Alumno: Juan Aguilera Ramos
Tutor: Jesús Gabaldón

Proyecto Final de Máster

Repositorios Online para Sironta

Índice

1. Introducción....3
2. Objetivos....4
3. Estado del arte....5
4. Planificación....6
1.Fases de entrega a la empresa [planificación inicial]....6
2.Estado de las fases de entrega a la empresa....6
3.Diagrama de Gantt [planificación inicial]....6
5. Puesta en marcha del entorno de trabajo....8
6. Análisis....9
4.Base de datos....9
5.Código....9
6.Especificación de la URI....10
7.Adaptación de la URI al modelo actual....11
8.Protocolos soportados....11
9.Entorno de pruebas....11
7. Diagrama de clases....12
8. Diagrama de secuencia....13
9. Implementación....14
10. Interfaz de Usuario....15
11. Inclusión de un nuevo tipo de repositorio....18
12. Conclusiones....19
13. Bibliografía....20

Repositorios Online para Sironta

1. Introducción

Este Proyecto de final de Máster (PFM) se sitúa dentro del marco de la plataforma Sironta, desarrollada por la empresa Techideas. Sironta es un programa multiplataforma y software libre, liberada bajo licencia Affero GPL (AGPL). Sironta es una herramienta *peer to peer* (P2P) que permite la colaboración e interrelación entre distintas personas mediante la utilización de salas virtuales. En éstas se pueden compartir ficheros y sobretodo se pueden editar documentos de manera colaborativa.

Este PFM pretende dotar de una nueva funcionalidad al programa, permitiéndole tener el repositorio de los documentos en un lugar accesible permanentemente. Esto permitirá que los usuarios de una sala no tengan que esperar que se conecte el anfitrión de la misma, para poder trabajar sobre esos documentos. Para ello hay que desarrollar un *plugin* que permita seleccionar el tipo de repositorio, así como la autentificación necesaria. Se tendrá que diseñar un protocolo lo suficientemente genérico para permitir que se puedan añadir nuevos *plugins* para otros sistemas de almacenamiento, ya sean *online* o en intranet.

Sironta dispone de un editor de documentos OpenOffice empotrado incluyendo las herramientas *Writer*, *Calc*, *Impress* y *Draw*. Estos documentos se guardan en el repositorio de la sala y se sincroniza a los participantes de la misma cuando se conectan. Aparte de este tipo de documentos se pueden añadir y compartir cualquier fichero sin limitación de tamaño.

Una característica muy interesante es que posee un sistema de control de versiones de los documentos basado en *git*, el mismo que se usa entre los desarrolladores del Kernel de Linux, por lo que se puede retroceder en el historial si los cambios no son satisfactorios.

2. Objetivos

El objetivo principal de este PFM consiste en la adaptación del programa Sironta para que sea capaz de modificar la ubicación de sus repositorios, situándolos en un sitio accesible a todos los integrantes de la sala. Ello lleva implícito algunos sub-objetivos que son los siguientes:

- Análisis del estado actual de la plataforma
- Estudio de la BBDD de Sironta
- Estudio del programa Sironta
- Diseñar el sistema que permita la adición de nuevos repositorios
- Modificaciones en la BBDD de Sironta
- Diseño de la interfaz de usuario
- Implementación de un nuevo protocolo
- Implementación de la interfaz de usuario

3. Estado del arte

La plataforma Sironta, en el momento de iniciar este PFM, se encuentra en la versión 11 y por lo tanto es plenamente funcional. Es una herramienta multiplataforma, disponible para GNU/Linux, Microsoft Windows, y Mac OS. Ha sido desarrollada por la empresa TechIdeas y liberada a la comunidad del software libre.

Está creada en el lenguaje java, concretamente usando Eclipse RCP. Eclipse RCP es una plataforma para crear aplicaciones cliente. Incluye Equinox, una implementación del estándar OSGi. Esto permite la modularización del proyecto y hace más fácil la adición de nuevos módulos en forma de *plugins* que han de cumplir unas ciertas especificaciones establecidas en la API.

El funcionamiento hasta ahora de Sironta, por otra parte, requiere que el dueño de la sala esté conectado para que los integrantes de la misma puedan sincronizar los documentos. En este momento, un usuario tiene tres posibilidades de conexión, que dependen del sistema de conexión que usen y de las características *hardware* del mismo. Estas tres alternativas, que tienen la finalidad de favorecer la conectividad, evitando problemas de NAT, se gestionan de manera automática y transparente al usuario y tendrán que seguir funcionando de así con la implementación del nuevo protocolo.

4. Planificación

1. Fases de entrega a la empresa [planificación inicial]

Entre la empresa y yo hemos pactado tres entregas para ir haciendo un seguimiento del proyecto. Estas tres fases de entrega son las siguientes:

1. Entorno de desarrollo funcionando. Esta fase abarca la instalación, y configuración del sistema de control de versiones, así como del entorno de desarrollo. La entrega se realizará a finales de marzo.

2. Comprobación del nuevo protocolo. Esta fase incluye modificaciones en la base de datos de Sironta así como las modificaciones en el núcleo del programa. Está prevista la entrega para finales de abril.

3. Entrega final. En esta última entrega tiene que estar funcionando la interfaz de usuario que permite la selección de repositorio. Esta fase se presentará a finales de mayo.

2. Estado de las fases de entrega a la empresa

Debido a problemas en la puesta en marcha del entorno de trabajo, y a modificaciones de los requerimientos, las fases de entrega han sufrido cambios.

1. Entorno de desarrollo funcionando. Esta fase ya está entregada pero debido a los problemas mencionados se ha retrasado la entrega.

2. Comprobación del nuevo protocolo. En esta fase se han realizado las subfases de análisis de la base de datos y del código existente. Esto ha supuesto dos reuniones con la empresa que, gracias a su ayuda, me ha permitido encarar este proyecto con garantías. Después de analizar el proyecto he llegado a la conclusión que, al menos en primera instancia, no es imprescindible la creación de un nuevo *plugin*, aunque puede ser que en estados más adelantados del proyecto si que se requiera.

   Así pues, después del trabajo de análisis y diseño realizado queda pendiente la implementación, con la ventaja que la estructura del proyecto tal y como se diseñó favorece la adopción de los nuevos tipos repositorios.

3. Entrega final. Esta última entrega está pendiente, aunque es previsible que se tenga que retrasar alguna semana.

3. Diagrama de Gantt [planificación inicial]

A continuación podemos ver la planificación original de las diferentes tareas y subtareas:

Nombre	Fecha de inicio	Fecha de fin
T1. Puesta en marcha entorno	21/03/11	26/03/11
T1.1 Puesta en marcha del sistema de control de versiones	21/03/11	24/03/11
T1.2 Descarga del proyecto y puesta en marcha en el IDE	24/03/11	26/03/11
T2. Cambios en la BBDD de Sironta	28/03/11	9/04/11
T2.1 Estudio de la BBDD de Sironta	28/03/11	2/04/11
T2.2 Modificaciones para gestionar nuevos tipos de repositorio	4/04/11	9/04/11
T3. Realización del nuevo protocolo	11/04/11	4/05/11
T3.1 Análisis	11/04/11	14/04/11
T3.2 Diseño	14/04/11	16/04/11
T3.3 Implementación	18/04/11	4/05/11
T4. Realización del GUI	4/05/11	25/05/11
T4.1 Análisis	4/05/11	7/05/11
T4.2 Diseño	9/05/11	11/05/11
T4.3 Implementación	11/05/11	25/05/11
T5. Implementación de un segundo protocolo	25/05/11	1/06/11

A partir de esta planificación se puede apreciar las tareas realizadas y las que quedan por hacer. A día de la entrega a la empresa este es el estado de las tareas:

T1: Completada
T2:
- **T2.1**: Completada
- **T2.2**: Pendiente

T3:
- **T3.1**: Completada
- **T3.2**: Completada
- **T3.3**: Completada*

T4: Completada
T5: Pendiente*

Aunque la tarea 3.3 pone que está completada hay que puntualizar que sólo he implementado el protocolo *file*, para el acceso a un repositorio compartido en local. Mi intención y la del proyecto originalmente, era la implementación de otro protocolo más pero debido a problemas iniciales, que han retrasado el proyecto, no ha sido posible.

La tarea 5 fue concebida originalmente por si sobraba tiempo para añadir alguno de los protocolos más complicados, como por ejemplo dropbox.

5. Puesta en marcha del entorno de trabajo

La planificación actual no se ha podido llevar a cabo debido a problemas de logística. Aunque el proyecto Sironta está liberado bajo licencia AGPL no estaba creado el sistema de control de versiones de cara a la comunidad. Por lo tanto, se ha tenido que crear para mi y posteriores colaboradores. Tampoco estaba habilitado el acceso remoto para la base de datos de Sironta pero esto lo solucionaron con phpMyAdmin. Para el sistema de control de versiones, después de una de las reuniones nos decantamos por *git*, el usado entre otros proyectos por el kernel de linux y el mismo que usa internamente Sironta. La elección fue puramente académica ya que nos pareció interesante la manera de trabajar de este sistema.

Por mi parte, siguiendo las directrices de Javier Noguera, me instalé el IDE Eclipse en su versión para *RCP Developers* y me fui documentando de cómo funciona el sistema de trabajo por plugins RCP, siguiendo los pasos de un tutorial y mirando la documentación de eclipse RCP y SWT. También me instalé el cliente de *git* en línea de comandos.

Una vez puesto en marcha el *git* por parte de la empresa me pasaron mi usuario y contraseña tanto del control de versiones como del acceso a la base de datos. Procedí a descargame el proyecto desde el *git* con la orden:

```
$ git clone ssh://juanaguilera@www.sironta.com/home/juanaguilera/sironta
```

Y después creé una nueva rama para no trabajar con la *master*:

```
$ git branch juanaguilera-2011-04-repositories
```

```
$ git checkout juanaguilera-2011-04-repositories
```

Aún así, no funcionó correctamente a la primera porque faltaba por incluir el directorio de funcionalidades (*features*) y como íbamos mal de tiempo decidimos optar por que me pasaran el proyecto entero. Así pues, una vez descargado el proyecto lo abrí con *eclipse* y al compilar me daban fallos. Buscando información de los mismos comprobé que tenía dependencias que cumplía así que las instalé desde el mismo *eclipse*. Las dependencias en concreto eran el paquete *jgit* y las bibliotecas gráficas *swt*.

6. Análisis

4. Base de datos

Actualmente se gestionan los dos diferentes tipos de sala a partir de dos tablas. La principal, *rooms*, guarda la información común a los dos tipos, esto es, el identificador único, el nombre de la sala, y el indicador único del propietario de dicha sala. La otra tabla, *dedicatedRooms,* es específica para las salas dedicadas e incluye como clave foránea el identificador de la sala a la que hace referencia, por lo tanto tan solo es necesario que incluya la información exclusiva que hace que una sala sea dedicada.

Las salas dedicadas son una solución que se puso para aquellos casos en los que fuera necesario tener un repositorio siempre disponible. Estas, se alojan en el servidor de *sironta* y necesitan una URL de acceso, un usuario y una contraseña, que son los datos que se guardan en esta tabla de salas dedicadas.

La similitud de estas últimas salas con el objetivo principal de este proyecto es evidente por lo que será un buen punto de partida. En una primera fase hemos decidido que se aprovecharán estas tablas ya que tanto si el nuevo repositorio está en un ftp, sftp, servicio de almacenamiento online, … necesitará la URI, el usuario y la contraseña.

Además, al estar el protocolo dentro de la URI no es imperativo añadir otro campo ya que se puede extraer de esta y sería mantener datos redundantes.

5. Código

Cuando *sironta* se conecta, primero consulta los metadatos almacenados en local para ver las salas que tiene y entonces las carga en el programa. Esto es así para permitir el modo *offline*, o sea que si no hubiera conexión el usuario al menos podría trabajar con lo que tiene en su disco duro.

Después hace peticiones GET al servidor para consultar las salas que tiene y poder sincronizar los datos si es necesario. Este punto es importante ya que habrá que modificar la manera de procesar estas peticiones para poder obtener la URI del repositorio y de esta el protocolo y los datos pertinentes a la conexión al repositorio remoto.

Analizando el código hemos visto que podemos aprovechar la creación de las salas ya que su comportamiento se asemeja mucho al de las salas dedicadas. El primer paso será unificar los dos constructores y aprovechar la clase *DedicatedServerInfo*. Esta clase guarda la información almacenada en correspondiente tabla en el servidor, es decir, URL, usuario y contraseña. Al tener que gestionar diferentes protocolos habrá que añadir un nuevo campo que lo refleje.

Después de analizar el código, he visto que hay otro punto importante. Tenemos una clase abstracta que representa los repositorios de *sironta*, *SirontaRepository* de la cual heredan dos clases, *RoomGuestSirontaRepository* y *RoomOwnerSirontaRepository* dependiendo de si se es el propietario de la sala o no. De la primera hereda *RoomDedicatedSirontaRepository* dado que todas las máquinas se conectaran al servidor dedicado. *SirontaRepository* dispone, entre otros, de dos métodos abstractos que tendrán que implementar sus hijos. El que variará, en función del tipo de repositorio nuevo añadamos, es *getRemoteConfig* que como su nombre indica nos dará la configuración necesaria para conectarse al repositorio.

Así pues, habrá que añadir una clase por cada nuevo tipo de repositorio que herede de *RoomGuestSirontaRepository* y habrá que sobrecargar el método *getRemoteConfig*.

Por lo tanto, la planificación inicial en la que partíamos de que había crear un nuevo *plugin* ha habido que cambiarla ya que no lo necesitamos.

6. Especificación de la URI

Como he comentado, el campo URI de las salas cobra especial importancia ya que en ellas se especificará el protocolo. La primera fase tiene que ser soportar los dos protocolos actuales pero previendo que se puedan soportar el resto.

Mi primera idea era usar una cadena a semejanza del protocolo ftp. Así pues la cadena URI quedaría definida de la siguiente manera:

```
IdProt://usuario[:password]@host[:puerto][/ruta]
```

- IdProt: Identificador de protocolo. (ssh, ftp, file, sironta, ...)
- usuario: En caso que no sea necesario, por ejemplo en el caso del repositorio en local se pondrá el nombre del propietario de la sala.
- password: [opcional] Contraseña de acceso al repositorio.
- host: Dominio o dirección IP del equipo donde esté el repositorio.
- puerto: [opcional] Si no se usa el puerto predeterminado para este servicio hay que especificarlo.
- ruta: [opcional] Ubicación dentro del árbol de directorios del equipo donde esté el repositorio. No necesaria si usa la ubicación predeterminada.

Ejemplos:

```
ssh://juan:su_password@sironta.com
```

```
file://juan@192.168.1.10/directorio_compartido
```

7. Adaptación de la URI al modelo actual

En el modelo actual, en la información de conexión de la sala, ya se incluye una URI con el usuario y contraseña por separado. He decidido emplear la misma especificación pero sin usar estos dos campos para no tener que cambiar tanto el protocolo y sobretodo teniendo en cuenta la interfaz de usuario. En la interfaz de usuario he preferido poner el usuario y contraseña por separado pensando en futuros protocolos. Si bien, en un ftp, ssh, y similares el tener estos campos dentro de la URI es normal, en casos como Amazon S3 o dropbox no será así.

8. Protocolos soportados

En un principio creía que *sironta* tenía implementados los protocolos de *git*, *http*, *https*, *ftp* y *ssh* para el intercambio de información. Pero probando el protocolo *ftp* he podido comprobar que la capa de transporte no estaba terminada. En una primera aproximación he desarrollado el protocolo *file* que será muy útil en entornos LAN ya que permitirá adaptar Sironta a las necesidades empresariales.

Se ha modificado la estructura del programa para que sea fácil la inclusión de nuevos protocolos. La intención del proyecto es también el poder usar los servicios de almacenamiento online como Dropbox o Amazon S3 pero por razones de tiempo ha sido imposible realizarlo. En el caso de Amazon S3 la biblioteca *jgit* permite el acceso a este servicio online con lo que su implementación será más sencilla.

9. Entorno de pruebas

La empresa me ha facilitado el acceso a la base de datos de desarrolladores y mediante el cambio de una variable en el código fuente tengo acceso a esta en el programa. Debido a que las salas se consultan al servidor mediante peticiones GET he tenido que editar la url de la petición para apuntar a mi servidor web, en *localhost,* donde he podido editar más fácilmente la cadena que tiene que interpretar el programa. La petición que mi servidor devuelve, una vez adaptado para probar el protocolo *ssh* es la siguiente:

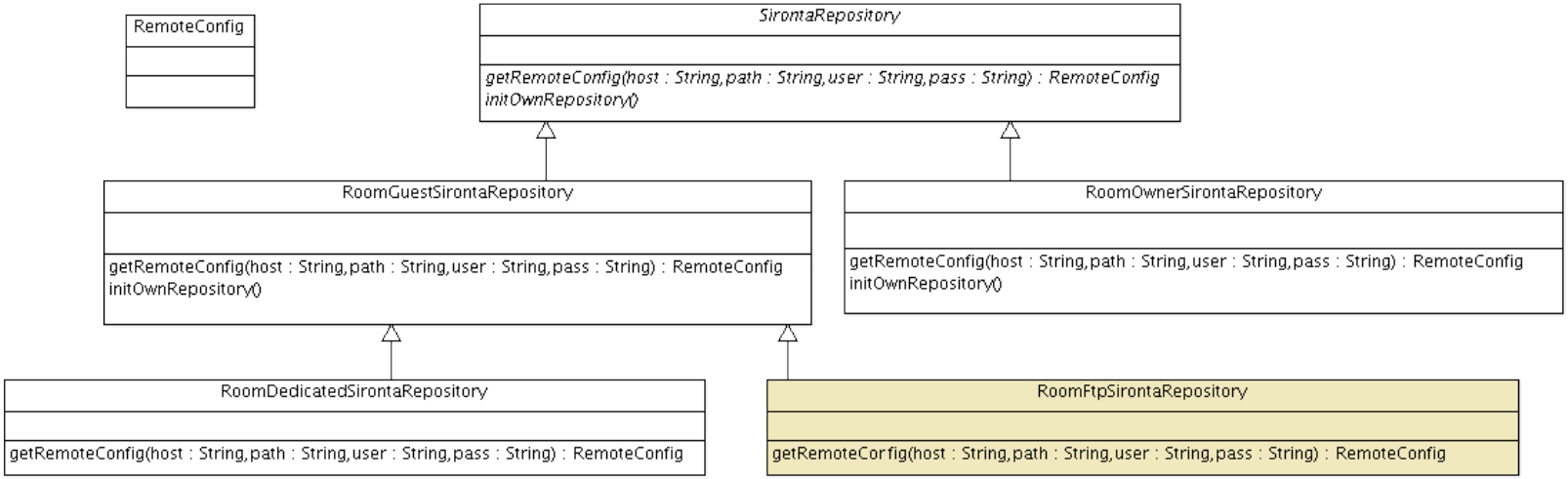
```
_result_:OK

Mi
salaaaa:juanaguileraramos@gmail.com#ssh://localhost/home/juan/salasSironta/Mi_s
alaaaa/docs/remote.git;usuarioSSH;contraseñaSSH#juanaguileraramos@gmail.com;
```

Este formato sigue la especificación existente para los casos de los repositorios dedicados. Esta es la siguiente:

```
nombre_sala:id_propietario#URI;usuarioServidorRemoto;contraseñaServidorRemoto#i
d_usuario1;[id_usuario2;]
```

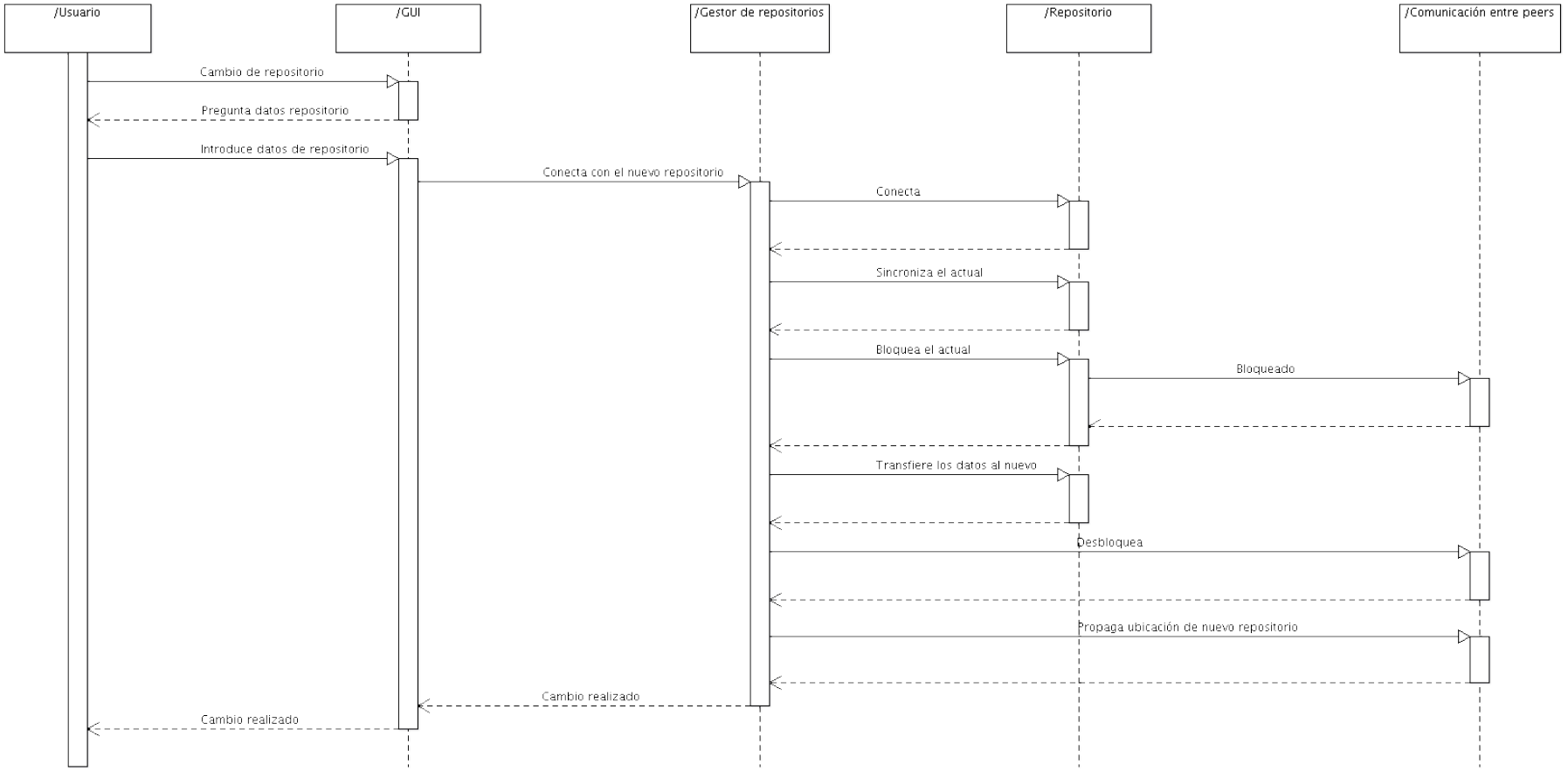
7. Diagrama de clases

El actual diagrama de clases se adapta bastante bien a las modificaciones que hay que realizar. Esto es debido a que ya existían dos tipos de repositorios y en el diseño original se usó herencia. Por cada nuevo tipo de repositorio que se añada se tendrá que seguir el mismo esquema, añadir una clase que herede de *RoomGuestSirontaRepository* e implemente el método *getRemoteConfig*. En el siguiente diagrama, a modo de ejemplo, he puesto la clase *RoomFtpSirontaRepository* que cumple dicho esquema.



8. Diagrama de secuencia

En este diagrama de secuencia se puede ver de manera abstracta cuales son los pasos para cambiar un repositorio de ubicación.



9. Implementación

Los cambios en el código reflejan con bastante fidelidad la planificación inicial excepto en un caso que no había tenido en cuenta. Y es que, hay que hacer una diferenciación del protocolo cuando se utiliza el sistema actual de Sironta. Esto es así debido a que entonces el propietario hace de servidor y por tanto tiene que hacer las modificaciones en el GIT de manera local. Además debe de usar el método *registerHTTPProcessor* de la clase *RoomOwnerSirontaRepository* para el registro del servicio.

Por lo demás, se han seguido las pautas y se han creado repositorios heredados de la misma clase. Estos repositorios se crean de uno u otro tipo dependiendo del protocolo que utilicen, ya que al mismo protocolo lo único que cambiará será su ubicación y sus datos de autentificación. Para ello creé un *enum* que sirve para almacenar los protocolos que soportamos actualmente. A día de hoy el *enum Protocol* tiene los valores SIRONTA, SSH y FILE a espera de que se puedan soportar más.

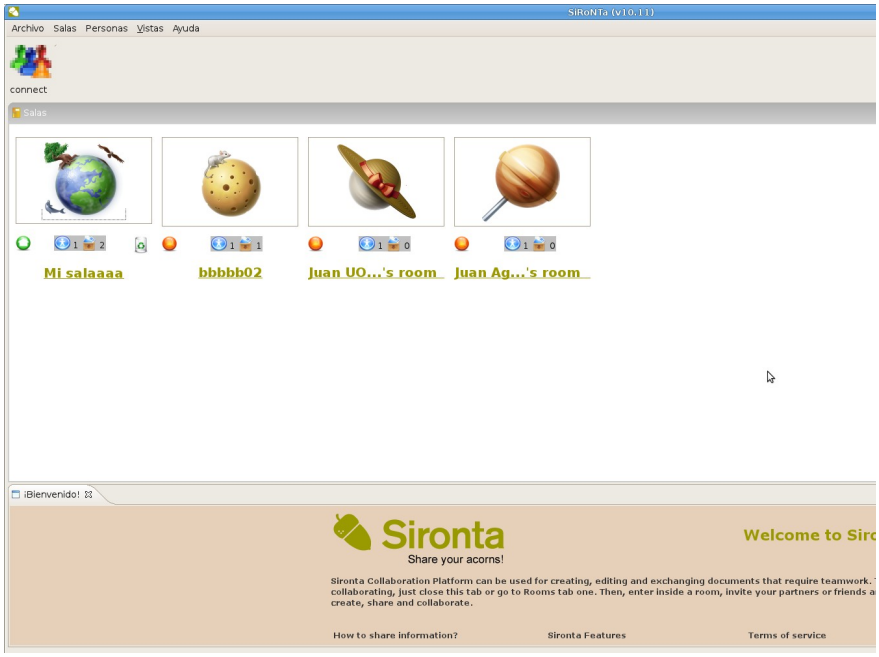
Así, con un *switch-case* de este *enum* en el constructor de la clase *DocumentModel*, se puede fácilmente añadir un nuevo tipo. Este constructor se llama en el momento que se llama al método *getDocumentModel* de la clase *RoomImpl*. De este constructor también he quitado el campo booleano que indicaba si era dedicado o no, ya que ahora nos guiaremos por el criterio mencionado. Este cambio conllevó la supresión de la variable miembro correspondiente en esta clase.

Por coherencia con el nuevo esquema he renombrado la clase *RoomDedicatedSirontaRepository* a *RoomSshSirontaRepository* ya que este es el protocolo que utiliza. Así, el protocolo *file* añadido por mi, coge esta especificación nominal en su clase llamándose *RoomFileSirontaRepository*. También he renombrado la clase DedicatedServerInfo por la actual ServerInfo, además de que le he añadido el campo *roomProtocol*.

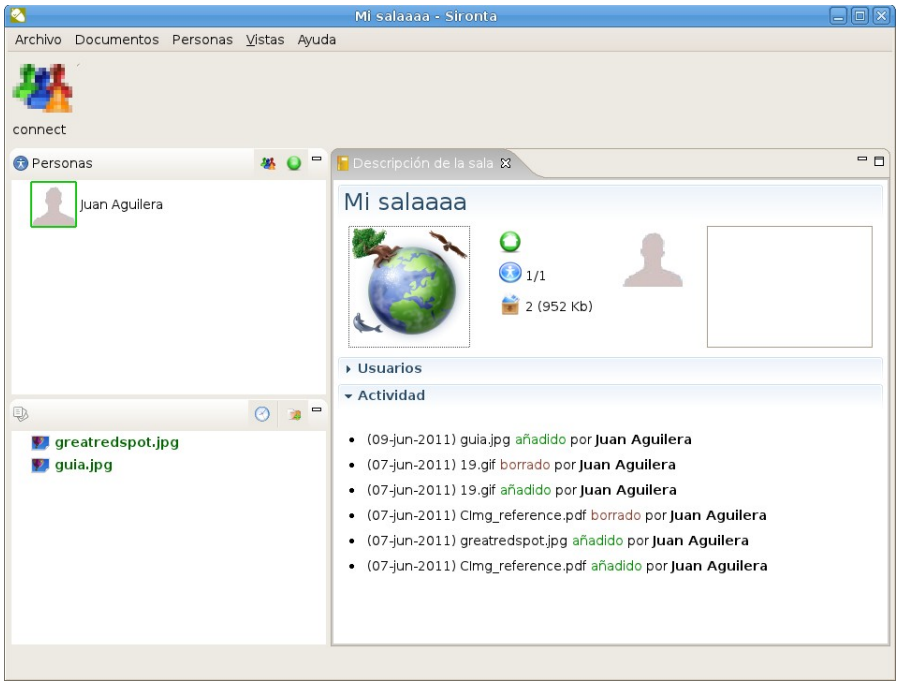
En la parte de la interfaz gráfica he tenido que modificar dos clases, *RoomDescriptionDetailsComposite* donde se diseña el GUI propiamente dicho y la clase *RoomDetailsDialog* que es el diálogo que engloba a la anterior y donde he tenido que añadir los datos relativos al *ServerInfo* para cuando se le da a guardar los cambios.

10. Interfaz de Usuario

Los cambios realizados en la interfaz de usuario se han concentrado en una sola pantalla, la de configuración de la sala. A esta configuración sólo se puede acceder si se es el dueño de la sala. Para llegar a ella tenemos que entrar en la sala pulsando sobre su icono en la pantalla principal a la que accedemos después de identificarnos.



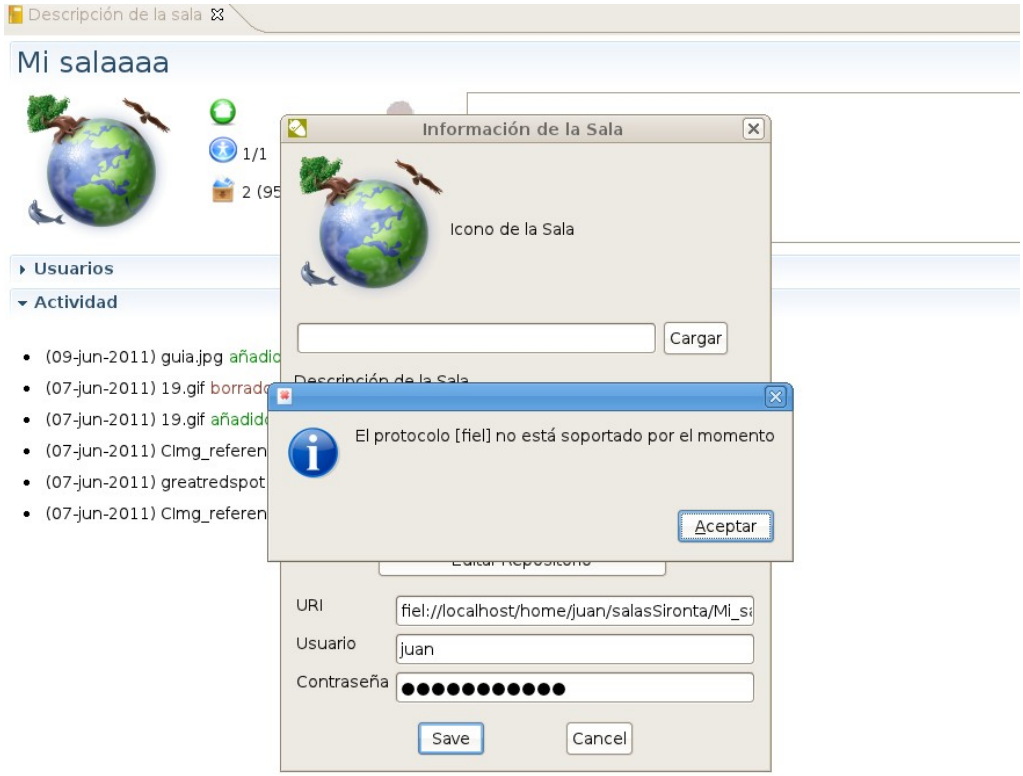
En este caso he usado la primera sala de ejemplo que creé.



En ella podemos ver el historial de cambios en el repositorio. Si pulsamos en la imagen de la sala llegamos a la información de la misma que es la pantalla que he tenido que modificar.

A esta pantalla le he añadido el botón “Editar Repositorio”, y los campos URI, Usuario y Contraseña. Cuando se carga la ventana estos campos están desactivados por defecto para evitar modificaciones indeseadas. Para editarlos tan sólo tenemos que pulsar el botón, y ya los tendremos disponibles. El campo contraseña lo he puesto con la ocultación de caracteres activa por mayor seguridad. A continuación podemos ver un detalle de los nuevos campos.

He tenido en cuenta que es posible el error al meter la URI. Si se falla en el host, el usuario o en el path, lo único que ocurriría sería que no se podría conectar al servicio y sironta ya muestra un mensaje de error en estos casos. La validación del protocolo es importante ya que es la que decide qué configuración de repositorio se coge. Por tanto, si alguien introduce uno incorrecto, hay que filtrarlo en esta pantalla. Por ejemplo, aquí el usuario se equivocó y tecleó incorrectamente el protocolo "file" como "fiel" y el programa muestra la correspondiente advertencia.



Una vez que se introducen correctamente los campos y se pulsa "Guardar" los cambios son reflejados en el *DocumentModel* y guardados en disco actualizando los metadatos anteriores de la caché.

11. Inclusión de un nuevo tipo de repositorio

Tal y como ha quedado la estructura del programa la inclusión de un nuevo tipo de repositorio necesita de tres pasos.

1. Antes que nada hay que crear una clase derivada de *RoomGuestSirontaRepository* y sobrescribir el método *getRemoteConfig*. Este método nos permite configurar todos los aspectos necesarios para la conexión al repositorio *git* mediante la librería *jgit*.

2. El segundo paso pasa por tener una clase *Transport* adaptada al protocolo. En Sironta actualmente está implementada la clase *FullTransportHttp* que permite la conexión mediante Http y Https. Asimismo mediante la clase *Transport* proporcionada por *jgit* se puede acceder al servicio de almacenamiento online de Amazon.

3. Por último hay que crear una nueva entrada en el *enum Protocol* que he creado en la clase *RepositoyProcessInfo*. Este *enum* guarda los protocolos soportados para la comunicación con el repositorio remoto.

Un ejemplo lo tenemos en el protocolo *file* que sí que he implementado. Este era un primer paso ya que en el entorno empresarial es habitual ver un servidor con un directorio NFS o una unidad de red compartida. Para ello, el segundo paso, el de implementar la clase que gestione la capa de transporte, no ha sido necesario. Tan solo he tenido que crear la clase *RoomFileSirontaRepository* que hereda directamente de *RoomGuestSirontaRepository* tal y como explicaba anteriormente, con el método *getRemoteConfig* sobreescrito cambiando el protocolo que cogí de base, el *ssh*, y cambiándolo por *file*.

Para poder usar esta nueva clase he creado el valor FILE en el *enum Protocol*. Como en ningún lado se muestra el nombre del protocolo de cara al usuario el valor del *enum* está definido como el identificador del protocolo (ssh, ftp, http, …), en mayúsculas.

Con estos pasos ya se tendría que poder ubicar el repositorio en otro servidor o bajo otro protocolo. De la manera que está implementado la interfaz de usuario ya es capaz de determinar a partir de la URI entrada por el usuario si el protocolo está soportado o no, ya que comprueba el *enum* para ver si el valor entrado se corresponde con uno de ellos.

12. Conclusiones

Han sido unos resultados un tanto agridulces, ya que no he conseguido terminar todos los objetivos que me había planteado aunque el principal, que era la adaptación de la plataforma para que aceptara otros protocolos en otras ubicaciones, está conseguido.

En cuanto a la reorganización del código y la asimilación del proyecto creo que he hecho una buena faena. El programa mantiene las funcionalidades de versiones anteriores y añade la de poder cambiar el repositorio a un directorio local, o una unidad de red. Además el repositorio por protocolo *ssh* ahora no tendrá que estar localizado en el servidor web de sironta.com con lo que se le dota de mayor flexibilidad al programa. Cualquiera con un servidor *ssh* puede tener un repositorio de Sironta copiando el directorio de la sala en él.

Ha faltado implementar un protocolo como *ftp*, o *Amazon S3*, ampliamente utilizados, pero al ser un proyecto de software libre podrá ser añadido en un futuro más fácilmente gracias a la estructura que he diseñado. El verdadero punto flaco ha sido que no he tenido tiempo de implementar el traspaso automático del repositorio a la nueva ubicación. Aún así, la localización del repositorio no es algo que se cambia cada día por lo que era más urgente la posibilidad de cambio de ubicación aunque sea a costa de hacerlo manualmente.

Del proyecto en sí, me ha aportado la experiencia de programar con un paradigma bastante distinto del que estaba acostumbrado. He estado 3 años desarrollando en c++ y con el cambio me he encontrado con un lenguaje moderno y con una api muy bien diseñada y muy amplia. El lenguaje java lo conocía de la carrera, pero nunca había tocado un proyecto basado en plugins *RCP*, como este y me ha parecido un sistema muy interesante por su modularidad. Tampoco había creado interfaces gráficas usando SWT por lo que también he aprendido un poco como se componen los *widgets* con sus respectivos *layouts*.

Además he aprendido como funciona el sistema de control de versiones GIT, muy usado en el software libre por lo que creo que me vendrá bien en un futuro.

13. Bibliografía

- **Lars Vogel** *Eclipse RCP Tutorial*
  http://www.vogella.de/articles/EclipseRCP/article.html

- **Oracle** *Java API*
  http://download.oracle.com/javase/6/docs/api/index.html

- **Jarvana** *Documentación jgit*
  http://goo.gl/3w0Ur

- **Eclipse** *SWT: The Standard Widget Toolkit*
  http://www.eclipse.org/swt