema de ficheros, sin el coste de estar revisando el árbol de directorios periódicamente. Por ejemplo, se puede utilizar para indizar un sistema de forma incremental, para notificar a una aplicación de que un archivo de configuración ha sido notificado, o para actualizar la vista de un explorador de ficheros.

Sin embargo, esta solución no es apropiada para el problema en cuestión por los siguientes motivos, entre otros posibles:

- La información que muestra es escasa: No dice nada sobre el proceso y el usuario que ejecuta la operación, ni sobre la cantidad de información transmitida.
- No permite la observación recursiva de directorios, tenemos que crear una nueva instancia por cada directorio que queramos observar. Esto es inviable en un entorno con muchos usuarios y donde se crean y eliminan directorios continuamente, debido al consumo de memoria de cada nuevo proceso y la necesidad de estar vigilando el árbol de directorios constantemente para decidir si hay que crear nuevas instancias de inotify.
- No ofrece ningún sistema para registrar las operaciones, por lo que en todo caso se podría inotify como base para crear una solución apropiada, pero nunca como único componente.

2.1.2. LoggedFS

LoggedFS es un sistema de ficheros virtual implementado en FUSE que registra las llamadas que se hacen a través de él. Al ser un sistema virtual, es el sistema de ficheros subyacente el que se encarga de almacenar y gestionar los datos. LoggedFS permite registrar cualquier operación de FUSE, y nos permite configurar cuáles a través de un fichero XML. Podemos registrar o no las operaciones según el tipo de operación, el usuario que la ejecute o los ficheros sobre los que se ejecuta, y podemos usar expresiones regulares.

La información que contiene un fichero de registro de loggedFS es [2]:

- La hora de la operación.
- El tipo de operación.
- La cantidad de bytes transferidos, si aplica.
- El fichero sobre el que se ejecuta la operación.
- El resultado de la operación.
- El PID del proceso que la realiza.
- El nombre del proceso.
- El UID del usuario que ejecuta dicho proceso.

A continuación se muestra un ejemplo de la información que registra loggedFS:

```
17:29:34 (src/loggedfs.cpp:552) LoggedFS running as a public filesystem
17:29:34 (src/loggedfs.cpp:547) LoggedFS not running as a daemon
```

```
17:29:34 (src/loggedfs.cpp:666) LoggedFS starting at /var.
17:29:34 (src/loggedfs.cpp:691) chdir to /var
17:29:35 (src/loggedfs.cpp:136) getattr /var/ {SUCCESS} [ pid = 8700 kded [kdeinit] uid = 1000 ]
17:29:41 (src/loggedfs.cpp:136) getattr /var/ {SUCCESS} [ pid = 10923 ls uid = 1000 ]
17:29:41 (src/loggedfs.cpp:136) getattr /var/run {SUCCESS} [ pid = 10923 ls uid = 1000 ]
17:29:41 (src/loggedfs.cpp:136) getattr /var/run/nscd {FAILURE} [ pid = 10923 ls uid = 1000 ]
17:29:41 (src/loggedfs.cpp:136) readdir /var/ {SUCCESS} [ pid = 10923 ls uid = 1000 ]
17:29:41 (src/loggedfs.cpp:136) getattr /var/pouak {SUCCESS} [ pid = 10923 ls uid = 1000 ]
```

LoggedFS es software libre y está publicado bajo la licencia GPL [3]. Puesto que se trata de un sistema muy sencillo y fácilmente modificable que tiene algunas de las funciones que se buscan para este proyecto, se usará de base para la elaboración del mismo.

No obstante, tiene como mínimo dos carencias que hacen que no sea una solución completa al problema que aquí se plantea:

- Las opciones de configuración son insuficientes: No permite filtrar por hora, por ejemplo, o establecer reglas.
- La información se presenta de forma poco intuitiva y no se pueden extraer directamente conclusiones de ella, como contrastar datos u obtener estadísticas del uso de sistema de ficheros.

2.2. Herramientas utilizadas

En esta sección se describen las herramientas que ya existen y que pueden ser utilizadas para el desarrollo del proyecto, a fin de minimizar la cantidad de trabajo necesario para ello.

2.2.1. FUSE

FUSE es un mecanismo para crear sistemas de ficheros virtuales sin permisos de administrador en sistemas tipo UNIX [4]. Para ello, se ejecuta todo el código relativo a las operaciones sobre el sistema de archivos en modo usuario, y las llamadas se envían a una interfaz que es la que realiza la comunicación con el núcleo. De esta forma sólo necesitamos permisos elevados una vez para instalar dicha interfaz y a partir de ahí se pueden crear y utilizar los sistemas de ficheros como usuario normal.

FUSE es especialmente apropiado para crear sistemas de archivos virtuales, los cuales no almacenan ficheros sino que proveen una capa intermedia entre el usuario y el sistema de ficheros que utiliza el núcleo. Por ello, podemos crear sistemas de ficheros que requieran operaciones de alto nivel, como hacer consultas HTTP (como por ejemplo WikipediaFS [5]) o, en este caso, registrar el uso del sistema, y reutilizar el sistema que ya tengamos instalado en nuestro disco duro para el almacenamiento físico de los archivos, como ext4 o NTFS. Esto presenta varias ventajas:

- Se evita la duplicidad de código al no tener que escribir todas las partes comunes a un sistema de ficheros cada vez que hacemos uno nuevo.
- Se puede cambiar fácilmente entre sistemas virtuales sin tener que perder datos o reiniciar el ordenador, y de esta forma tener varias posibles «vistas» de nuestros datos en función del uso que les queramos dar.
- Con FUSE es más fácil aislar nuestro sistema virtual para evitar problemas de seguridad y/o estabilidad, ya que provee una API cerrada para acceder a las partes sensibles del sistema. Por tanto, se pueden evitar situaciones como que establecer una conexión permita la ejecución remota de código con permisos elevados (como podría suceder si implementáramos esta funcionalidad a nivel de núcleo del sistema).

La implementación que se utilizará para este proyecto es la original, que consiste en un módulo para Linux que se puede cargar de forma dinámica.

A continuación se muestran las llamadas al sistema que se realizan cuando ejecutamos un comando que requiera acceso al punto en el que está montado el sistema de ficheros creado con FUSE, en este caso «ls -l /tmp/fuse»:

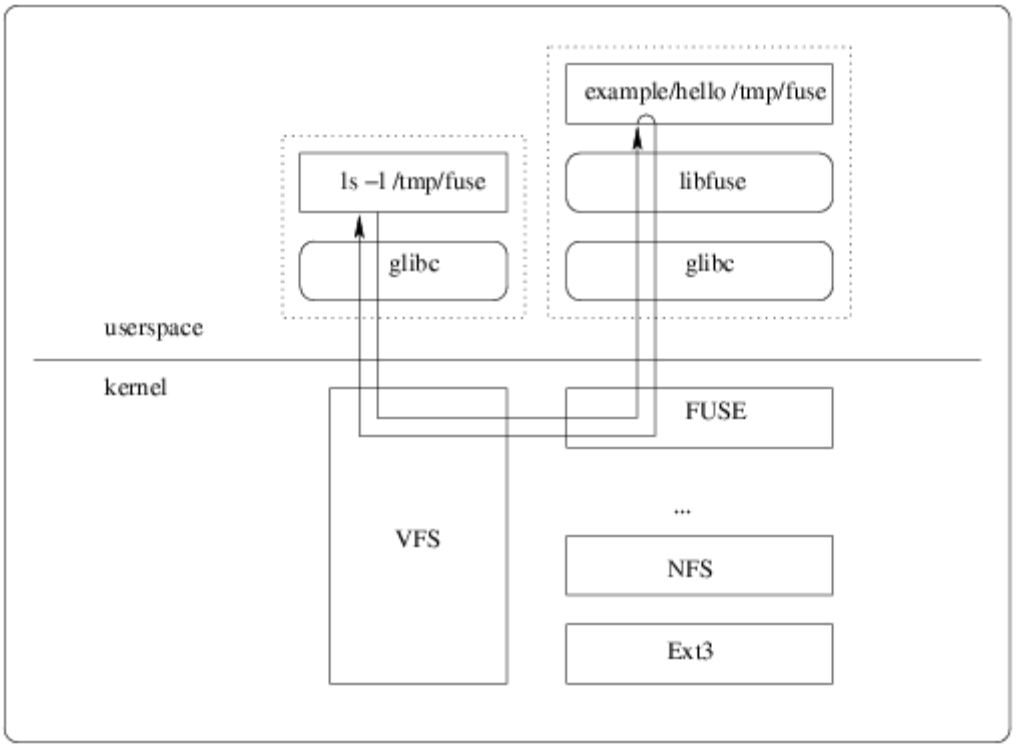

Ilustración 1. Ruta de una llamada en FUSE

2.2.2. Debian GNU/Linux

Debian es un sistema operativo libre [6] que utiliza software del proyecto GNU para la mayoría de herramientas básicas (intérprete de C, consola, etc.) y Linux como núcleo del sistema. Además, incorpora un sistema gestor de paquetes llamado Advanced Packaging Tool, o APT, que nos permite instalar software desde repositorios, que pueden ser locales, como un CD-ROM, o remotos, como un servidor web.

Entre sus múltiples ventajas, son especialmente relevantes para este proyecto las siguientes:

- Es compatible con POSIX, por lo que disponemos de mucha información sobre cómo debería comportarse un sistema de ficheros en Debian, además de que adaptarlo a otros sistemas operativos también compatibles con POSIX (que son la mayoría, excluyendo algunos sistemas privativos como Windows) es una tarea relativamente sencilla.
- La mayoría de componentes de software que integra posee una extensa documentación, ya que las directrices para la creación de paquetes de Debian especifican que debe acompañarse (como

mínimo) un manual de usuario con cada paquete, de lo contrario se considera un defecto que debe ser subsanado [7].

- Posee todas las herramientas necesarias para la elaboración del proyecto, desde los intérpretes para los distintos lenguajes de programación hasta las suites ofimáticas para la elaboración de la memoria y la gestión del tiempo.
- Es utilizado habitualmente tanto en los ordenadores de la universidad como en el equipo que se empleará para el desarrollo del proyecto, por lo que esto facilitará la implantación y las pruebas del prototipo.

2.2.3. Perl

Perl es un lenguaje de programación de propósito general concebido originalmente para facilitar la administración de sistemas tipo UNIX. La sintaxis es una mezcla entre los lenguajes de programación más populares dentro de este ámbito, como C y Bourne Shell [8].

Se trata de un lenguaje interpretado que permite mucha libertad en la sintaxis y la puntuación del código, pudiendo en muchas ocasiones omitir paréntesis, colocar palabras clave en varias posiciones distintas con el mismo efecto, etc. Además, es famoso por poder realizar tareas complejas en poco código en comparación con otros lenguajes, lo que ha dado origen a la práctica del «Perl golf», consistente en realizar una tarea concreta en el menor número de caracteres posible [9].

Entre sus principales virtudes destacan:

- La facilidad que proporciona para el procesamiento de textos. Dado que uno de sus principales propósitos iniciales era el de procesado de informes, Perl posee muchas funciones nativas para ello, lo que hace que el código sea más eficiente y compacto. Además, tiene un sistema de expresiones regulares muy completo basado en el de «sed», que ha servido como estándar para otros lenguajes durante mucho tiempo.
- Perl es capaz de comunicarse con software creado con otros lenguajes utilizando lo que se conoce como *bindings*, de forma que podemos por ejemplo llamar a una función de C desde una aplicación en Perl. Esto nos permite utilizarlo para conectar entre sí componentes dispares con poco esfuerzo.

2.2.4. SQLite

SQLite [10] es un sistema gestor de bases de datos cuya peculiaridad reside en que éstas se almacenan en un único fichero, y las consultas se lanzan directamente sobre el fichero utilizando un intérprete. La mayoría de sistemas gestores de bases de datos, por otra parte, utilizan un servidor que resuelve las consultas.

Por ello, para este caso particular, tiene ciertas ventajas sobre otros sistemas:

- Al estar ideado únicamente para bases de datos locales almacenadas en un fichero, es la solución más eficiente para este caso, con un gran rendimiento y un consumo de memoria muy reducido.
- No necesita ningún tipo de configuración o instalación, por lo que ahorra muchos problemas de implantación. Además, debido a esto las bases de datos se pueden copiar de un ordenador a otro simplemente moviendo el fichero que la contiene.
- El código es de dominio público, por lo que no existen barreras para su uso, ni a nivel legal ni económico. Otros sistemas que necesitan de una licencia costosa son inviables para la elaboración de este proyecto dado el limitado presupuesto.
- Utiliza un lenguaje SQL transaccional, por lo que todas las instrucciones podrían reutilizarse con otro DBMS en un futuro si fuera necesario.

2.2.5. GTK

GIMP Toolkit, o GTK [11], es un conjunto de librerías de código que nos permite crear interfaces gráficas para el entorno de escritorio GNOME, que ha sido históricamente uno de los entornos de escritorio gráficos más populares para GNU/Linux y otros sistemas como OpenSolaris o FreeBSD.

- Provee un marco muy completo para desarrollar interfaces gráficas, permitiendo la creación de todo tipo de elementos como botones, menús, barras de progreso, etc. Además otorga un estilo

uniforme a todas las aplicaciones, por lo que da una mayor coherencia aunque las aplicaciones se desarrollen independientemente, y evita tener que crear temas e iconos nuevos.

- Existen varios módulos para utilizar GTK desde Perl [12] que simplifican mucho el trabajo. Además las librerías de GTK están implementadas en su mayoría en C por lo que tienen un bajo consumo de memoria y un tiempo de respuesta muy bajo.
- GNOME es el entorno de escritorio que viene por defecto en Debian y el que está instalado en los equipos de desarrollo y de pruebas para este proyecto. Por tanto, se puede utilizar GTK directamente sin necesidad de instalar dependencias adicionales.

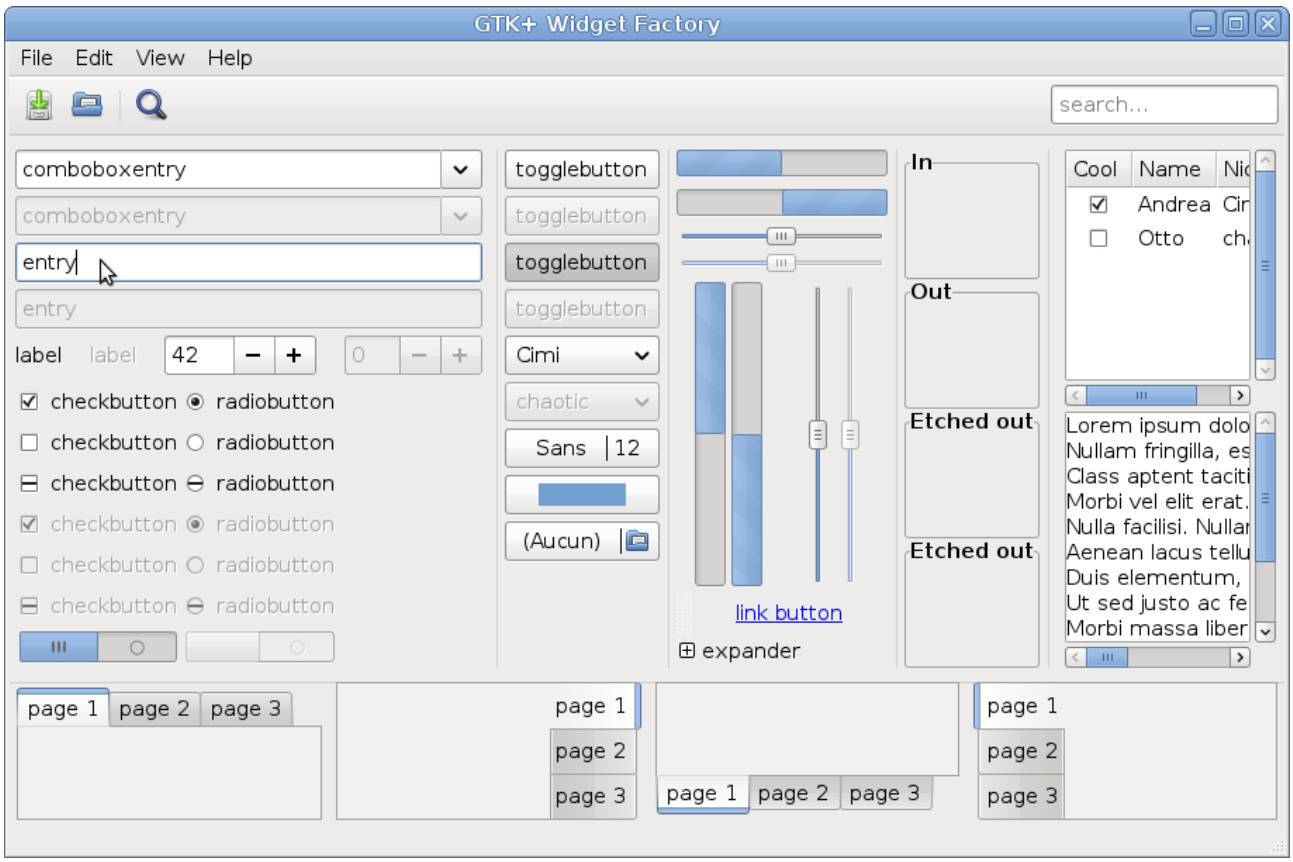

Ilustración 2. Ejemplo de interfaz gráfica en GTK

2.2.6. D3js

D3js (acrónimo de *data-driven documents* y JavaScript) [13] es una librería de JavaScript diseñada para construir gráficos y formularios dinámicos a partir de conjuntos de datos. En este proyecto se utilizará para presentar los informes generados a partir de la información que se ha recopilado del uso del sistema.

Los principales motivos que han llevado a la elección de esta librería y no otra son:

- Está publicada bajo la licencia BSD de 3 cláusulas [14], la cual es una licencia de software libre permisiva que nos permite hacer prácticamente lo que queramos excepto utilizar el nombre del autor o autores sin su consentimiento. Por tanto, se puede incluir en el proyecto sin ningún tipo de problema.
- Proporciona un alto grado de flexibilidad en relación a otras librerías de gráficos para JavaScript, las cuales por lo general disponen de un conjunto limitado de gráficos predefinidos. Con D3js podemos crear nuestros propios gráficos personalizados.

- D3js, dado que está fuertemente orientada al uso de ficheros de datos como base de los informes, permite importar de forma sencilla datos de ficheros de texto formateados, como por ejemplo ficheros CSV, los cuales se pueden generar de forma automática.
- Al utilizar JavaScript, los gráficos se podrán incrustar en un documento web y visualizarse en cualquier navegador convencional. De esta forma no tenemos que preocuparnos del formato de los informes y de si disponemos de la aplicación necesaria para visualizarlos. Además de esta forma los gráficos pueden ser interactivos, y permitir que el usuario altere la información que se muestra de diversas formas.

A continuación se muestra un ejemplo de un gráfico de barras hecho con D3js [15]. Aunque no puede apreciarse bien en la imagen, el gráfico es interactivo y nos permite acotar el rango de fechas que queremos mostrar en los tres gráficos superiores mediante unos tiradores colocados en el gráfico inferior.

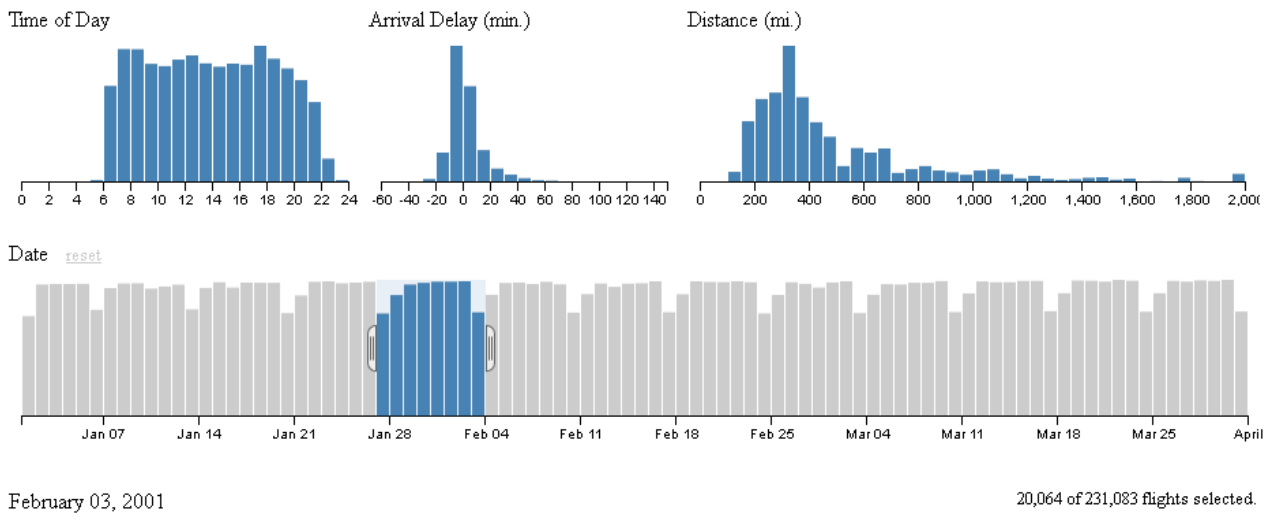

Ilustración 3. Ejemplo de un gráfico interactivo en d3js

2.2.7. Eclipse

Eclipse [16] es un entorno de desarrollo integrado multilenguaje y extensible utilizado ampliamente para el desarrollo de aplicaciones, tanto a nivel empresarial como personal o académico. Al tener una arquitectura modular y ser software libre, existen innumerables modificaciones y entornos basados en Eclipse para el desarrollo con distintas tecnologías, como por ejemplo IBM Rational [17] u Oracle Workshop [18]. Para este proyecto se utilizará el Eclipse disponible en los repositorios de Debian, junto con los paquetes WDT (para el desarrollo de aplicaciones web) y EPIC (para la integración con Perl).

Algunas de las principales ventajas de Eclipse son:

- Posibilidad de combinar distintos lenguajes en un solo entorno. En este caso concreto, podemos tener pestañas abiertas con código HTML, JavaScript y Perl en la misma sesión y tenemos resaltado de sintaxis, autocompletado, etc. para cada uno de estos lenguajes.
- Provee todas las herramientas auxiliares que podemos necesitar para el desarrollo, como por ejemplo un intérprete de comandos, un navegador web, un depurador o un explorador de bases de datos.
- Eclipse comprueba la sintaxis del código según lo escribimos y nos advierte de los errores que encuentra, por lo que nos ahorramos tener que probar a ejecutar el código varias veces para encontrar errores como sucede con los editores de texto normales.

A continuación se muestra una captura de pantalla de una ventana de Eclipse donde pueden apreciarse algunas de las características descritas:

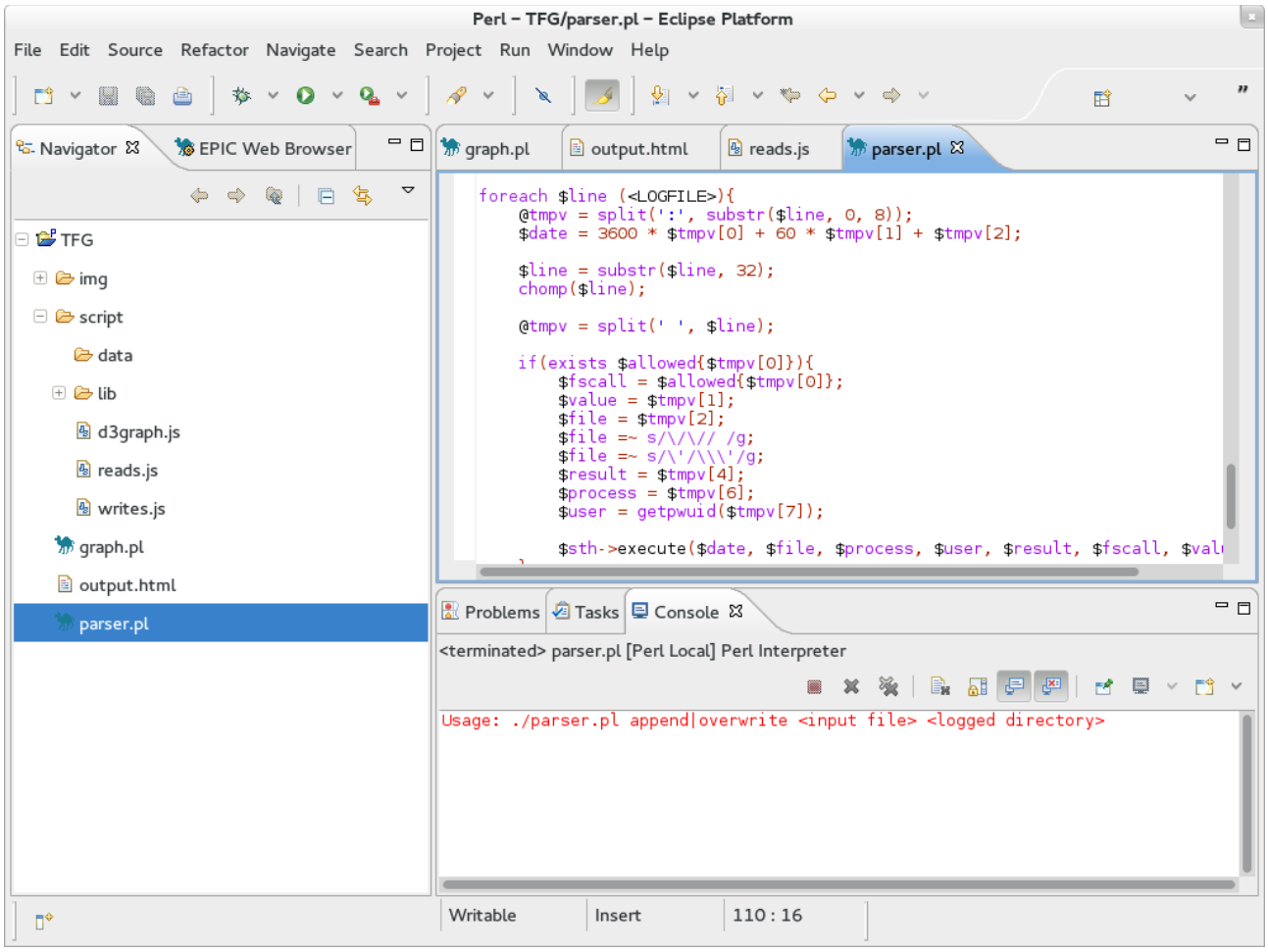

Ilustración 4. Ejemplo de una ventana de Eclipse

2.3. Marco regulador

Puesto que el proyecto trata sobre el registro de información generada por usuarios, cabe tener en cuenta la legislación vigente al respecto. En concreto, la ley que concierne a este proyecto es la Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal [19]. Dicha ley « […] tiene por objeto garantizar y proteger, en lo que concierne al tratamiento de los datos personales, las libertades públicas y los derechos fundamentales de las personas físicas, y especialmente de su honor e intimidad personal y familiar».

Según el apartado a) de la sección 2 del artículo 2 de dicha ley, ésta no será de aplicación «a los ficheros mantenidos por personas físicas en el ejercicio de actividades exclusivamente personales o domésticas». Por tanto, para el desarrollo del proyecto no se aplicaría esta ley, puesto que no se tratan los datos de personas ajenas a éste ni se va a implantar el sistema fuera del ámbito doméstico.

No obstante, si se quisiera instalar el sistema de ficheros en un entorno con diversos usuarios, como la universidad, sí que habría que tener en cuenta la ley y posiblemente modificar la información registrada para que sea anónima. Por ello se intentará en la medida de lo posible facilitar esta labor en la elaboración del prototipo.

3. Desarrollo de la solución

3.1 Metodología

La metodología utilizada para el desarrollo de este proyecto corresponde a un proceso secuencial en cascada, en el que cada paso depende de haber finalizado el anterior y podemos trazar la existencia de un componente en cualquier fase a su correspondiente en fases anteriores. Esto nos permite asegurar, por ejemplo, que no se haya introducido una funcionalidad que no era requerida en un principio, o viceversa.

A continuación se muestra un esquema con las fases principales de la elaboración del proyecto. Cabe tener en cuenta que en este caso no hay periodo de mantenimiento pues la aplicación es un prototipo que no va a ser implantado en un entorno de producción.

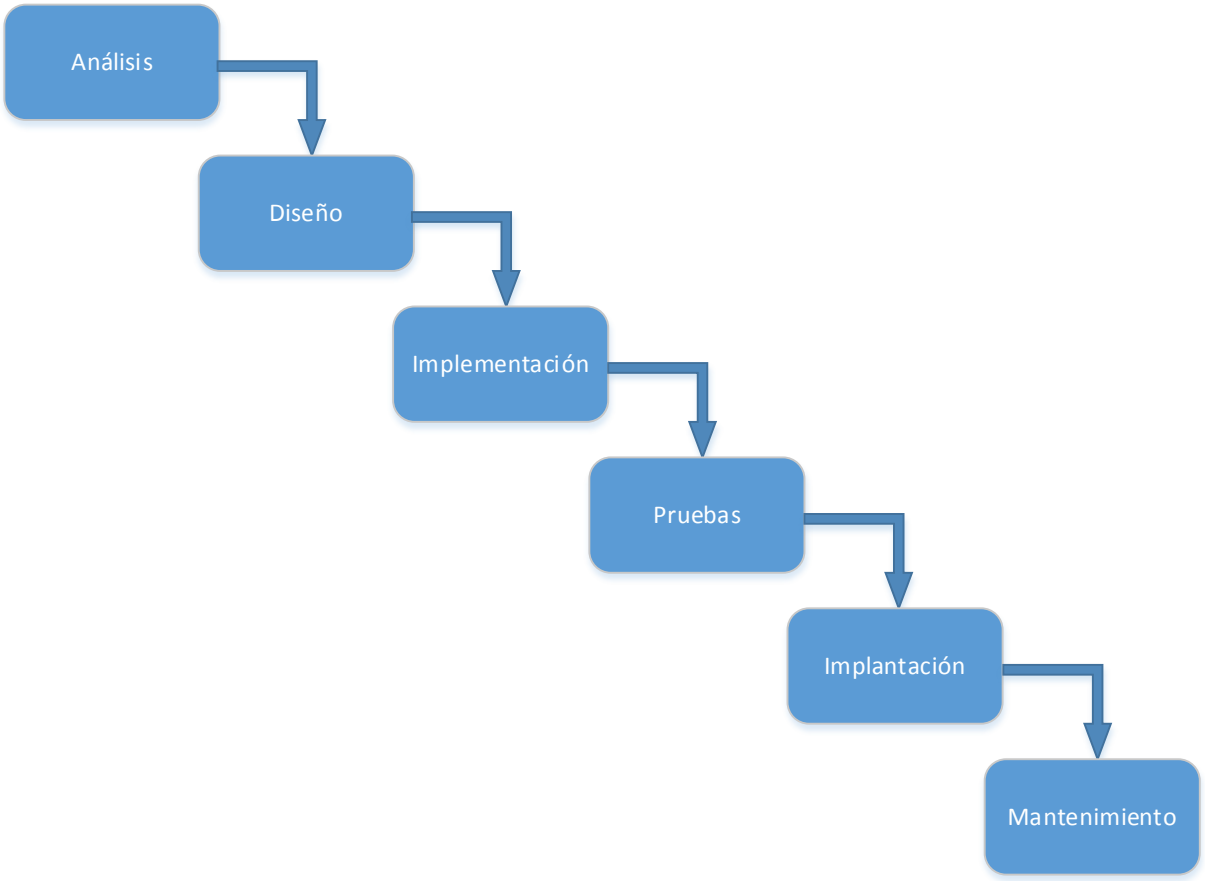

Ilustración 5. Fases de una metodología en cascada

La principal alternativa al modelo en cascada son las metodologías ágiles, que se basan ir documentando el software según se implementa, permitiendo cambios frecuentes y correcciones sobre el planteamiento original. Las principales críticas al modelo en cascada son:

- Resulta muy difícil tener una parte del proyecto terminada y sin errores antes de comenzar con la siguiente. Por tanto, puede ser beneficioso no bloquear el desarrollo hasta que una etapa esté perfecta y en su lugar seguir adelante con lo que haya y, si es necesario, realizar correcciones según surjan los problemas.
- El modelo en cascada dificulta la inclusión de modificaciones en el sistema cuando se encuentra en una fase avanzada, como por ejemplo un cambio de requisitos debido a una nueva necesidad cuando ya se está implementando, o una corrección en el diseño cuando se están realizando las pruebas.

Sin embargo, pese a estos inconvenientes, dadas las características de este proyecto esta metodología se considera la más adecuada. Los principales motivos para ello son:

- Cada parte del desarrollo es independiente de la siguiente. Por ello, podemos dejar las últimas fases incompletas o prescindir de ellas. Esto es importante porque la fase de mantenimiento no se lleva a cabo y la implementación no tiene por qué estar completa, al tratarse de un prototipo que se presenta fundamentalmente como prueba de concepto. En cambio, el análisis y el diseño sí tienen que quedar completos. En una metodología ágil no puede haber esta discordancia entre el análisis y diseño y la implementación, al realizarse en paralelo.
- Todo el desarrollo queda trazado de principio a final, de tal forma que no pueden surgir partes o funcionalidades nuevas de la aplicación que no hayan sido contempladas en el diseño. De cara al cliente esto nos permitiría también poder justificar el que se haya implementado una funcionalidad concreta y cómo se ha hecho.
- Esta metodología simplifica mucho el trabajo y elimina mucho margen de error si la primera fase de todas, el análisis, se realiza debidamente, por lo que funciona mejor en entornos donde los requisitos son inmutables y están definidos de forma que el diseñador los interprete correctamente, como es el caso.

3.2. Análisis

En esta sección se realiza un estudio de la funcionalidad que ha de tener el sistema, sin entrar en detalles sobre cómo debe estar estructurado y los componentes que debe tener. Para ello, se utilizan los modelos de casos de uso – más generales – y requisitos – más concretos. Además, se proporciona una matriz de trazabilidad que indica en qué caso de uso se origina cada requisito.

3.2.1. Casos de uso

Los casos de uso representan escenarios de interacción entre la aplicación y sus potenciales usuarios. Son el mayor nivel de abstracción de la funcionalidad y características que ha de tener, por tanto son muy generales y no tienen en cuenta cómo se ha de producir la interacción.

3.2.1.1. Formato

Para definir un caso de uso se deben establecer los siguientes atributos:

- **Código:** un identificador unívoco para el caso de uso. El formato que se seguirá es CU-*n*, donde *n* es un número natural que empieza en 0 y se incrementa en 1 con cada caso de uso subsiguiente.
- **Nombre:** una frase breve que describe en términos generales a qué hace referencia el caso de uso.
- **Actores:** el conjunto de usuarios a los que aplica el caso de uso. Los actores se definen previamente y no se puede hacer referencia a un actor que no haya sido definido.
- **Precondiciones:** una descripción del contexto en el que puede tener lugar el caso de uso. Si no se cumplen las precondiciones, el caso de uso no es aplicable a esa situación.
- **Escenario básico:** una descripción de cómo transcurre el caso de uso en circunstancias normales.
- **Escenario(s) alternativo(s):** una descripción de cómo transcurre el caso de uso en circunstancias excepcionales y distintas a las del escenario básico. Este campo es opcional.
- **Postcondiciones:** una descripción de las modificaciones que se producen en el sistema como consecuencia de llevar a cabo el caso de uso.

3.2.1.2. Actores

A continuación se definen los distintos actores que pueden interactuar con el sistema:

- **Usuario:** un usuario normal que utiliza el sistema de ficheros y obtiene datos de él, pero que no tiene necesariamente permisos para ponerlo en marcha en primer lugar, ni es su responsabilidad.
- **Administrador:** un usuario que se encarga de la instalación y configuración del sistema y al que se permite controlar todos los aspectos de la ejecución de éste y de la información que se genera.

Cabe destacar que el concepto de actor no se refiere a persona física sino a un rol en el uso de la aplicación, y por tanto una misma persona puede tomar el papel de uno u otro dependiendo del uso que quiera darle a la misma. Sin embargo, la separación es necesaria en un entorno con múltiples usuarios donde no todos tengan permiso para desempeñar ambas funciones.

3.2.1.3. Listado de casos de uso

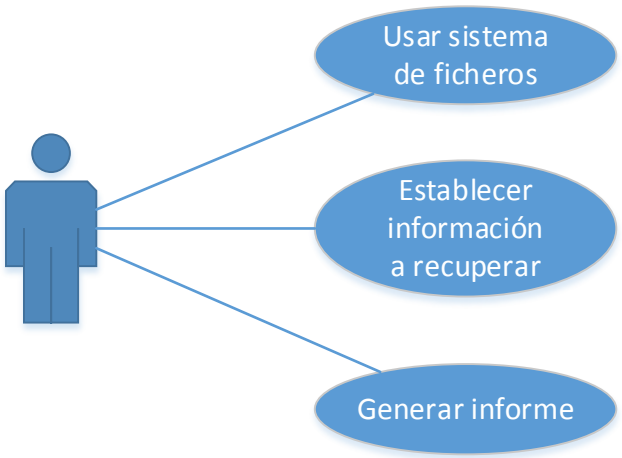

Ilustración 6. Casos de uso de un usuario normal

Código	CU-1
Nombre	Usar el sistema de ficheros.
Actores	Usuario.
Precondiciones	1. Tener montado el sistema de ficheros.
Escenario básico	1. Se accede al directorio en el cual se quiere efectuar la operación. 2. Se realiza la operación.

Postcondiciones	1. La operación queda realizada correctamente.

Tabla 2. CU-1

Código	CU-2
Nombre	Configurar qué información se quiere recuperar.
Actores	Usuario.
Precondiciones	N/A
Escenario básico	1. Se seleccionan qué datos se quieren recuperar de entre las opciones disponibles.
Postcondiciones	1. Las opciones seleccionadas quedan guardadas. 2. Estas opciones se le proporcionan al sistema que genera los informes.

Tabla 3. CU-2

Código	CU-3
Nombre	Generar informe de uso.
Actores	Usuario.
Precondiciones	1. Tener información previa del uso del sistema. 2. Haber establecido qué información se quiere recuperar.
Escenario básico	1. Se selecciona el conjunto de datos que se quiera emplear para generar el informe. 2. Se pide a la aplicación que genere un informe.
Postcondiciones	1. Se genera un informe con los datos deseados.

Tabla 4. CU-3

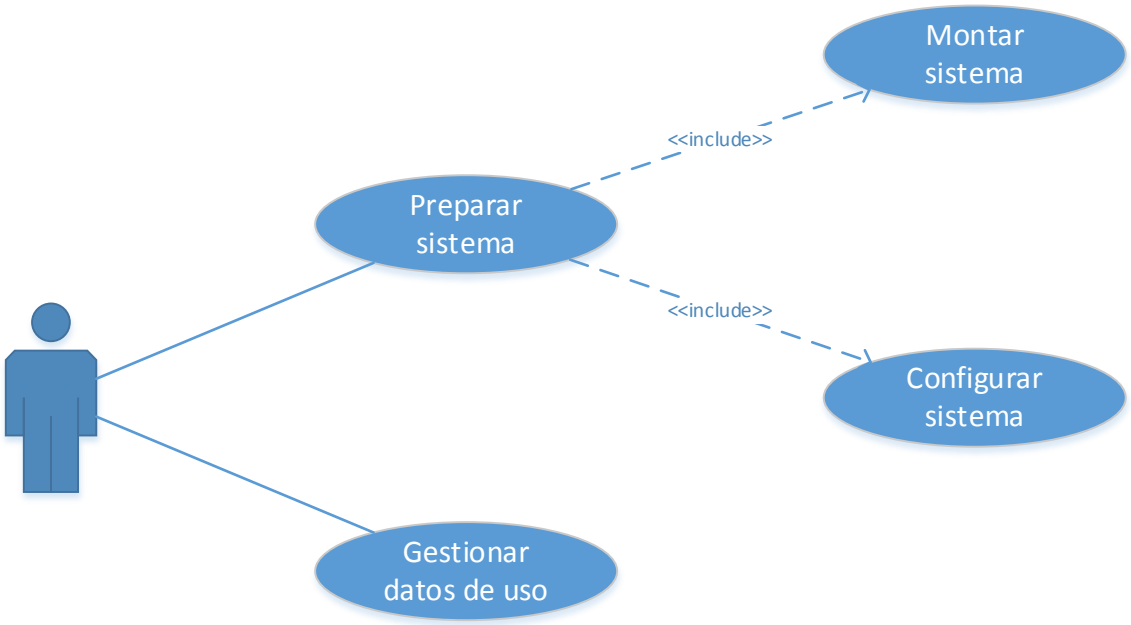

Ilustración 7. Casos de uso de un administrador

Código	CU-4
Nombre	Preparar sistema
Actores	Usuario.
Precondiciones	N/A
Escenario básico	1. Elegir un punto de montaje. 2. Establecer opciones de montaje. 3. Montar el sistema.
Escenario alternativo	1. Seleccionar un sistema ya montado. 2. Cambiar la configuración del sistema.
Postcondiciones	1. El sistema queda listo para su uso.

Tabla 5. CU-4

Código	CU-5
Nombre	Guardar datos de uso
Actores	Usuario.
Precondiciones	1. Tener datos del uso que se le ha dado al sistema en la sesión actual.
Escenario básico	1. Se selecciona una base de datos ya existente. 2. Se pide a la aplicación que guarde los datos de la sesión.
Escenario alternativo	1. Se crea una nueva base de datos. 2. Se pide a la aplicación que guarde los datos de la sesión.
Escenario alternativo	1. Se selecciona una base de datos ya existente. 2. Se selecciona la base de datos donde se quieran importar los datos. 3. Se pide a la aplicación que importe los datos.
Postcondiciones	1. Los datos quedan guardados en la base de datos seleccionada. 2. La información queda seleccionada para utilizarse en la generación de un informe.

Tabla 6. CU-5

3.2.2 Requisitos

Los requisitos representan características concretas que debe cumplir la aplicación, siendo más específicos y cubriendo cada uno un ámbito menor que los casos de uso.

3.2.2.1 Formato

Dependiendo de si se refiere a la funcionalidad que debe tener la aplicación o no, podemos dividir los requisitos en dos clases principales:

- **Funcionales:** El requisito especifica algo que debe hacer la aplicación, lo cual se extrae directamente de los casos de uso como una necesidad para que éstos puedan llevarse a cabo.
- **No funcionales:** El requisito especifica alguna restricción que debe cumplir la aplicación que no afecte directamente a la funcionalidad, como por ejemplo niveles de rendimiento, usabilidad o adecuación a un estándar.

Para cada requisito se establecen los siguientes atributos:

- **Código:** Un identificador único para el requisito. En el caso de los requisitos funcionales, será RF-*n*, y en el caso de los no funcionales, RNF-*n*, donde *n* es un número natural secuencial que empieza en 1 y se incrementa con cada requisito.
- **Descripción:** Una explicación precisa del requisito.
- **Estabilidad:** La medida en la que el requisito puede variar a lo largo del desarrollo de la propuesta. Valores posibles:
    - Alta: El requisito no va a variar, o lo va a hacer de forma que no se altere el componente del diseño que lo implemente. Por ejemplo, se modifica la redacción para una mayor claridad.
    - Media: Se espera que el requisito pueda variar ligeramente, de forma que impacte sólo en el componente del diseño que lo implemente.
    - Baja: Se espera que el requisito tenga un grado de variabilidad moderado y puede que haya que modificar algunos aspectos de la arquitectura del sistema.

- **Prioridad:** El orden en el que debe ser implementado el requisito. Los requisitos con prioridad alta deberán implementarse antes que los requisitos con prioridad baja. Valores posibles:
    - Alta: El requisito debe ser implementado antes que aquellos que tengan prioridad media o baja.
    - Media: El requisito debe ser implementado antes que aquellos que tengan prioridad baja, pero no antes que aquellos con prioridad alta.
    - Baja: El requisito no debe ser implementado antes que aquellos que tengan prioridad media o alta.
- **Necesidad:** Indica si el requisito es indispensable o no para que la aplicación cumpla su propósito. Valores posibles:
    - Esencial: El requisito es necesario.
    - Opcional: El requisito no es necesario, aunque sí deseable.
- **Origen:** Indica el caso de uso en el cual tiene origen el requisito.

3.2.2.2. Requisitos funcionales

Código	RF-1		
Descripción	El sistema debe permitir la creación de ficheros.		
Estabilidad	Alta	**Prioridad**	Alta
Necesidad	Esencial	**Origen**	CU-1

Tabla 7. RF-1

Código	RF-2		
Descripción	El sistema debe permitir la eliminación de ficheros.		
Estabilidad	Alta	**Prioridad**	Alta

Necesidad	Esencial	**Origen**	CU-1

Tabla 8. RF-2

Código	RF-3		
Descripción	El sistema debe permitir la lectura de ficheros.		
Estabilidad	Alta	**Prioridad**	Alta
Necesidad	Esencial	**Origen**	CU-1

Tabla 9. RF-3

Código	RF-4		
Descripción	El sistema debe permitir la escritura de ficheros.		
Estabilidad	Alta	**Prioridad**	Alta
Necesidad	Esencial	**Origen**	CU-1

Tabla 10. RF-4

Código	RF-5		
Descripción	El sistema debe permitir establecer el usuario propietario de un fichero.		
Estabilidad	Alta	**Prioridad**	Alta
Necesidad	Esencial	**Origen**	CU-1

Tabla 11. RF-5

Código	RF-6

Descripción	El sistema debe permitir establecer el grupo propietario de un fichero.		
Estabilidad	Alta	**Prioridad**	Alta
Necesidad	Esencial	**Origen**	CU-1

Tabla 12. RF-6

Código	RF-7		
Descripción	El sistema debe permitir modificar los permisos de un fichero.		
Estabilidad	Alta	**Prioridad**	Alta
Necesidad	Esencial	**Origen**	CU-1

Tabla 13. RF-7

Código	RF-8		
Descripción	El sistema debe permitir establecer la máscara que se aplicará a los permisos de nuevos ficheros.		
Estabilidad	Alta	**Prioridad**	Media
Necesidad	Opcional	**Origen**	CU-4

Tabla 14. RF-8

Código	RF-9		
Descripción	El sistema debe permitir la creación de enlaces simbólicos.		
Estabilidad	Alta	**Prioridad**	Media
Necesidad	Opcional	**Origen**	CU-1

Tabla 15. RF-9

Código	RF-10		
Descripción	El sistema debe registrar las operaciones que se realicen sobre el árbol de directorios completo del sistema montado.		
Estabilidad	Alta	**Prioridad**	Alta
Necesidad	Esencial	**Origen**	CU-3

Tabla 16. RF-10

Código	RF-11		
Descripción	El sistema debe registrar la hora de la operación.		
Estabilidad	Alta	**Prioridad**	Media
Necesidad	Esencial	**Origen**	CU-2

Tabla 17. RF-11

Código	RF-12		
Descripción	El sistema debe registrar la ruta del fichero sobre el que se realiza la operación.		
Estabilidad	Alta	**Prioridad**	Alta
Necesidad	Esencial	**Origen**	CU-2

Tabla 18. RF-12

Código	RF-13

Descripción	El sistema debe registrar el tamaño del fichero sobre el que se realiza la operación.		
Estabilidad	Alta	**Prioridad**	Media
Necesidad	Esencial	**Origen**	CU-2

Tabla 19. RF-13

Código	RF-14		
Descripción	El sistema debe registrar el proceso que realiza la operación.		
Estabilidad	Alta	**Prioridad**	Media
Necesidad	Esencial	**Origen**	CU-2

Tabla 20. RF-14

Código	RF-15		
Descripción	El sistema debe registrar el usuario que realiza la operación.		
Estabilidad	Alta	**Prioridad**	Media
Necesidad	Esencial	**Origen**	CU-2

Tabla 21. RF-15

Código	RF-16		
Descripción	El sistema debe registrar el resultado de la operación.		
Estabilidad	Alta	**Prioridad**	Baja
Necesidad	Opcional	**Origen**	CU-2

Tabla 22. RF-16

Código	RF-17		
Descripción	El sistema debe registrar el tipo de operación que se realiza.		
Estabilidad	Alta	**Prioridad**	Alta
Necesidad	Esencial	**Origen**	CU-2

Tabla 23. RF-17

Código	RF-18		
Descripción	El sistema debe registrar la cantidad de información transferida, si es aplicable a la operación.		
Estabilidad	Alta	**Prioridad**	Media
Necesidad	Opcional	**Origen**	CU-2

Tabla 24. RF-18

Código	RF-19		
Descripción	El sistema debe poder montarse en un árbol de directorios existente.		
Estabilidad	Alta	**Prioridad**	Alta
Necesidad	Esencial	**Origen**	CU-4

Tabla 25. RF-19

Código	RF-20
Descripción	El sistema no debe sobrescribir la estructura de ficheros que haya en el punto de montaje.

Estabilidad	Alta	**Prioridad**	Alta
Necesidad	Esencial	**Origen**	CU-4

Tabla 26. RF-20

Código	RF-21		
Descripción	Los cambios en el sistema de ficheros virtual deben preservarse al desmontarse.		
Estabilidad	Alta	**Prioridad**	Alta
Necesidad	Esencial	**Origen**	CU-4

Tabla 27. RF-21

Código	RF-22		
Descripción	El sistema debe poder generar gráficas a partir de la información de uso.		
Estabilidad	Baja	**Prioridad**	Alta
Necesidad	Esencial	**Origen**	CU-3

Tabla 28. RF-22

Código	RF-23		
Descripción	El sistema debe permitir establecer qué operaciones se deben registrar de entre todas las posibles.		
Estabilidad	Alta	**Prioridad**	Baja
Necesidad	Opcional	**Origen**	CU-4

Tabla 29. RF-23

Código	RF-24

Descripción	El sistema debe permitir parar y reanudar el registro sin montar y desmontar el sistema de ficheros.		
Estabilidad	Alta	**Prioridad**	Baja
Necesidad	Opcional	**Origen**	CU-4

Tabla 30. RF-24

Código	RF-25		
Descripción	El usuario debe poder elegir qué información se recupera para la generación del informe.		
Estabilidad	Media	**Prioridad**	Alta
Necesidad	Esencial	**Origen**	CU-2

Tabla 31. RF-25

Código	RF-26		
Descripción	El usuario debe poder elegir qué información se contrasta en los gráficos, seleccionando los conjuntos de datos deseados para los ejes del mismo.		
Estabilidad	Media	**Prioridad**	Baja
Necesidad	Opcional	**Origen**	CU-2

Tabla 32. RF-26

Código	RF-27		
Descripción	La información recogida debe poder almacenarse en una base de datos.		
Estabilidad	Alta	**Prioridad**	Alta
Necesidad	Esencial	**Origen**	CU-5

Tabla 33. RF-27

Código	RF-28		
Descripción	El sistema debe permitir la creación de una base de datos con el esquema necesario para almacenar la información.		
Estabilidad	Alta	**Prioridad**	Media
Necesidad	Esencial	**Origen**	CU-5

Tabla 34. RF-28

Código	RF-29		
Descripción	El sistema debe permitir importar información de una base de datos creada anteriormente con la aplicación.		
Estabilidad	Media	**Prioridad**	Media
Necesidad	Opcional	**Origen**	CU-5

Tabla 35. RF-29

3.2.2.3. Requisitos no funcionales

Código	RNF-1		
Descripción	Toda la funcionalidad debe ser accesible a través de una interfaz gráfica, exceptuando el uso del sistema de ficheros.		
Estabilidad	Media	**Prioridad**	Baja
Necesidad	Opcional	**Origen**	CU-2,3,4,5

Tabla 36. RNF-1

Código	RNF-2

Descripción	Los permisos de los ficheros deben seguir el estándar IEEE 1003 de permisos para ficheros [20].		
Estabilidad	Alta	**Prioridad**	Alta
Necesidad	Esencial	**Origen**	CU-1

Tabla 37. RNF-2

Código	RNF-3		
Descripción	El texto en la base de datos debe estar codificado en UTF-8 sin marca de ordenación.		
Estabilidad	Alta	**Prioridad**	Alta
Necesidad	Esencial	**Origen**	CU-5

Tabla 38. RNF-3

Código	RNF-4		
Descripción	Los informes generados deben ser documentos HTML5 válidos.		
Estabilidad	Alta	**Prioridad**	Media
Necesidad	Esencial	**Origen**	CU-3

Tabla 39. RNF-4

Código	RNF-5		
Descripción	El sistema de ficheros podrá ser utilizado por distintos usuarios de forma simultánea.		
Estabilidad	Alta	**Prioridad**	Media
Necesidad	Esencial	**Origen**	CU-1

Tabla 40. RNF-5

Código	RNF-6		
Descripción	La base de datos deberá estar protegida frente a ataques de inyección SQL.		
Estabilidad	Alta	**Prioridad**	Alta
Necesidad	Esencial	**Origen**	CU-5

Tabla 41. RNF-6

Código	RNF-7		
Descripción	No se realizarán operaciones sobre el sistema de ficheros sin utilizar las funciones de FUSE.		
Estabilidad	Alta	**Prioridad**	Alta
Necesidad	Esencial	**Origen**	CU-1

Tabla 42. RNF-7

3.2.2.4 Matriz de trazabilidad

En la siguiente tabla se muestra la correspondencia entre requisitos del sistema y el caso de uso al cual se aplica. Los requisitos no funcionales, por tratarse de rasgos generales de la aplicación, pueden corresponder a más de un caso de uso; los requisitos funcionales corresponden a un solo caso de uso.

	CU-1	CU-2	CU-3	CU-4	CU-5
RF-1	X				
RF-2	X				
RF-3	X				
RF-4	X				
RF-5	X				
RF-6	X				
RF-7	X				
RF-8				X	
RF-9	X				
RF-10			X		
RF-11		X			

RF-12		X			
RF-13		X			
RF-14		X			
RF-15		X			
RF-16		X			
RF-17		X			
RF-18		X			
RF-19				X	
RF-20				X	
RF-21				X	
RF-22			X		
RF-23				X	
RF-24				X	
RF-25		X			
RF-26		X			
RF-27					X
RF-28					X
RF-29					X
RNF-1		X	X	X	X
RNF-2	X				
RNF-3					X
RNF-4			X		
RNF-5	X				
RNF-6					X
RNF-7	X				

Tabla 43. Matriz de trazabilidad de requisitos

3.3. Diseño

En esta sección se realiza una descripción detallada de los componentes que ha de tener la aplicación, con objeto de concretar los requisitos de usuario y eliminar ambigüedades para la fase de implementación. En términos generales, el sistema constará de las siguientes partes:

- Una interfaz gráfica que permita el acceso a toda la funcionalidad requerida por el usuario. Ésta se encargará de enviar las instrucciones de montar y desmontar el sistema de ficheros, así como establecer su configuración, enviar las instrucciones de generación de informes y las de carga y guardado de datos en la base de datos.
- Un sistema de ficheros virtual que se encargue de comunicarse mediante FUSE con el sistema de ficheros nativo para realizar todas las operaciones que se requieren, y de registrar aquellas que desee el usuario. El registro se hará a un fichero temporal, para simplificar el proceso y evitar la inserción de datos no deseados en la base de datos.
- Un módulo controlador de base de datos que se encarga de procesar los datos del fichero de registro e insertarlos en una base de datos, además de la creación de nuevas bases de datos con el esquema adecuado y la recuperación de datos de una base de datos existente para generar los informes.
- Un módulo generador de informes encargado de recoger la información del controlador de base de datos y procesarla para presentársela al usuario según la configuración que haya escogido. Los informes se presentarán como documentos web que el usuario podrá visualizar directamente en un navegador.

A continuación se muestra un diagrama con una visión general de la arquitectura del sistema, indicando el flujo de datos entre los distintos componentes de la aplicación.

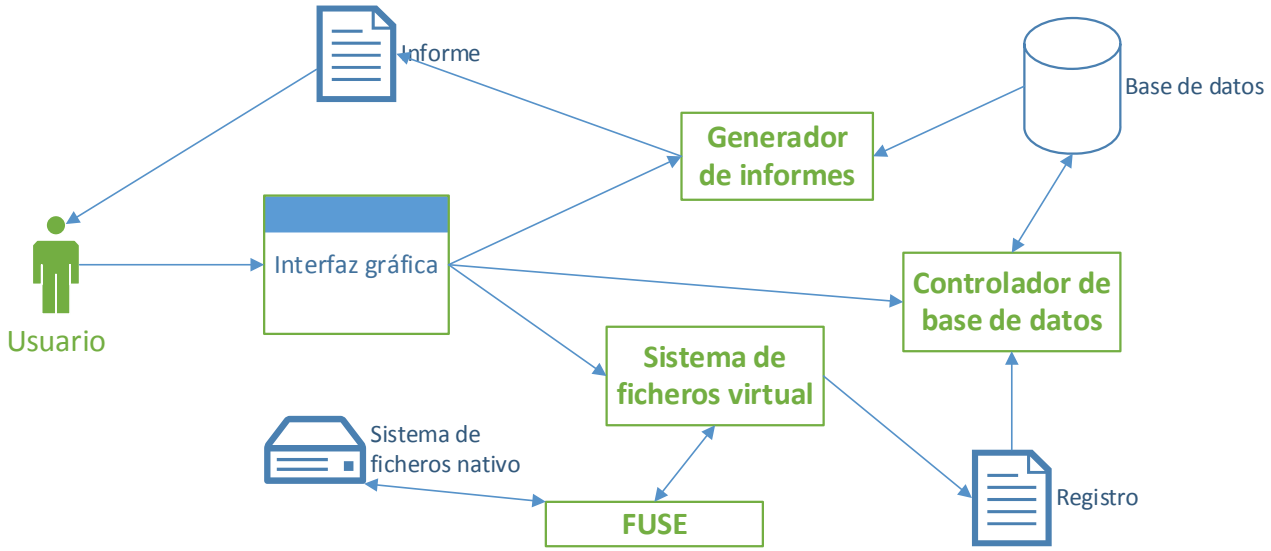

Ilustración 8. Arquitectura general del sistema

3.3.1. Modelo de datos

Para almacenar los datos de uso de forma persistente se utiliza una base de datos relacional. Dicha base de datos tiene el siguiente esquema:

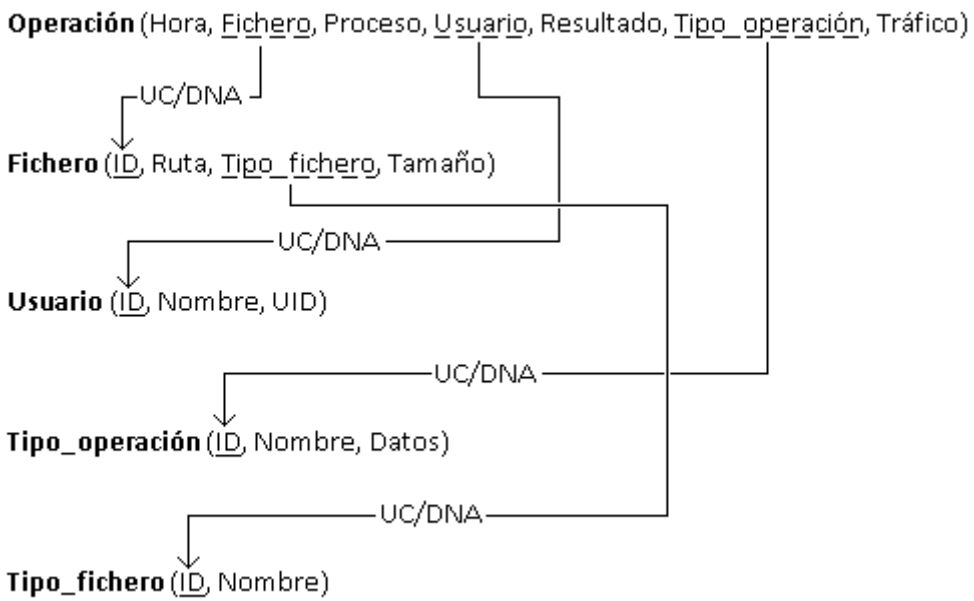

Ilustración 9. Esquema relacional

Todas las operaciones de actualización de datos se realizan en cascada (*on update cascade* o UC) para permitir la actualización del sistema de identificadores en un futuro. El comportamiento por defecto para

todas las operaciones de borrado sobre una clave ajena es de no permitir la operación (*on delete no action* o DNA), y sería necesario borrar primero la entrada a la que se hace referencia para preservar la integridad de la base de datos. Esta política se sigue para minimizar el riesgo de pérdida de datos.

A continuación se describen los campos que contiene cada tabla en detalle.

3.3.1.1. Tabla operación

En esta tabla se almacenan todas las operaciones que se han realizado sobre el sistema de ficheros.

Campo	Tipo	Descripción
Hora	Número entero	El segundo del día en el que se ha producido la operación. Rango: 0 – 194399
Fichero	Número entero	Clave ajena para el campo *ID* de la tabla *fichero*.
Proceso	Texto	El nombre del proceso que realiza la operación.
Usuario	Número entero	Clave ajena para el campo *ID* de la tabla *usuario*.
Resultado	Número entero	El resultado que devuelve la operación. 0 implica que la operación se ha realizado con éxito, el resto de valores distintos errores. Rango:0-255
Tipo_operación	Número entero	Clave ajena para el campo *ID* de la tabla *tipo_operación*.
Tráfico	Número entero	La cantidad de bytes que se han transferido con la operación, si aplica.

Tabla 44. Operación

3.3.1.2. Tabla fichero

En esta tabla se almacena toda la información pertinente a los ficheros sobre los cuales se ha realizado alguna operación. Esta tabla debe actualizarse cada vez que se realiza algún cambio sobre un fichero, como moverlo de ruta o escribir en él aumentando su tamaño.

Campo	Tipo	Descripción
ID	Número entero	Identificador unívoco para la entrada. Clave primaria.
Ruta	Texto	Ruta absoluta al fichero, con el formato */raíz/directorio/fichero*
Tipo_fichero	Número entero	Clave ajena para el campo *ID* de la tabla *tipo_fichero*.
Tamaño	Número entero	El tamaño en bytes del fichero.
Eliminado	Valor lógico	Indica si el fichero ha sido eliminado del sistema de ficheros o no. Valores posibles: *true* (si ha sido eliminado), *false* (si no ha sido eliminado).

Tabla 45. Fichero

3.3.1.3. Tabla usuario

En esta tabla se almacena la información sobre los usuarios que realizan las operaciones. Dado que tanto el nombre como el ID de usuario en UNIX se pueden «reciclar» en un futuro si el usuario se elimina, es necesario guardar un registro de las combinaciones de nombre e ID que ha habido para no atribuir operaciones erróneamente a un usuario que no las ha realizado.

Campo	Tipo	Descripción
ID	Número entero	Identificador unívoco para la entrada. Clave primaria.
Nombre	Texto	El nombre de usuario.
UID	Número entero	El ID de usuario para el sistema operativo. En principio el rango es de 0 – 65534 [21].

Tabla 46. Usuario

3.3.1.4. Tabla tipo_operación

En esta tabla se guardan los distintos tipos de operaciones que se pueden registrar, su función es como la de un tipo de datos enumerado que se puede ir ampliando conforme el sistema admita más tipos de operaciones.

Campo	Tipo	Descripción
ID	Número entero	Identificador unívoco para la entrada. Clave primaria.
Nombre	Texto	El nombre de la llamada al sistema asociada a la operación. Ejemplos: read, write, chmod, getattr…
Datos	Valor lógico	Indica si la operación tiene como resultado una transferencia de datos (los metadatos no se tienen en cuenta). Valores posibles: *true* (si se transfieren datos), *false* (si no se transfieren datos)

Tabla 47. Tipo_operación

3.3.1.5. Tabla tipo_fichero

Esta tabla cumple una función similar a la de tipos de operaciones pero en su lugar se almacenan los tipos de ficheros conocidos, por si se quisiera implementar la funcionalidad de registrar operaciones sobre un determinado conjunto de tipos de ficheros, o incluir esta información en los informes generados. El tipo de fichero se define utilizando el estándar MIME [22].

Campo	Tipo	Descripción
ID	Número entero	Identificador unívoco para la entrada. Clave primaria.
Nombre	Texto	La cadena de caracteres que identifica el tipo de fichero.

Tabla 48. Tipo_fichero

El campo ID de las tablas *fichero*, *usuario*, *tipo_operación* y *tipo_fichero* se declarará como AUTOINCREMENT para que SQLite se encargue de la generación de identificadores unívocos.

3.3.2. Componentes

3.3.2.1. Sistema de ficheros

En este componente se implementan todas las operaciones sobre el sistema de ficheros, así como la configuración del registro y el volcado de datos al fichero de registro.

3.3.2.1.1. Operaciones

Las operaciones que deben poder registrarse, según los requisitos del sistema, son las siguientes:

Función de FUSE	Descripción	Transfiere datos
open	Abre un fichero para escritura o lectura.	No
read	Lee de un fichero.	Sí
write	Escribe a un fichero.	Sí
readdir	Lee el contenido de un directorio.	No
mkdir	Crea un directorio.	No
rmdir	Elimina un directorio.	No
chown	Cambia el propietario de un fichero.	No
chmod	Cambia los permisos de un fichero.	No
readlink	Lee un enlace.	No
link	Crea un enlace físico.	No
symlink	Crea un enlace simbólico.	No
unlink	Elimina un enlace.	No

Tabla 49. Funciones que deben interceptarse

Cada operación será interceptada por el sistema de ficheros virtual, que registrará la operación y el resultado de la misma en caso de que la configuración así lo indique y llamará a la función de FUSE correspondiente para que lleve a cabo la operación. La correspondencia entre funciones de FUSE y operaciones del sistema de ficheros virtual deberá almacenarse en un objeto de tipo *fuse_operations* que

se le pasará a la función *fuse_main* cuando se monte el sistema, según la forma habitual de implementar sistemas de ficheros virtuales en FUSE.

Cuando se realice una operación, se podrá obtener el contexto en el que se ha ejecutado mediante la función *fuse_get_context*, de la cual podemos extraer el UID y el PID del proceso que la realiza.

3.3.2.1.2. Formato del fichero de registro

Las operaciones quedarán registradas en un fichero con el formato que se describe a continuación. Cada operación deberá corresponder a una línea del fichero. Las operaciones deberán registrarse mediante el uso de la función *rLog*.

<hora> <operación> <datos transferidos> <fichero> <resultado> <uid> <proceso>

- La hora tiene que tener formato HH:MM:SS de 24 horas.
- La operación tiene que ser el nombre de la función de FUSE utilizada.
- La cantidad de datos transferidos tiene que estar en bytes.
- El nombre del fichero tiene que ser una ruta absoluta al fichero.
- El resultado tiene que ser un número entero que contenga el valor que devuelve la operación.
- El UID tiene que ser el identificador del usuario que realiza la operación dentro del sistema operativo.
- El nombre de proceso tiene que ser una ruta absoluta al fichero ejecutable del proceso.

3.3.2.1.3. Formato del fichero de configuración

El fichero de configuración tendrá un formato XML con la siguiente estructura:

Nodo	Padre	Atributos	Multiplicidad
config	(nodo raíz)	-enabled: true|false -file: string	1

includes	config	N/A	0 | 1
include	includes	-name: string -uid: int -action: string -return: int	0..n
excludes	config	N/A	0 | 1
exclude	excludes	-name: string -uid: int -action: string -return: int	0..n

Tabla 50. Estructura del fichero de configuración

El nodo *config* es el nodo raíz del fichero, y tiene dos atributos:

- *enabled*, que nos indica si el registro deber activarse (en caso de tener el valor «true») o desactivarse (en caso de tener el valor «false»). Este atributo es obligatorio.
- *file*, que indica el nombre del fichero al cual se deben registrar las operaciones. Este atributo es opcional. Si no se establece un valor, se aplicará el valor por defecto que es «./log.txt».

Los nodos de tipo *include* nos permiten definir reglas para determinar si una operación en concreto se registra, mientras que los nodos de tipo *exclude* nos permiten indicar si una operación debe ser excluida del registro. Por defecto no se registra ninguna operación, por lo que habrá que especificar explícitamente las operaciones que quieran registrarse. Ambos nodos tienen 4 atributos que se muestran a continuación. Los atributos son opcionales y el valor por defecto es «*», que recoge todos los valores posibles.

- *name*: el nombre que debe tener el fichero para que se dispare la regla, incluyendo la extensión. El valor de este atributo es una expresión regular con formato de PCRE [23].
- *uid*: el ID de usuario que debe tener la operación para que se dispare la regla. El valor de este atributo es un número entero.

- *action*: la función de FUSE que debe utilizar la operación para que se dispare la regla. El valor de este atributo es una de las funciones especificadas en el apartado anterior.
- *return*: el código de salida que debe devolver la operación para que se dispare la regla. El valor de este atributo es un número entero que indica el valor que devuelve la función de FUSE (0 para resultado exitoso, otro número para distintos errores).

3.3.2.2. Generador de informes

Este componente se encarga de consultar al controlador de base de datos y generar un documento que informe sobre los datos recuperados. Dicho documento contendrá gráficos contrastando la información de uso. El usuario deberá especificar qué información se contrasta según una serie de opciones disponibles.

En el eje de ordenadas se podrá elegir una de las siguientes medidas:

- El total de veces que se realiza una operación concreta.
- La cantidad de bytes leídos.
- La cantidad de bytes escritos.

En el eje de abscisas se podrá elegir una de las siguientes clasificaciones:

- Operación que se realiza.
- Usuario que realiza la operación.
- Proceso que realiza la operación.
- Fichero sobre el que se realiza la operación.

Además, se podrán aplicar filtros para sólo utilizar información que cumpla determinadas condiciones. Se podrá establecer un valor concreto para cada una de las clasificaciones anteriores, además de los siguientes criterios:

- El intervalo de horas en el que se realiza la operación.

- El intervalo de tamaño de ficheros que se quiere contemplar.
- El resultado de la operación.

Según estos tres parámetros se podrá definir el gráfico que queremos generar. Por ejemplo, si elegimos el total de operaciones en el eje de ordenadas, el usuario en el eje de abscisas y filtramos por operaciones de lectura y establecemos el intervalo de horas de 12:00 a 18:00, tendremos un gráfico que nos muestre el total de lecturas realizadas por usuario en dicho intervalo horario:

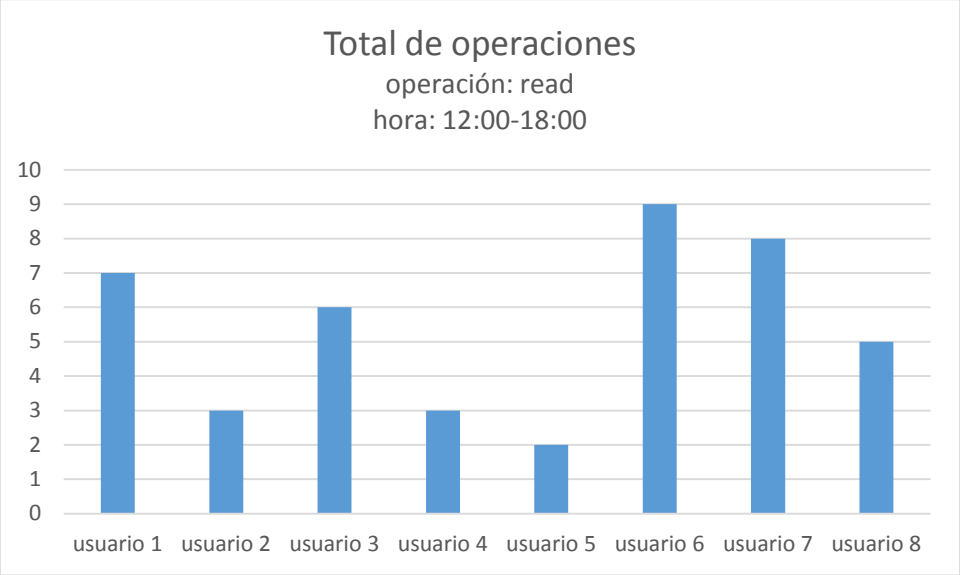

Ilustración 10. Ejemplo de gráfico generado

Para representar internamente las reglas de generación de gráficos se utilizará la sintaxis:

{

atributo 1:valor

atributo 2:inicio,fin

…

atributo n:valor

}

Para cada gráfico que se desee generar. A continuación se muestran todos los atributos que debe tener una definición de gráfico. Todos los atributos son obligatorios, aunque los filtros pueden ser nulos, teniendo como valor NULL si no se quiere aplicar ninguno.

Atributo	Descripción	Valores posibles	Puede ser nulo
y	Valores del eje de ordenadas.	-op: total de operaciones -le: total de datos leídos -es: total de datos escritos	No
x	Valores del eje de abscisas.	-op: operaciones -us: usuarios -pr: procesos -fi: ficheros	No
op	Filtrado por operación.	Todas las funciones del apartado 3.3.2.1.1.	Sí
us	Filtrado por usuario.	Cadena de texto con el nombre de usuario.	Sí
pr	Filtrado por proceso.	Cadena de texto con el nombre de proceso.	Sí
fi	Filtrado por fichero.	Cadena de texto con el nombre de fichero.	Sí
ho	Filtrado por hora.	Intervalo de números.	Sí
ta	Filtrado por tamaño.	Intervalo de números.	Sí
re	Filtrado por resultado.	Número entero con el resultado.	Sí

Tabla 51. Atributos de un gráfico

3.3.2.3. Controlador de base de datos

Este componente se encarga de gestionar todas las conexiones a la base de datos. En concreto, la funcionalidad que deberá implementar consiste en:

- La creación de una base de datos con el esquema especificado en el apartado 3.1.1.
- Volcar datos del fichero de registro a la base de datos.

- Recibir el conjunto de reglas de filtrado y devolver la información que corresponda.

Estas tres funciones deben de estar accesibles a los demás componentes del sistema en tres métodos separados, con el siguiente esquema:

Crear base de datos	
Parámetros	1. Ruta al fichero de base de datos.
Valor de retorno	Código de resultado: 0 si la operación es exitosa, código de error si no.

Tabla 52. Esquema del método para crear base de datos

Volcar datos de registro	
Parámetros	1. Ruta al fichero de registro. Valor por defecto si no se establece ninguno: ./log 2. Ruta al fichero de base de datos donde se quiere insertar la información.
Valor de retorno	Número de registros volcados, o -1 si se ha producido un error.

Tabla 53. Esquema del método para volcar datos de registro

Recuperar información	
Parámetros	1. La medida para el eje de ordenadas. 2. La clasificación para el eje de abscisas. 3. Una matriz con los filtros que se quieran aplicar. Será de tamaño fijo, y si no se quiere aplicar el filtro el elemento correspondiente deberá tener valor NULL. Los filtros con intervalos usarán una coma como separador.
Valor de retorno	Los valores obtenidos, separados por comas.

Tabla 54. Esquema del método para recuperar información

Para el volcado de datos, se necesita cierta información adicional a la que hay en el fichero de registro. Ésta se obtendrá de la siguiente forma:

- El nombre de usuario se obtendrá a partir del UID con la función *getpwuid* de Perl.
- El tamaño de fichero se obtendrá utilizando el operador *–s* de Perl.
- El tipo de fichero se obtendrá mediante la herramienta del sistema *file*.

En cuanto a la recuperación de datos, se utilizarán sentencias SELECT de SQL para recuperar el campo que queramos presentar, y se transformarán los filtros establecidos por el usuario en condiciones en la cláusula WHERE. Todas las operaciones de inserción y selección de datos que realice el controlador deberán parametrizarse a fin de aumentar su seguridad y eficiencia.

3.3.2.4. Matriz de trazabilidad

En la siguiente tabla se muestra la correspondencia entre requisitos del sistema y el componente que lo implementa. Para los requisitos no funcionales se indican los componentes a los que aplica, por lo que puede haber más de uno; los requisitos funcionales sólo pueden ser implementados por un componente, o de lo contrario habría redundancias en la funcionalidad de la aplicación.

	Sistema de ficheros	Generador de informes	Controlador de base de datos
RF-1	X		
RF-2	X		
RF-3	X		
RF-4	X		
RF-5	X		
RF-6	X		
RF-7	X		
RF-8	X		
RF-9	X		
RF-10	X		
RF-11	X		
RF-12	X		
RF-13	X		
RF-14	X		
RF-15	X		
RF-16	X		

RF-17	X		
RF-18	X		
RF-19	X		
RF-20	X		
RF-21	X		
RF-22		X	
RF-23	X		
RF-24	X		
RF-25		X	
RF-26		X	
RF-27			X
RF-28			X
RF-29			X
RNF-1	X	X	X
RNF-2	X		
RNF-3			X
RNF-4		X	
RNF-5	X		
RNF-6			X
RNF-7	X		

Tabla 55. Matriz de trazabilidad de componentes

3.3.3. Interfaz gráfica

En este apartado se describen unos prototipos para la interfaz gráfica del sistema, indicando la funcionalidad que se espera que tenga dicha interfaz además de una breve descripción del flujo de operación entre pantallas.

3.3.3.1. Montar y configurar sistema

En la pantalla principal deberá permitirse montar y desmontar el sistema y modificar todas las opciones de configuración, que en este caso se reducen a: activar y desactivar el registro, establecer reglas de registro y determinar un punto de montaje.

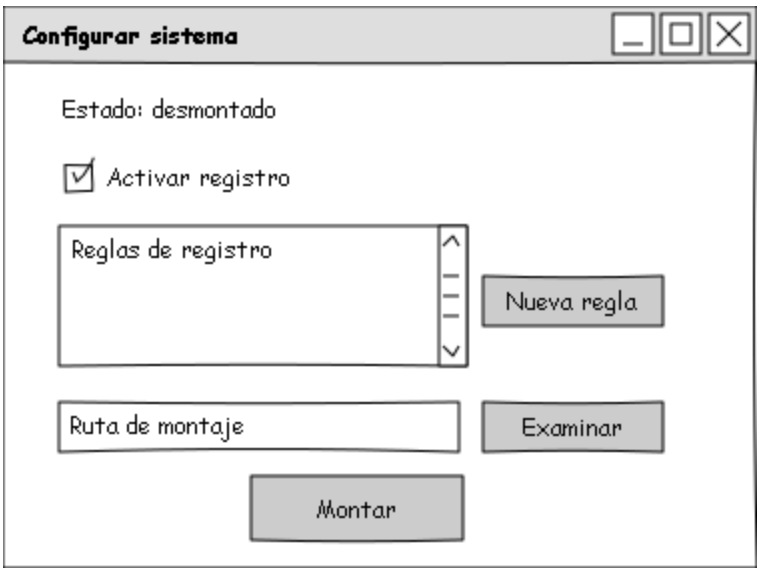

Ilustración 11. Configurar sistema, sin montar

Al montar el sistema se deberá informar al usuario del resultado de la operación mediante un cuadro de diálogo. Si la operación ha tenido éxito deberá mostrarse el siguiente mensaje:

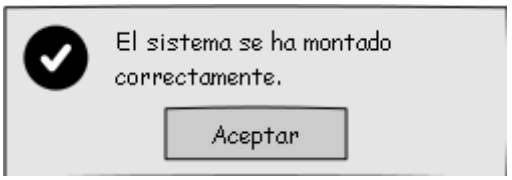

Ilustración 12. Mensaje de éxito

Si por el contrario la operación falla, deberá advertir del error e incluir una explicación de éste. Dado que se utiliza la herramienta *fusermount* para el montaje, el mensaje de error debe ser el que devuelva dicha herramienta.

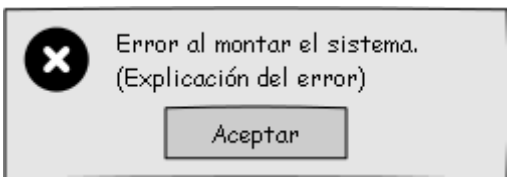

Ilustración 13. Mensaje de error

Una vez montado el sistema con éxito, cambiarán las opciones de la pantalla principal para permitir al usuario desmontar el sistema o cambiar la configuración y aplicar los cambios.

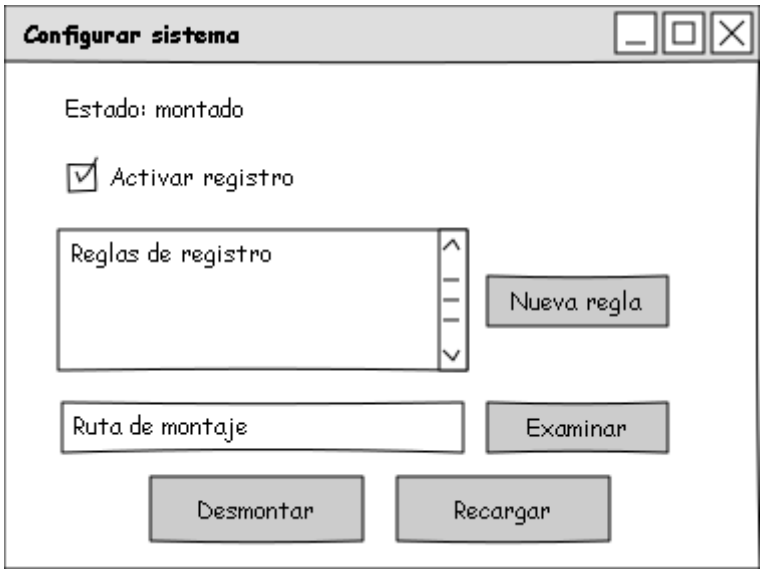

Ilustración 14. Configurar sistema, montado

Al crear una nueva regla, se abrirá una ventana en la que se podrán introducir los parámetros que definen la regla, según la configuración especificada en el apartado 3.3.2.1.:

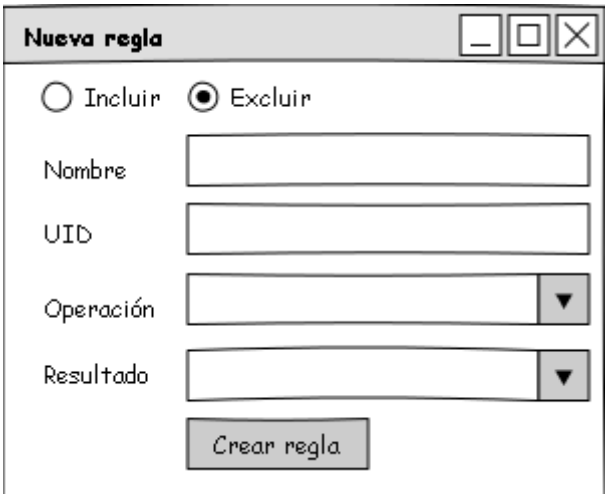

Ilustración 15. Nueva regla de registro

Al pulsar sobre el botón «Examinar» para seleccionar un punto de montaje, se deberá abrir una nueva ventana donde se permita navegar por el árbol de directorios del sistema para elegir uno de ellos como punto de montaje:

Ilustración 16. Seleccionar punto de montaje

3.3.3.2. Generar informe

Esta sección de la interfaz consta de dos partes, la pantalla del generador de informes, donde se indica qué información se quiere incluir, y el documento web generado. En la pantalla principal se mostrará una lista de posibles gráficos que se pueden generar y se permitirá establecer el fichero de salida:

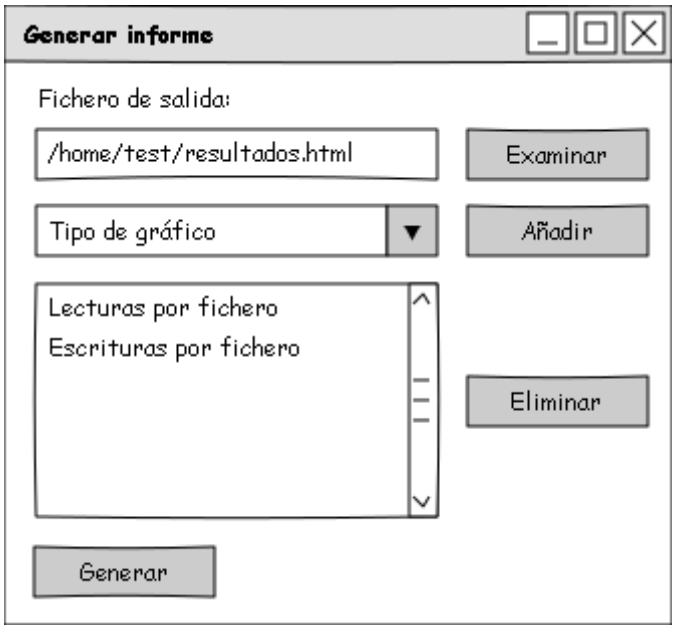

Ilustración 17. Generar informe

Al pulsar sobre el botón «Examinar» para seleccionar el fichero de salida, deberá aparecer una ventana emergente que nos permita elegir la ruta y el nombre del fichero:

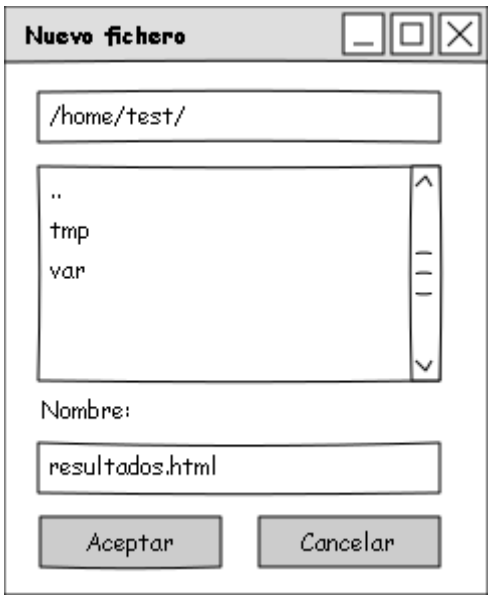

Ilustración 18. Nuevo fichero de resultados

El informe generado será un documento web que se deberá visualizar en un navegador, fuera de la interfaz gráfica de esta aplicación. A continuación se muestra un ejemplo de cómo debería quedar el documento generado para los dos tipos de gráficos seleccionados:

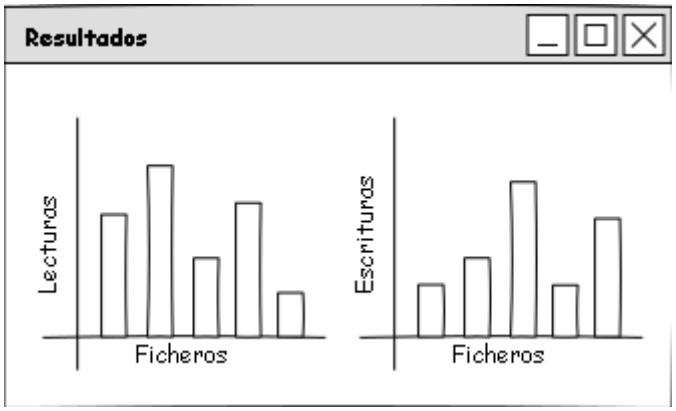

Ilustración 19. Ejemplo de documento generado

3.3.3.3. Gestionar datos

En la pantalla principal se deberá mostrar la base de datos actualmente seleccionada, o en caso de no haber ninguna, un mensaje que lo indique:

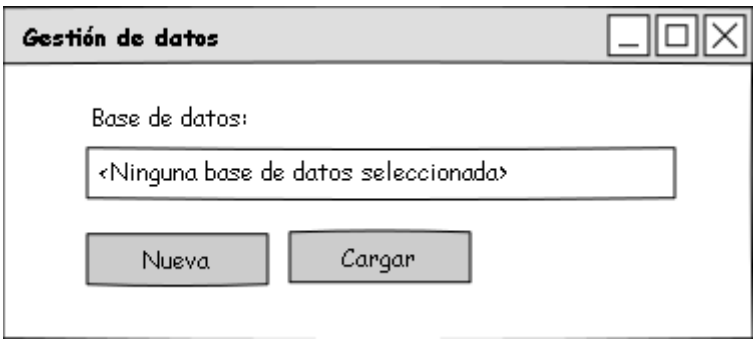

Ilustración 20. Gestión de datos, sin BD seleccionada

Para crear una nueva base de datos, se abrirá una ventana en la que se podrá introducir la ruta y el nombre de la base de datos nueva:

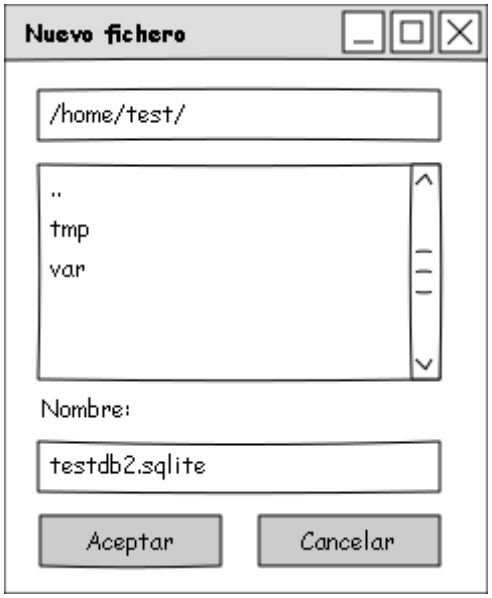

Ilustración 21. Nueva base de datos

Si por el contrario se desea cargar una base de datos ya existente, en la ventana emergente se permitirá seleccionar un fichero dentro del árbol de directorios:

Ilustración 22. Seleccionar base de datos

Una vez seleccionada una base de datos, se dará la opción en la pantalla principal de guardar los datos de la sesión actual en la base de datos seleccionada:

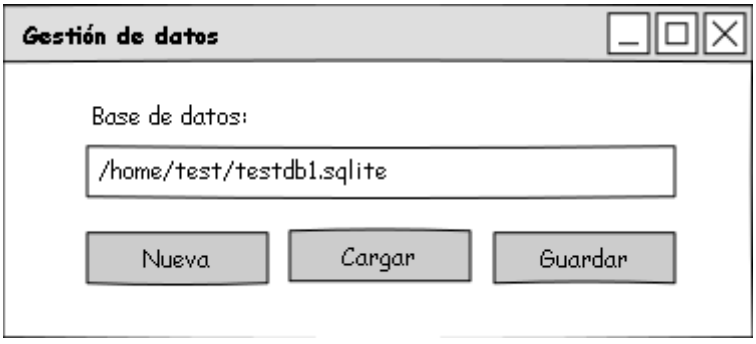

Ilustración 23. Gestión de datos, con BD seleccionada

Antes de realizar la operación de guardado, se pedirá la confirmación del usuario para evitar guardados indeseados, ya que la operación no es fácilmente reversible una vez finalizada:

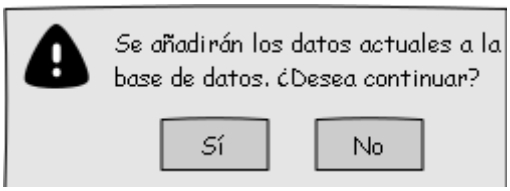

Ilustración 24. Mensaje de confirmación de guardado

4. Evaluación

Uno de los principales inconvenientes del uso de sistemas de ficheros virtuales es que, al añadir una capa más de procesado sobre el sistema de ficheros nativo, tienen un impacto negativo en el rendimiento. Por ello es de interés analizar hasta qué punto influye esto a nivel práctico para poder decidir si esta implementación sería viable en un entorno de producción.

En esta sección se prueba el sistema desarrollado y se mide el rendimiento para tres casos principales:

- Sin montar el sistema virtual. Este es el caso óptimo en cuanto a rendimiento.
- Con el sistema virtual montado pero el registro desactivado. Este caso se tiene en cuenta para decidir hasta qué punto influye el mero hecho de que sea un sistema virtual implementado con FUSE, sin tener en cuenta detalles concretos de este sistema que también pueden disminuir el rendimiento.
- Con el sistema virtual montado y el registro activado. De esta forma se evalúa el impacto real. Se prevé una diferencia considerable respecto de los casos anteriores, puesto que al activar el registro se añade una operación de escritura a cada operación que se realiza sobre el sistema.

Se ha elegido un soporte físico especialmente lento para la realización de las pruebas con el fin de disminuir la variabilidad en los resultados. Se trata de un disco duro SATA-I a 5400rpm formateado con NTFS. El directorio donde se realizan las pruebas es «/mnt/testdir». El punto de montaje «/tmp» corresponde a un sistema de ficheros en RAM que se utiliza para volcar datos dando una velocidad realista de la lectura del fichero de origen sin hacer de cuello de botella.

4.1. Escritura y lectura de datos

Tanto para la lectura como para la escritura se utiliza un tamaño de bloque de 1 MiB y se realizan 1024 iteraciones. Por tanto, la transferencia total de datos que se realiza en cada operación sobre el sistema de ficheros es de 1 GiB. La herramienta empleada para ello es *dd* [24], una herramienta que permite la transferencia de bloques de datos directamente entre ficheros.

Para probar la velocidad de escritura se utiliza el siguiente comando:

```
dd if=/dev/zero of=/mnt/testdir/file bs=1M count=1K
```

Para probar la velocidad de lectura se utiliza el siguiente comando:

```
dd if=/mnt/testdir/file of=/tmp/testfile bs=1M count=1K
```

Los resultados son los siguientes (en segundos):

	Sin montar	**Con registro desactivado**	**Con registro activado**
Escritura	10,4204	41,5744	145,11
Lectura	0,330552	0,95367	2,06152

Tabla 56. Resultados de dd

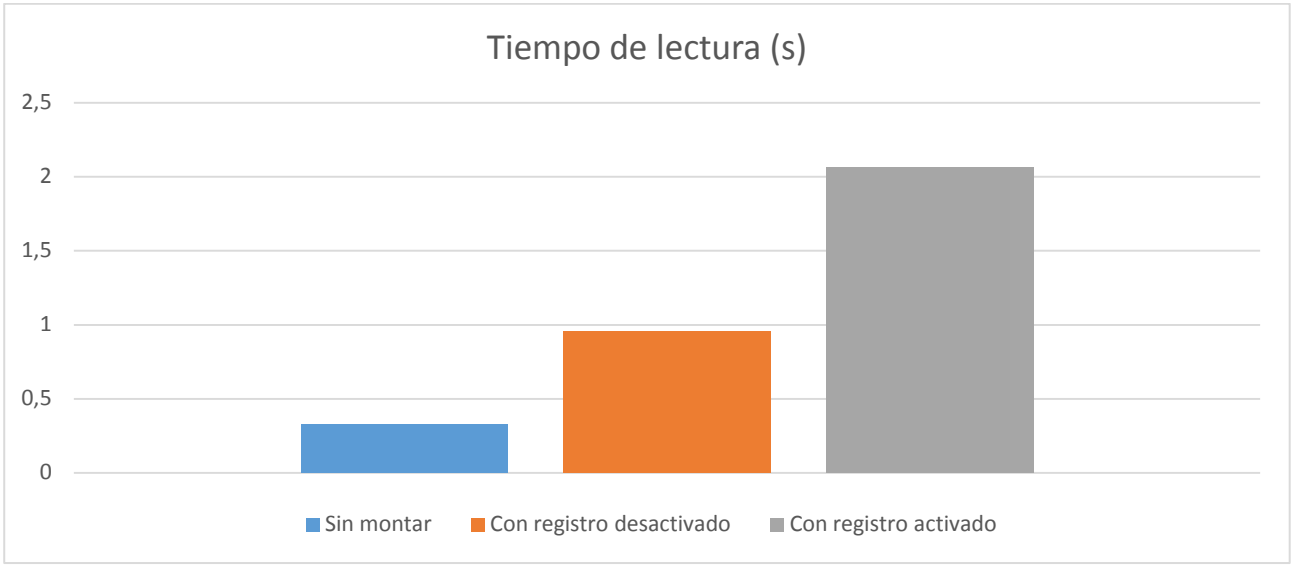

Ilustración 25. Tiempos de lectura

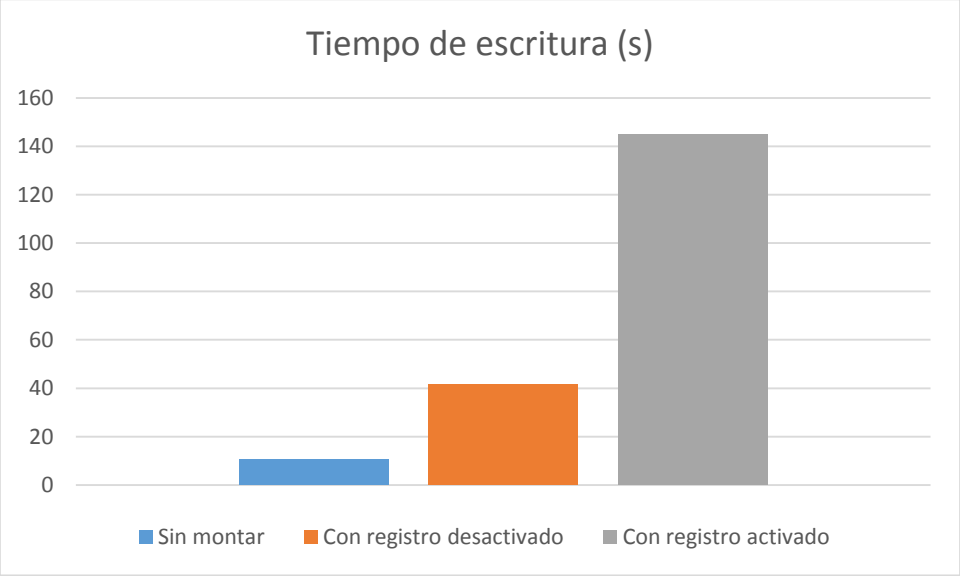

Ilustración 26. Tiempos de escritura

4.2. Rendimiento en metadatos

Además del rendimiento para transferencia de datos, es interesante medir el rendimiento en la modificación de metadatos. Cabe esperar que también haya una pérdida de rendimiento a este nivel, al tener que pasar todas estas operaciones por FUSE.

La herramienta que se utiliza para ello es *fdtree* [25]. Esta herramienta crea un árbol de directorios de forma recursiva con el número de ficheros por directorio que le especifiquemos. Para estas pruebas, se crea un árbol de profundidad 4 con 7 directorios por nivel y 5 ficheros por directorio.

Para ello se utiliza el siguiente comando:

```
fdtree.bash –f 5 –d 7 –o /mnt/testdir
```

Los resultados son los siguientes (en ficheros por segundo):

	Sin montar	**Con registro desactivado**	**Con registro activado**
Creación de directorios	466	400	350

Creación de ficheros	411	254	155
Eliminación de ficheros	2334	1273	1077
Eliminación de directorios	2801	2801	1400

Tabla 57. Resultados de fdtree

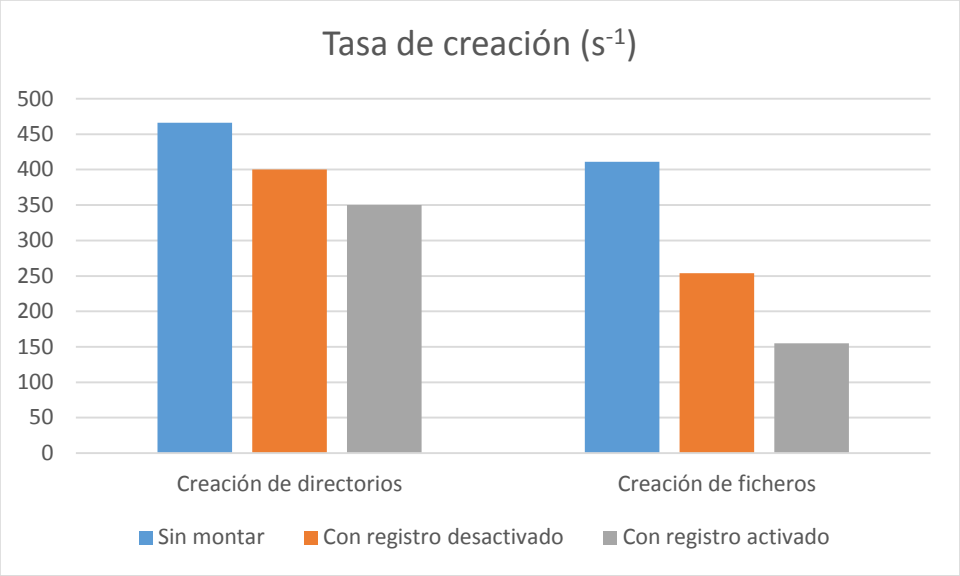

Ilustración 27. Tasa de creación

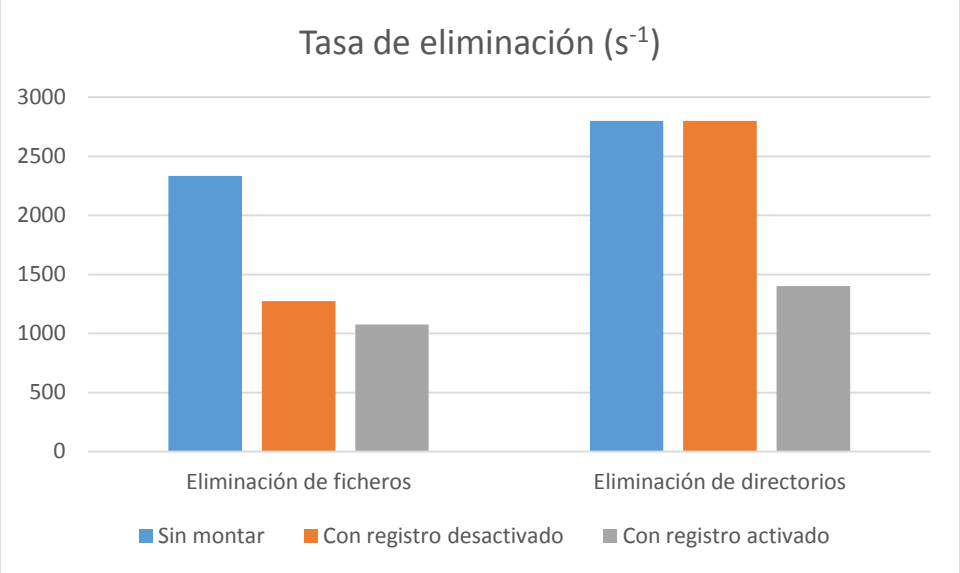

Ilustración 28. Tasa de eliminación

5. Planificación y presupuesto

En esta sección se detalla la planificación del tiempo para la realización del proyecto, así como todos los costes incurridos en ella. El proyecto comienza el día 18/11/13 y termina 4 meses más tarde, el 18/02/14. El total de días dedicados al proyecto es de 66.

A lo largo del ciclo de vida de este proyecto se han producido pequeñas modificaciones en la asignación de tiempo, que no han afectado a las fechas de inicio y de fin. La planificación que se muestra aquí es la definitiva.

5.1. Planificación

A continuación se muestra un resumen del tiempo asignado a cada fase del proyecto:

ID	Nombre de tarea	Duración	Comienzo
1	**Estudio previo**	**4 días**	**lun 18/11/13**
2	Problemática	1 día	lun 18/11/13
3	Viabilidad	1 día	mar 19/11/13
4	Estado de la cuestión	1 día	mié 20/11/13
5	Planificación	1 día	jue 21/11/13
6	Documentación del estudio previo	4 días	lun 18/11/13
7	**Análisis**	**14 días**	**vie 22/11/13**
8	Casos de uso	6 días	vie 22/11/13
9	Requisitos del sistema	8 días	lun 02/12/13
10	Documentación del análisis	14 días	vie 22/11/13
11	**Diseño**	**16 días**	**jue 12/12/13**
12	Arquitectura del sistema y componentes	8 días	jue 12/12/13
13	Modelo de datos	3 días	mar 24/12/13
14	Prototipo de interfaz gráfica	5 días	vie 27/12/13
15	Documentación del diseño	16 días	jue 12/12/13
16	**Implementación**	**24 días**	**vie 03/01/14**
17	Sistema de ficheros	8 días	vie 03/01/14
18	Conexión con base de datos	4 días	mié 15/01/14

19	Generador de informes	9 días	mar 21/01/14
20	Interfaz gráfica	3 días	lun 03/02/14
21	**Implantación**	**2 días**	**jue 06/02/14**
22	Instalación y ejecución en entorno de prueba	2 días	jue 06/02/14
23	Manual de usuario	2 días	jue 06/02/14
24	**Evaluación**	**3 días**	**lun 10/02/14**
25	Pruebas de rendimiento	2 días	lun 10/02/14
26	Documentación de la evaluación	1 día	mié 12/02/14
27	**Redacción de conclusiones**	1 día	jue 13/02/14
28	**Redacción de trabajos futuros**	1 día	lun 17/02/14
29	**Revisión final**	1 día	mar 18/02/14

Tabla 58. Planificación del proyecto

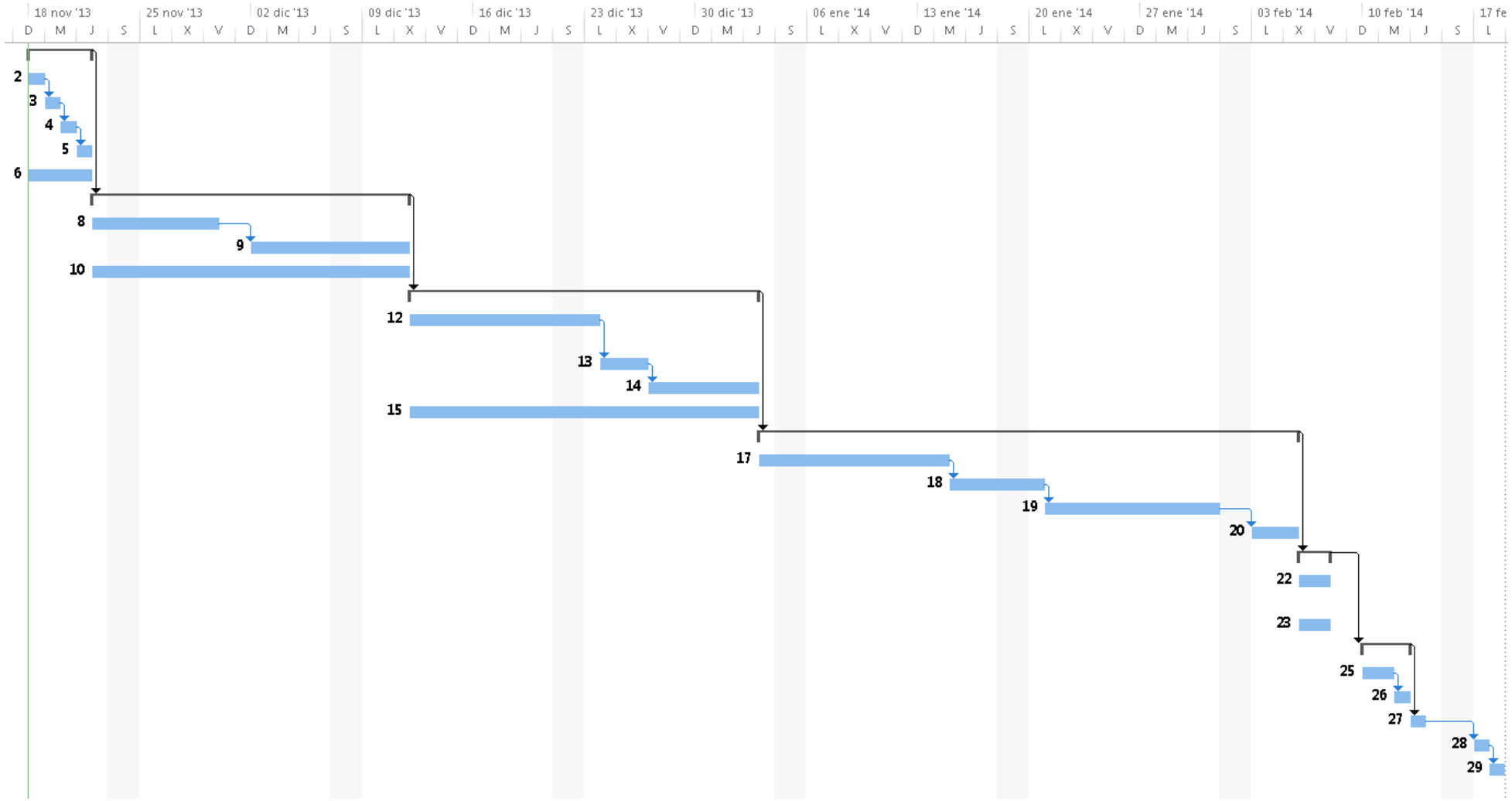

Tabla 59. Diagrama de Gantt del proyecto

5.2. Presupuesto

En esta sección se hace un desglose de todos los costes en los que se ha incurrido a lo largo de la elaboración del proyecto. El período de depreciación para material y aplicaciones informáticas se considera de 36 meses, y el coste atribuible se calcula mediante la siguiente fórmula:

*Coste atribuible = (Coste total / Período de depreciación) * Tiempo de uso*

En todos los costes se incluyen los impuestos que sean aplicables, como el IVA.

5.2.1. Costes

5.2.1.1. Personal

La jornada laboral es de 8 horas al día. El desglose de horas es, por tanto:

- El jefe de proyecto se encarga del estudio previo, el análisis de requisitos y las conclusiones y revisión final. Por tanto, debe dedicar un total de 21 días, lo que equivale a 168 horas de trabajo.
- El analista se encarga de realizar el diseño técnico a partir de los requisitos, por lo que tiene asignado un total de 16 días de trabajo, o lo que es lo mismo, 128 horas.
- El programador se encarga de la implementación, la implantación y las pruebas. Por tanto, tiene 29 días asignados, lo que equivale a 232 horas de trabajo.

Cargo	**Salario (€/hora)**	**Tiempo de trabajo (horas)**	**Coste (€)**
Jefe de proyecto	18,75	168	3150
Analista	12,5	128	1600
Programador	9,375	232	2175
		Total bruto	6925
		Seguridad Social (23,6%)	1634,3
		Total neto	8559,3

Tabla 60. Costes de personal

5.2.1.2. Material informático

En esta sección se detallan los costes asociados al material informático empleado en el proyecto. Para el período de depreciación se ha seleccionado el máximo entre la vida útil del aparato y la duración de la garantía.

Tipo	Coste total (€)	Período de depreciación (días)	Período de uso (días)	Coste atribuible (€)
Portátil	1000	1080	66	61,1
Impresora	90	1080	66	5,5
Memoria externa	15	720	66	1,4
			Total	68

Tabla 61. Costes de material informático

5.2.1.3. Aplicaciones

En esta sección se detallan los costes asociados a las aplicaciones informáticas empleadas en el proyecto. Dichos costes pertenecen al precio de una licencia de uso para una persona, en caso de tener que pagarla. El período de amortización se considera el tiempo que tarda en aparecer una versión nueva que reemplace a la aplicación, que suelen ser unos 3 años para productos de Microsoft.

Nombre	Coste total (€)	Período de amortización (días)	Período de uso (días)	Coste atribuible (€)
Debian GNU/Linux	0	1080	66	0
Eclipse	0	1080	66	0
Oracle Virtualbox	0	1080	66	0
MS Windows 7	375	1080	66	22,9

Professional				
MS Office 2013	539	1080	66	32,9
MS Project 2013	1369	1080	66	83,7
MS Visio 2013	739	1080	66	45,2
			Total	184,7

Tabla 62. Costes de aplicaciones

5.2.1.4. Bienes fungibles

En esta sección se detallan los costes asociados al material que se ha consumido para la realización del proyecto.

Tipo	**Cantidad**	**Coste (€)**
Papel	500 folios	20
Bolígrafos	50 unidades	10
Tinta de impresora	1 cartucho	60
	Total	90

Tabla 63. Costes de bienes fungibles

5.2.1.5. Costes indirectos

En esta sección se desglosan los costes indirectos de la realización del proyecto, que no encajan en ninguna de las categorías anteriores.

Tipo	**Cantidad**	**Coste (€)**
Electricidad	4 meses	160
Línea telefónica	4 meses	120
Desplazamiento	10 viajes en coche	50

	Total	330

Tabla 64. Costes indirectos

5.2.2. Resumen

A continuación se proporciona un resumen de todos los costes asociados al proyecto:

Personal	8559,3
Material informático	68
Aplicaciones	184,7
Bienes fungibles	90
Costes indirectos	330
Total	9232

Tabla 65. Resumen de costes

6. Conclusiones y trabajos futuros

6.1. Conclusiones

Los objetivos originales del proyecto eran tres:

1. El análisis de la información que queremos obtener del uso de un sistema de ficheros.
2. El diseño de un sistema de ficheros que nos permita obtener dicha información.
3. La implementación de un prototipo para servir de ejemplo y/o de base para un posible sistema futuro más completo.

No cabe duda de que al ir del plano más abstracto a lo más concreto lo más difícil son siempre los primeros pasos. En este proyecto, uno de los mayores retos afrontados ha sido el de decidir exactamente qué información le interesa al usuario del sistema de ficheros, y cómo debe presentársela.

Para ello, se ha decidido intentar ofrecer un nivel alto de personalización, para que sea el usuario el que decida qué información quiere obtener, y de flexibilidad, para dejar lugar a posibles ampliaciones, como por ejemplo el registro de nuevos tipos de operaciones o nuevos filtros para los gráficos.

Además, puesto que es un tema en el que todavía quedan muchas posibilidades y líneas de investigación, se ha intentado en todo momento que el trabajo fuera lo suficientemente genérico y adaptable como para generar datos para otros sistemas, como por ejemplo de inteligencia artificial para el análisis de datos.

En general, se puede considerar que el proyecto ha cumplido los objetivos propuestos, aunque la problemática es mucho más extensa y compleja de lo que se pueda abordar en un proyecto de este calibre. Por ello, todavía hay lugar para infinidad de proyectos que aborden este mismo problema, algunos de los cuales podrían nutrirse del trabajo desarrollado aquí.

6.2. Trabajos futuros

La implementación que se ha realizado en este proyecto supone una primera aproximación a la solución del problema, pero por restricciones de tiempo se ha dejado fuera mucha funcionalidad de la que sería interesante disponer para desarrollar un sistema completo que sea utilizado por el público general. Además, quedarían por desarrollar los mecanismos necesarios para procesar la información generada, puesto que en este proyecto principalmente se ha tratado el hecho de registrarla y no qué se puede hacer con ella.

Algunas de las posibles líneas de trabajo futuras que se podrían desarrollar son:

- La generación de perfiles de uso que puedan ser procesados automáticamente. Por ejemplo, esto nos permitiría registrar el uso normal que se hace de un sistema y detectar anomalías, que podrían ser debidas a una intrusión o un virus informático.
- La implementación de un sistema de alarmas, que nos avise cuando tienen lugar distintos eventos en el sistema de ficheros. Por ejemplo, si se supera un umbral de datos escritos al disco en un período de tiempo determinado, o se cambian los permisos en un árbol de directorios que tenemos monitorizado.
- La creación de reglas de filtrado configurables por el usuario. Con ello se podría optimizar mucho más el rendimiento del sistema y el tamaño de los registros, permitiendo descartar operaciones que no nos interesen. Por ejemplo, podríamos excluir en función del proceso, hora del día, etc. y combinar todas las condiciones para crear un perfil de registro.
- La configuración más precisa de los datos generados en el informe, como por ejemplo permitir contrastar dos parámetros cualesquiera de las operaciones en lugar de tener un conjunto de gráficas predefinido.
- Sería interesante además realizar una implementación del sistema a nivel de núcleo para entornos en los que el registro deba estar activado constantemente. De esta forma se podría mitigar considerablemente el problema del rendimiento del sistema de ficheros, a cambio de sacrificar flexibilidad y portabilidad.

Anexo I. Acrónimos

Acrónimo	Significado
FUSE	Filesystem in userspace, un mecanismo para acceder a funciones del sistema de ficheros en modo usuario en sistemas tipo Unix.
GPL	General Public Licence, una licencia que permite la libre distribución y modificación de software, siempre que las modificaciones se distribuyan bajo la misma licencia.
API	Application programming interface, conjunto de funciones que ofrece una aplicación para permitir la comunicación con aplicaciones externas.
GNU	GNU is Not Unix (acrónimo recursivo), un proyecto cuyo objetivo es el de proporcionar un clon de Unix hecho exclusivamente con software libre.
POSIX	Portable Operating System Interface, un conjunto de estándares que intentan mantener la compatibilidad entre distintos sistemas operativos.
SQL	Standard Query Language, el lenguaje más comúnmente usado para instrucciones en bases de datos.
DBMS	Database Management System, un sistema que se encarga de la organización física de la base de datos y de procesar las instrucciones que se ejecuten sobre ella.
HTML	Hypertext Markup Language, el lenguaje usado para la presentación de documentos web, que indica al navegador los elementos que tiene que dibujar y cómo deben estar organizados.
IEEE	Institute of Electrical and Electronics Engineers, una asociación dedicada al desarrollo de estándares y protocolos tecnológicos.
UTF-8	Unicode Transformation Format – 8 bit, una codificación de caracteres universal de longitud variable, en la que se utilizan menos bits para representar caracteres más comunes.
UID	User identifier, un número asignado a cada usuario en un sistema tipo Unix.
SATA	Serial Advanced Technology Attachment, un tipo de interfaz de comunicación con dispositivos de almacenamiento, en general discos duros y unidades ópticas.
NTFS	New Technology File System, el sistema de ficheros estándar para Windows.
RAM	Random Access Memory, memoria temporal de un ordenador, a cuyas posiciones se puede acceder de forma independiente, sin pasar por otras antes.
MiB	Mebibyte, el equivalente a 1048576 bytes.
GiB	Gibibyte, el equivalente a 1024 MiB.

Anexo II. Manual de instalación

El desarrollo y las pruebas del prototipo se han realizado en Debian 7 (*Wheezy*) y la versión siguiente, que se encuentra en fase de pruebas (*Jessie*). Por lo tanto, se recomiendan como mínimo las versiones que se encuentran en los repositorios de Debian 7 de cada una de las aplicaciones. Se recomienda también disponer de 512MB de RAM y un procesador x86 de séptima generación o superior [26].

Para ejecutar el sistema es necesario disponer de:

- FUSE. Versión recomendada: 2.9.0 en adelante.
- Perl. Versión recomendada: 5.14 en adelante.
- GTK. Versión recomendada: 3.4.2 en adelante.
- Un navegador compatible con HTML5 y JavaScript. Se recomienda usar Mozilla Firefox a partir de la versión 17 (llamado «Iceweasel» en Debian [27]).
- Los módulos DBD::SQLite y Gtk3 de Perl, versiones recomendadas: 1.37 y 0.006 en adelante, respectivamente.

Se pueden instalar todas estas dependencias en Debian con el siguiente comando:

```
apt-get install fuse perl libgtk-3-0 iceweasel libdbd-sqlite3-perl
libgtk3-perl
```

En cuanto a la compilación del sistema, puesto que Perl es un lenguaje interpretado, sólo es necesario compilar el sistema de ficheros, escrito en C++. Para ello se proporciona un «Makefile» en el que se contienen todas las instrucciones que deben pasarse al compilador. Por tanto, necesitaremos:

- Un compilador de C++. Se recomienda usar g++ a partir de la versión 4.7.2.
- La herramienta *make*. Versión recomendada: 3.81 en adelante.
- Las cabeceras de C de las siguientes herramientas:
    - FUSE. Versión recomendada: 2.9.0 en adelante.
    - Rlog. Versión recomendada: 1.4 en adelante.
    - PCRE. Versión recomendada: 0.9.5 en adelante.
    - Libxml2. Versión recomendada: 2.8.0 en adelante.

Se pueden instalar todas estas dependencias en Debian con el siguiente comando:

```
apt-get install libfuse-dev librlog-dev libpcre++-dev libxml2-dev
```

Para compilar el sistema basta con situarse en el directorio raíz del código fuente y ejecutar el comando «make». Esto nos crea un fichero ejecutable que, si bien podemos invocar manualmente, no es necesario hacerlo puesto que la forma recomendada de hacerlo es a través de la interfaz gráfica.

Anexo III. Manual de usuario

Para las pruebas del sistema se han adaptado los componentes para que puedan ser invocados manualmente y de forma individual, para facilitar la detección de errores y poder visualizar más claramente el funcionamiento a nivel interno. Los pasos que se seguirían para utilizar la aplicación son los siguientes:

1. Compilamos el sistema de ficheros. Para ello nos situamos en el directorio src y ejecutamos el comando «make»:

   ```
   cd /mnt/src
   make
   ```

   Deberá aparecer un fichero ejecutable con el nombre «fs».

2. Montamos el sistema de ficheros. Debemos asegurarnos que nuestro usuario tiene permisos de escritura en el punto de montaje y que pertenece al grupo «fuse».

   ```
   ./fs conf.xml /mnt/testdir/
   ```

   El fichero de configuración (*conf.xml*) que se usa para este ejemplo es el siguiente:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <config log="true" file="/mnt/log">
        <includes>
             <include action="read" />
             <include action="write" />
        </includes>
   </config>
   ```

   Este fichero indica que deben registrarse únicamente las operaciones de lectura y escritura, para todos los ficheros, todos los usuarios y cualquier resultado. Los datos registrados se almacenarán en el fichero */mnt/log*. El sistema se montará en el directorio */mnt/testdir*.

3. Realizamos algunas operaciones sobre el sistema de ficheros para que queden registradas:

   ```
   cd /mnt/testdir
   echo 0 > file1
   cat file1 file1 > file2
   dd if=file2 of=file3 bs=1
   echo 0 >> file2
   cat file3
   ```

   El fichero de registro (*/mnt/log*) resultante tras realizar estas operaciones es el siguiente:

   ```
   14:12:08 write 2 /mnt/testdir/file1 0 1000 bash
   14:12:12 read 2 /mnt/testdir/file1 0 1000 cat
   ```

```
14:12:12 write 2 /mnt/testdir/file2 0 1000 cat
14:12:12 read 2 /mnt/testdir/file1 0 1000 cat
14:12:12 write 2 /mnt/testdir/file2 0 1000 cat
14:12:16 read 4 /mnt/testdir/file2 0 1000 dd
14:12:16 write 1 /mnt/testdir/file3 0 1000 dd
14:12:16 write 1 /mnt/testdir/file3 0 1000 dd
14:12:16 write 1 /mnt/testdir/file3 0 1000 dd
14:12:16 write 1 /mnt/testdir/file3 0 1000 dd
14:12:23 write 2 /mnt/testdir/file2 0 1000 bash
14:12:28 read 4 /mnt/testdir/file3 0 1000 cat
```

Podemos comprobar que se han registrado bien todas las operaciones.

4. Volcamos el fichero de registro a una nueva base de datos:

```
./controlador-bd.pl nueva /mnt/logdb.sqlite
./controlador-bd.pl guardar /mnt/log /mnt/logdb.sqlite
```

5. Llamamos al generador de informes para que genere los gráficos a partir de la información contenida en la base de datos. Para este caso, se pide generar un gráfico de lecturas por ficheros y otro de escrituras por ficheros:

```
./generador.pl /mnt/logdb.sqlite /mnt/rules /mnt/output.html
```

El fichero de reglas */mnt/rules* tiene el siguiente contenido:

```
{
y:op
x:fi
op:read
us:NULL
pr:NULL
fi:NULL
ho:NULL,NULL
ta:NULL,NULL
re:NULL
}
{
y:op
x:fi
op:write
us:NULL
pr:NULL
fi:NULL
ho:NULL,NULL
ta:NULL,NULL
re:NULL
}
```

Abrimos el fichero de salida (en esye caso */mnt/output.html*) en un navegador web. Para el ejemplo que se acaba de generar tendría el siguiente aspecto:

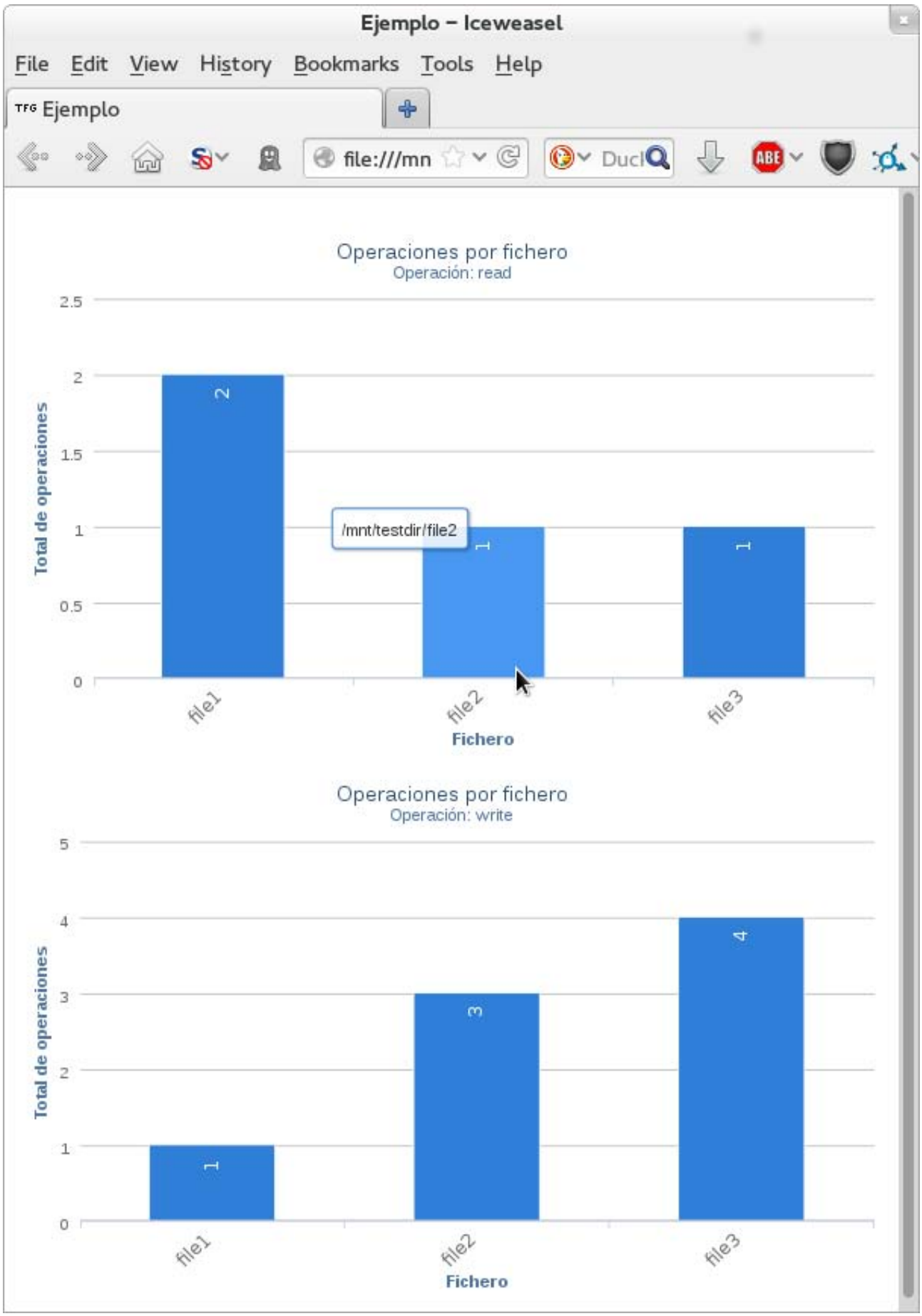

Ilustración 29. Ejemplo de documento generado

La forma de invocar cada componente es la siguiente:

```
./fs <fichero de configuración> <punto de montaje>
./controlador-bd nueva <fichero de base de datos>
./controlador-bd guardar <fichero de registro> <fichero de base de datos>
./generador.pl <fichero de base de datos> <fichero de reglas> <fichero de salida>
```

En la versión final de la aplicación se llamará a cada uno de los componentes desde la interfaz gráfica, con sus respectivos parámetros. Al tratarse de un prototipo la interfaz no es completamente funcional y por ello se ha optado por no utilizarla para el manual.

Referencias

1. Manual de inotify: http://man7.org/linux/man-pages/man7/inotify.7.html
2. Página de LoggedFS: http://loggedfs.sourceforge.net/
3. Licencia GPLv3: https://www.gnu.org/licenses/gpl.html
4. Página de FUSE: http://fuse.sourceforge.net/
5. Página de WikipediaFS: http://wikipediafs.sourceforge.net/
6. Contrato social de Debian: http://www.debian.org/social_contract
7. Política de documentación de Debian: http://www.debian.org/doc/debian-policy/ch-docs.html.
8. Introducción a Perl: http://www.perl.com/pub/2000/10/begperl1.html
9. Perl golf: http://perlgolf.sourceforge.net/
10. Página oficial de SQLite: https://sqlite.org/about.html
11. Página oficial de GTK: http://www.gtk.org/overview.php
12. Bindings de GTK para Perl: http://gtk2-perl.sourceforge.net/
13. Página oficial de d3js: http://d3js.org/
14. Licencia BSD de 3 cláusulas: http://opensource.org/licenses/BSD-3-Clause
15. Crossfilter: http://square.github.io/crossfilter/
16. Página oficial de Eclipse: http://eclipse.org/
17. IBM RAD: http://www-01.ibm.com/software/rational/eclipse/
18. Oracle Workshop: http://www.oracle.com/technetwork/developer-tools/workshop/overview/index.html
19. Ley Orgánica de Protección de Datos: https://www.boe.es/buscar/doc.php?id=BOE-A-1999-23750
20. Estándar IEEE 1003.1: http://pubs.opengroup.org/onlinepubs/009695399/basedefs/xbd_chap04.html#tag_04_04
21. UID en Debian: http://debianhelp.co.uk/usersid.htm
22. RFC 2046, tipos de datos MIME: https://tools.ietf.org/html/rfc2046
23. PCRE: http://www.pcre.org/
24. Manual de dd: http://man7.org/linux/man-pages/man1/dd.1.html
25. Página de descarga de fdtree: https://computing.llnl.gov/?set=code&page=sio_downloads
26. Requisitos mínimos de Debian: http://www.debian.org/releases/stable/i386/ch03s04.html.en
27. Iceweasel: https://packages.debian.org/wheezy/iceweasel