Integración de SyncML con ownCloud de KDE para sincronización de información personal

MEMORIA DE SISTEMAS INFORMÁTICOS

Antonio José Gallo Sánchez

Facultad de Informática
Universidad Complutense de Madrid

Septiembre 2011

Documento maquetado con TEXIS v.1.0+.

Integración de SyncML con ownCloud de KDE para sincronización de información personal

Memoria de Sistemas Informáticos

Dirigida por el Doctor
José Luis Risco-Martín

Facultad de Informática
Universidad Complutense de Madrid

Septiembre 2011

Resumen

El proyecto desarrollado consiste principalmente en el desarrollo de un servidor SyncML en ownCloud de KDE. ownCloud es una plataforma de "auto-hosting" . Significa básicamente que provee de unas cuantas aplicaciones básicas como navegador de ficheros y visores y editores de los formatos más populares. También provee de un API para permitir a los desarrolladores desarrollar aplicaciones web para esta plataforma, y a los usuarios instalarlos directamente desde un repositorio. SyncML es un protocolo estándar para sincronizar información genérica, y tiene una arquitectura cliente-servidor. Normalmente SyncML se utiliza para la sincronización de información personal, pero puede utilizarse para cualquier tipo de datos. También, una meta de este proyecto es el desarrollo de dos aplicaciones que utilizan esta nueva característica. Un visor y editor para los contactos sincronizados con ownCloud, y un cliente de escritorio SyncML para sincronizar cualquier tipo de datos con ownCloud. Este proyecto ha sido desarrollado junto al equipo ownCloud de KDE, y sera distribuido junto con ownCloud en las futuras versiones.

The developed project mainly consits in the development of a SyncML server into KDE's ownCloud. ownCloud is a "auto-hosting" platform. It roughly means that it provides a few core applications as a file browser and viewers and editors for the most popular file types. It also provides an API to let the developers develop web apps for this platform, and the users install them directly from a repository. SyncML is a standard protocol to sync generic information, and it has a client-server architecture. Usually SyncML is used for PIM (Personal Information Management) data syncing, but it can be used for any kind of data Also as a goal for this project is developing two applications which uses this new feature. A web viewer and editor for contacts synced with ownCloud, and a SyncML desktop client to sync any kind of data with ownCloud. This project has been developed with KDE's ownCloud team, and it is going to be distributed with ownCloud in future versions.

Índice

Índice de figuras

Índice de Tablas

Parte I

Trabajo desarrollado

Esta parte supone el grueso de la memoria. Contiene todo lo relevante al proyecto excepto los apéndices, que están en la segunda parte

Capítulo 1

Introducción y trabajo relacionado

It's stupidity. It's worse than stupidity:
it's a marketing hype campaign.
Richard Stallman en Johnson (2008)

Resumen: Durante esta breve introducción se explicarán someramente los conceptos utilizados a lo largo del trabajo, su papel en el proyecto, las herramientas ya existentes, y finalmente las mejoras y contribuciones que supone este proyecto.

1.1. Introducción

La computación en la nube es el modelo computacional de mayor crecimiento actualmente.

Las razones de este crecimiento son las ventajas que supone, tanto para empresas como para usuarios.

Con respecto a las empresas, la posibilidad de aplicar la economía de escala a recursos computacionales implica que determinadas empresas se dediquen a ello, ofreciendo a sus clientes mejores precios de los que a estos clientes les supondría el tener sus propios recursos computacionales equivalentes.

Además de las evidentes ventajas que supone la externalización de un servicio (reducción de personal especializado, eliminación de responsabilidad de problemas derivados del servicio, etc) hay determinadas ventajas en concreto del cloud computing para los clientes, como la posibilidad de pagar

sólo por el uso del servicio (lo que por ejemplo se ajusta perfectamente a los flujos de consumo de recursos computacionales a ciertas horas del día de las aplicaciones web) y la posibilidad de disponer aparentemente de infinitos recursos computacionales bajo demanda. En Armbrust et al. (2009) se trata este tema con la profunidad adecuada.

Según SmugMug, una página web dedicada al almacenamiento y gestión de fotografías que utiliza Amazon S3 (Uno de los múltiples servicios Cloud Computing de Amazon, éste para externalizar el alojamiento), el ahorro que les supone utilizar este servicio en vez de disponer de sus propios centros de datos es aproximadamente un millón de dólares anuales (MacAskill (2006)).

Es por esto que casi todas las grandes compañías de informática están poniendo cada vez más empeño en ofrecer más y mejores recursos cloud computing: SalesForge.com (Forge.com) Amazon (Amazon EC2 y S3), Microsoft (Windows Azure), Google (Google Apps Engine), IBM (SmartCloud)...

Pero ¿Qué supone el Cloud Computing para los usuarios finales?. La mayoría de las empresas que contratan servicios Cloud Computing (hardware-as-a-service) [1] lo hacen para utilizarlos en aplicaciones web para el gran público (software-as-a-service). Por lo tanto, podemos deducir que el Cloud Computing para los usuarios finales supone aplicaciones web de gran calidad. Las ventajas de las aplicaciones web son evidentes: compartición de datos, versionamiento automático, backups transparentes, ubicuidad... Mejoras al modelo existente que, en definitiva, suponen una mejora fundamental en la usabilidad de las computadoras.

Este documento describe la implementación de un servidor SyncML de propósito general para una aplicación web que a su vez sirve como framework, ownCloud.

También se implementan dos aplicaciones que explotan esta nueva funcionalidad añadida a ownCloud; un visor y editor de contactos personales para los contactos sincronizados con clientes SyncML destinados a sincronizar información personal, y un cliente genérico para sincronizar cualquier tipo de información con ownCloud.

Según palabras de sus desarrolladores, el objetivo de ownCloud es:

"Ofrecer acceso universal a tus ficheros gracias a un interfaz web o WebDAV. También proveer de una plataforma para ver y sincronizar tus contactos, calendarios y marcadores con todos tus dispositivos, y permitir edición básica a través de la web. La instalación tendrá requisitos mínimos para el servidor, y no requierirá permisos especiales. ownCloud será extensible vía una simple pero potente API para aplicaciones y plugins" own (2011)

Es decir, ownCloud pretende ser una aplicación web en la que una persona pueda almacenar, compartir, editar y sincronizar su información personal. Además, mediante el API del que provee, permite a los desarrolladores crear aplicaciones que puedan extender la funcionalidad de ownCloud, como por

[1] O más concretamente: infraestructure-as-a-service

ejemplo crear un editor web para determinado tipo de archivo.

1.2. Trabajo relacionado

Existen múltiples aplicaciones en la nube, que de una manera u otra cubren las funcionalidades de ownCloud, pero ninguna de ellas las cumple todas a la vez, si no que son servicios aislados para resolver una necesidad en concreto. Como por ejemplo, Google dispone de múltiples aplicaciones web, como cliente web de correo electrónico y contactos (Gmail) o un servicio de almacenamiento y gestión de fotografías (Picasa). Estas aplicaciones, aunque comparten cierta cantidad de datos entre sí, funcionan principalmente de manera aislada una de otra. Tampoco existe la posibilidad de extender su funcionalidad mediante aplicaciones de terceros.

ownCloud es el primer esfuerzo desarrollado para tener toda la información personal online unida de manera congruente, sin embargo, su integración con el escritorio es muy rudimentaria, la cual hasta el momento se reduce a la posibilidad de montar los directorios existentes en ownCloud mediante WebDAV

Por lo que añadir a ownCloud de la capacidad de sincronizarse de manera robusta con algún tipo de aplicación de escritorio tal y como hacen algunas aplicaciones comerciales se presenta como una alternativa muy interesante

Para ello, haremos uso de de un servidor SyncML de propósito general (implementado como una aplicación de ownCloud), lo que permite añadir de manera estándar y abierta, la capacidad de sincronizar virtualmente cualquier tipo de información con ownCloud, con cualquier cliente que implemente el estándar.

Aunque SyncML es utilizado típicamente para sincronizar información personal (contactos, tareas y calendario), su especificación es genérica y permite cualquier tipo de archivo. El servidor implementado permite sincronizar directamente con ownCloud esta información personal mediante cualquier cliente SyncML (que al ser estándar, tiene clientes en casi todas las plataformas), y mediante un cliente específico sincronizar cualquier tipo de información, ya que apenas existen clientes SyncML genéricos.

Nombre aplicación	Dropbox	Google	iCloud	ownCloud+SyncML
Desarrollador	Dropbox Inc	Google Inc	Apple Inc	KDE
API disponible	√	√	√	√
Fotografías	√	√	√	√
Videos	X	√	√	√
Archivos genéricos	√	X	X	√
Música	X	√	√	√
Precio ponderizado 50GB/año	119,88$	12,5$	100$	Coste hardware
Tareas	X	√	√	√
Contactos	X	√	√	√
Calendario	X	√	√	√
Documentos	X	√	√	√
Software libre	X	X	X	√

Tabla 1.1: Comparativa de ownCloud con las aplicaciones desarrolladas en este proyecto y otros servicios alternativos

En el próximo capítulo...

Durante el próximo capítulo analizaremos en mayor profundidad las contribuciones de éste proyecto a ownCloud.

Capítulo 2

Contribuciones del proyecto

If I have seen further it is only by standing on the shoulders of giants.

Newton (1976)

Resumen: Durante este capítulo, se esquematizará el trabajo desarrollado y se explicará para facilitar más tarde al lector la comprensión del mismo

2.1. Introducción

Como ya se ha mencionado anteriormente, el proyecto consiste en el desarrollo e implementación de un servidor SyncML como aplicación para ownCloud, y el desarrollo de dos aplicaciones que amplían y explotan esta característica en ownCloud: Un visor y editor web de los contactos sincronizados mediante un cliente PIM SyncML cualquiera, y un cliente de escritorio también basado en SyncML para sincronizar cualquier tipo de archivo con ownCloud.

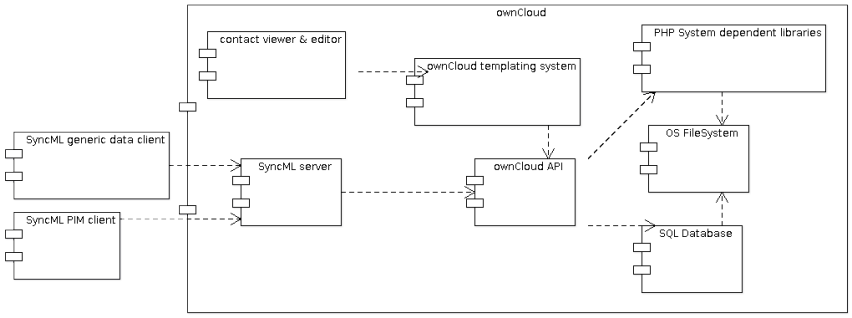

Figura 2.1: Diagrama de componentes simplificado de ownCloud y las aplicaciones desarrolladas.

La figura 2.1 muestra un diagrama de componentes simplificado donde se muestran las relaciones a nivel estructural entre las aplicaciones desarrolladas y ownCloud. Las aplicaciones desarrolladas son los componentes de nombre `SyncML generic data client`, `contact viewer & editor` y `SyncML server`.

2.2. Esquema

A continuación pasaremos a detallar los componentes implementados en el proyecto, y a describir someramente su relación con el resto del sistema

Como se aprecia en la figura 2.1, nuestro servidor SyncML (el componente con el nombre `SyncML server`) se implementa como aplicación para ownCloud, utilizando el API proveída por este (`ownCloud API`). Para esto, se ha partido de una implementación ya hecha de un servidor muy primitivo de SyncML, `PHPSyncML` (PHP (2011)) y se ha modificado y adaptado a ownCloud y los propósitos del proyecto

También podemos apreciar la relación de clientes, `SyncML PIM client` y `SyncML generic data client` con ownCloud. Éstos se comunicarán con el servidor SyncML implementado y este a través del API de ownCloud, escribirá los cambios necesarios resultado del proceso de sincronización con los clientes.

Las funciones de estos dos clientes son:

- Clientes SyncML para información personal `SyncML PIM client`: Estos clientes, que son la inmensa mayoría de los clientes que implementan SyncML que se encuentran en el mercado, se limitan a implementar el protocolo SyncML para sincronizar información personal. Como estándar de facto, esta información se intercambia como ficheros vCard

(para contactos) e iCalendar (para tareas y calendario)

- Clientes SyncML genéricos `SyncML generic data client`: Ya que en el mercado no se encuentra ningún cliente SyncML genérico de calidad aceptable, hemos optado por implementar uno para sincronizar cualquier tipo de archivo con ownCloud. Hay que resaltar que el protocolo SyncML es un protocolo de sincronización genérico para cualquier tipo de archivo, y por tanto la creación de un cliente por una tercera parte con las mismas características no debería suponer un problema.

Otro componente de la figura 2.1 objeto de este trabajo es `contact viewer and editor`. Se trata de un ejemplo que de una manera muy visual muestra una primera funcionalidad práctica del proyecto. Este editor web permite vistualizar y editar los contactos que hayamos sincronizado con ownCloud mediante cualquier cliente `SyncML PIM client`, para en la próxima resincronización propagar los cambios.

El añadir estas funcionalidades a ownCloud, en la práctica, permitirá el alojamiento de toda la información personal necesaria en un servidor ownCloud, y su sincronización con toda suerte de dispositivos, como teléfonos móviles o computadoras personales

En el próximo capítulo...

Durante el próximo capítulo se hablará del estado del arte. Qué aproximaciones existen en el mercado a las necesidades que cubre ownCloud, y las ventajas y desventajas de las mismas, incluyendo las de ownCloud.

Capítulo 3

Cloud Computing

The computer industry is the only industry that is more fashion-driven than women's fashion. Maybe I'm an idiot, but I have no idea what anyone is talking about. What is it? It's complete gibberish. It's insane. When is this idiocy going to stop?

Larry Ellison en Farber (2008)

Resumen: Durante este capítulo se hablará más profundamente de qué es el cloud computing, y un análisis y comparativa de las distintas aplicaciones que comparten las características más importantes de ownCloud, y su relación con el cloud computing.

3.1. Introducción

El cloud computing es un paradigma informático para habilitar acceso on-demand y conveniente en red a un conjunto de recursos de computación compartidos que pueden ser rápidamente proveídos y lanzados con un esfuerzo de gestión mínimo o de interacción con el proveedor de servicios Mell y Grance (2009)).

Aunque la computación en la nube se está imponiendo cada vez más como modelo computacional, su concepción como paradigma computacional no es nueva en absoluto.

Ya en 1960, John McCarty (comúnmente conocido por ser el inventor del lenguaje de programación Lisp) profetizó el nuevo modelo computacional escribiendo *"Quizás la computación sea algún dia organizada como utilidad pública"* (Abelson y Hal (1999))

Sin embargo, no fue hasta la década de los 90 cuando se empezó a concebir como modelo viable el antecesor del cloud computing: el *grid computing*. El terminó fue acuñado inicialmente por Ian Foster y Carl Kesselman en Foster y Kesselman (1999) como metáfora entre el conectarse a red eléctrica para obtener energía (sin tener una central eléctrica en tu negocio) y lo que supondría conectarse a una red para obtener recursos computacionales.

El grid computing puede entenderse como un subconjunto de lo que comprende el cloud computing, sin embargo puede apreciarse muy intuitivamente la evolución lógica de un modelo a otro.

El grid computing se trata únicamente de crear una red de recursos computacionales distribuidos que trabajen para un fin común. Esto nos permite tener una capacidad de cómputo virtualmente tan grande como computadores haya conectados a la misma, y además la posibilidad de implementar con relativa facilidad añadidos como tolerancia a fallos o redundancia.

Utilizando internet como esa red, aplicándole una economía de escala a este concepto y ofreciendo de manera práctica estos recursos como producto bajo demanda, podemos asentar las bases del cloud computing actual.

Sin embargo, ofrecer estos recursos de manera práctica fácil de utilizar a los potenciales clientes no es una tarea sencilla. ¿Cómo ofrecer los recursos de un grid muy potente, a múltiples clientes, con distintas necesidades que además pueden variar en tiempo real? Es aquí donde confluyen un conjunto de líneas de investigación (sistemas distribuidos, virtualización...) para resolver estos problemas de escalabilidad.

Un ejemplo destacable de un sistema destinado a solventar estos problemas es OpenNebula, proyecto muy estrechamente relacionado con la Universidad Complutense de Madrid. Sotomayor et al. (2009)

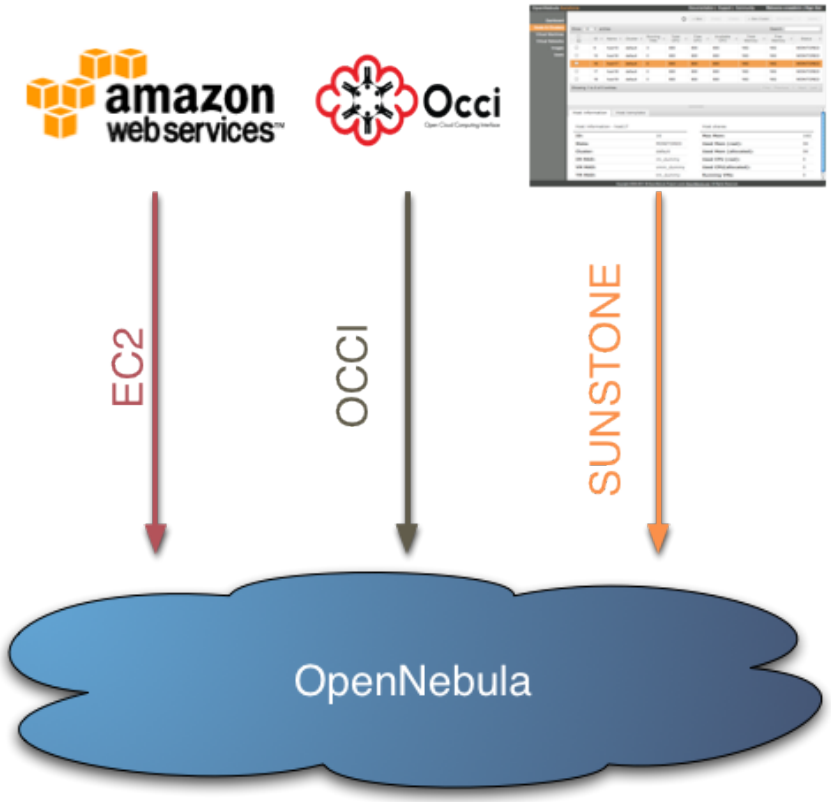

Figura 3.1: Esquema de OpenNebula.

OpenNebula es la primera plataforma de código abierto para desplegar "nubes" que permite flexibilidad en cuanto al backend computacional que emplea, siendo compatible con las principales soluciones del mercado y múltiples configuraciones distintas.

Así como el cloud computing ha ido evolucionando a lo largo del tiempo, las aplicaciones, y su dependencia o relación con recursos cloud computing también lo han hecho.

A continuación pasaremos a describir someramente distintos tipos de aplicaciones en función a sus funcionalidades comparables con ownCloud o su relación con el cloud computing, y la posición de ownCloud y las funcionalidades desarrolladas en este proyecto en esa clasificaciones.

Hay que resaltar que dado el inmenso número de aplicaciones, algunas de ellas no podrían encasillarse únicamente en una clasificación y sus características serían híbridas de dos o mas clasificaciones

3.2. Comparativa y análisis

3.2.1. Clasificación de aplicaciones en función a su dependencia o uso de la nube

3.2.1.1. Aplicaciones web

Este conjunto de aplicaciones son las que más explotan el modelo de cloud computing. Estas aplicaciones sólo necesitan por parte del usuario un navegador web para poder interactuar con ellas. Ningún dato relevante es almacenado en el ordenador del usuario, y normalmente son dependientes completamente de la conectividad de la que dispone el usuario.

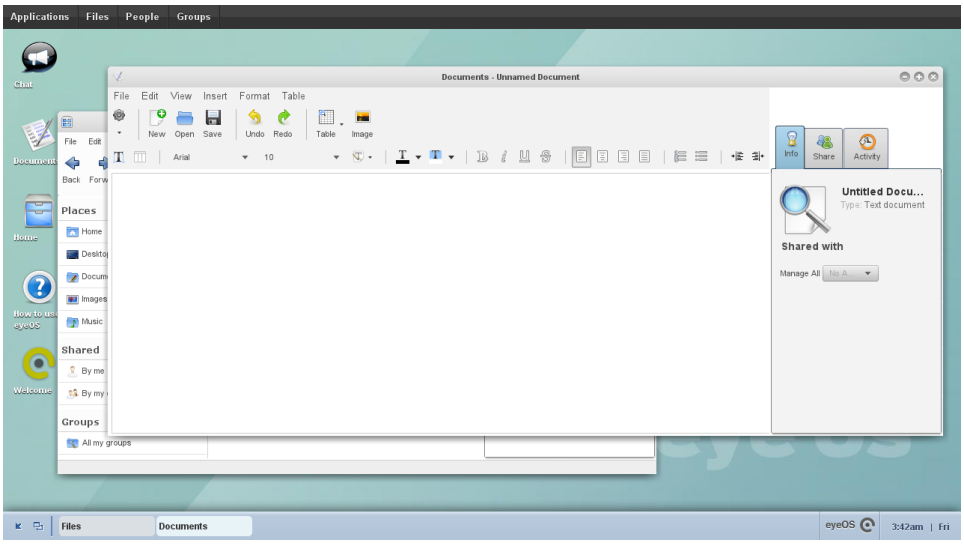

Figura 3.2: eyeOS, un sistema operativo para usuarios domésticos en la nube

El principal problema de estas aplicaciones reside en la conectividad de la que disponga el usuario. Si el usuario pierde su conexión (un escenario poco probable en conexiones fijas, pero cada vez mas común debido al auge de los dispositivos móviles), su trabajo es inmediatamente interrumpido. Existen esfuerzos por parte de la industria por paliar esta desventaja, creando sistemas de almacenamiento temporal a modo de caché en el ordenador del usuario. HTML5 contempla esta funcionalidad, aunque aún no está del todo detallada o implementada Hickson (2011).

Las ventajas de este sistema son obvias. Las actualizaciones, a ojos del usuario, dejan de existir, estando disponibles de forma transparente para todos los usuarios una vez han sido desarrolladas, sin ser lentas, pesadas y molestas. Además, permite al usuario trabajar en prácticamente cualquier máquina y desde cualquier plataforma, aunque obliga al mismo a tener una

conexión a internet disponible.

3.2.1.2. Aplicaciones destinadas únicamente a sincronizar datos genéricos del usuario

Este tipo de aplicaciones están destinadas a sincronizar datos del usuario entre distintas plataformas, usando como intermediario alojamiento en la nube.

Estas aplicaciones son muy prácticas, y permiten al usuario tener cualquier tipo de archivo sincronizado entre sus máquinas, además de normalmente ser directamente accesible desde cualquier equipo gracias a una aplicación web sencilla que a modo de navegador de archivos muestra los archivos que el usuario tiene sincronizados y permite descargarlos o incluso visualizarlos y editarlos.

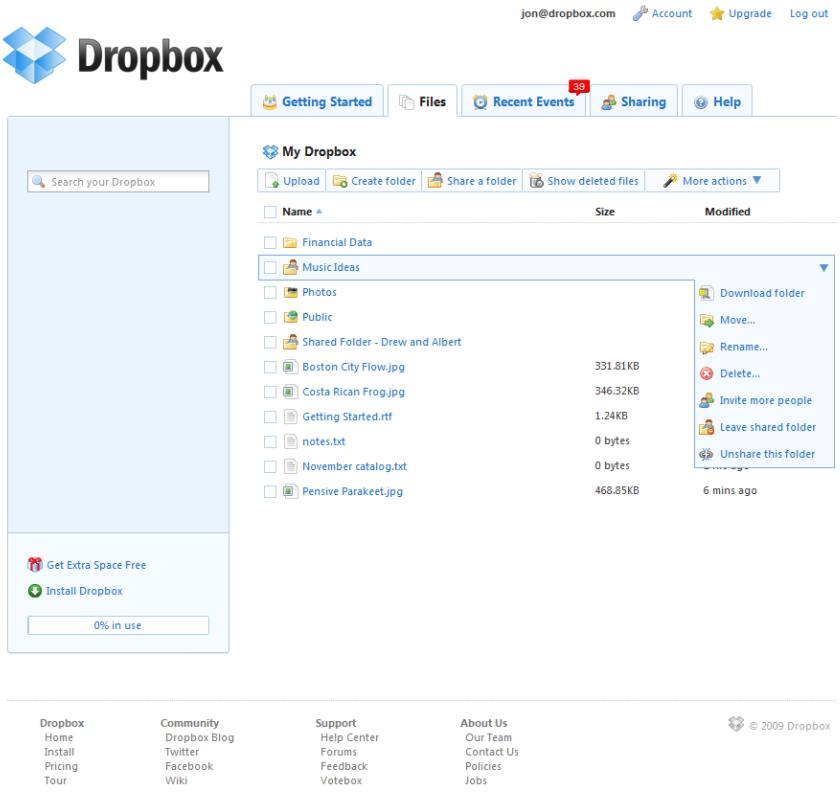

Figura 3.3: Interfaz web de Dropbox

La principal ventaja es su flexibilidad, ya que es el usuario el que elije qué tipos de archivo quiere sincronizar, y con qué aplicación trabaja con ellos.

En la práctica esto implica dotar indirectamente a las aplicaciones estándar de escritorio, normalmente de mayor calidad que sus homólogas web, de ubicuidad para los archivos con los que trabajan. Además, errores en la conexión del usuario tan sólo retrasarían la sincronización temporalmente, permitiendo al usuario seguir trabajando

3.2.1.3. Aplicaciones de escritorio que trabajan con datos de la nube

Este conjunto de aplicaciones son aquellas que siendo de escritorio, están sincronizadas de alguna manera con la nube, permitiendo la sincronización directa de los archivos con los que trabajan.

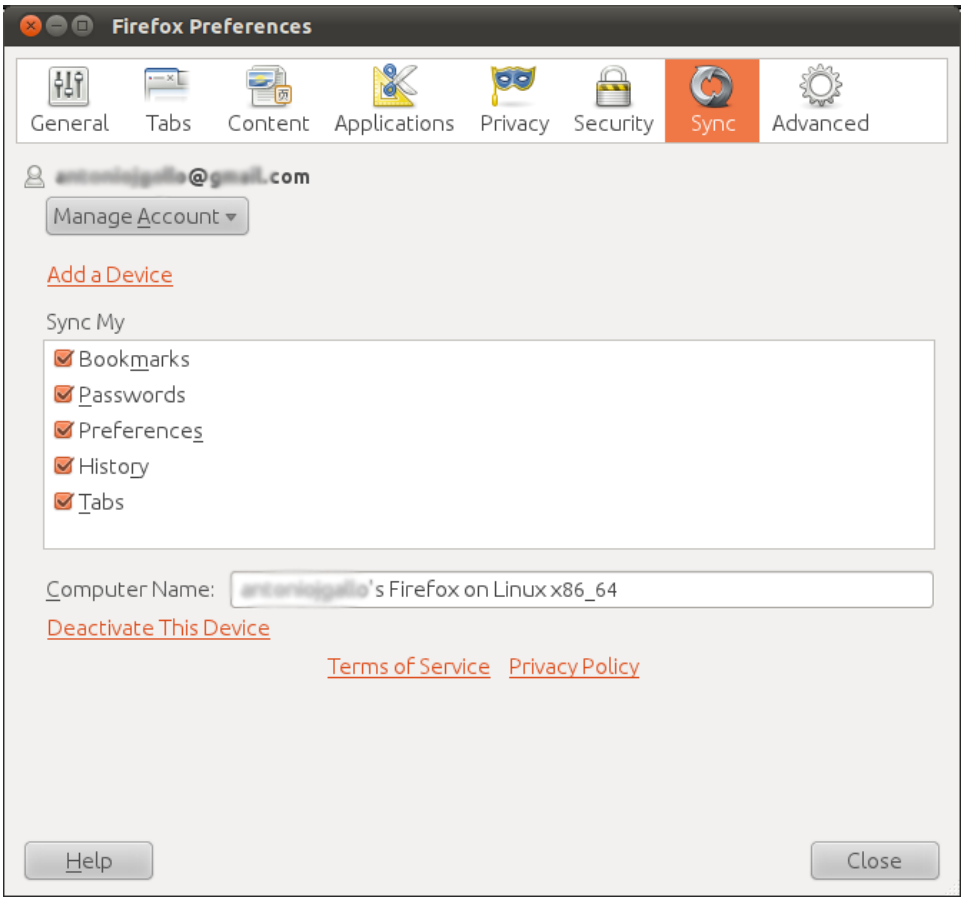

Figura 3.4: Pantalla de configuración de Firefox Sync

Su principal ventaja es la ubicuidad directa de los archivos con los que trabaja, y que los problemas de conexión puntuales de la máquina del usuario no afectan directamente a la productividad del mismo. La desventaja de

este modelo es la rigidez del mismo, no permite la sincronización para nada más que no sea para lo que está diseñada la aplicación. Ejemplos de estas aplicaciones pueden ser el sistema de sincronización de configuraciones de usuario de Android, o algunos componentes de la suite de Microsoft Office con el plugin de sincronización de Google.

Como puede apreciarse, el principal problema de todos estos modelos de aplicaciones en la nube es que de alguna manera están destinados a un fin específico, haciendo uso de servicios web que no son compatibles entre sí ni están integrados. ownCloud es la primera plataforma genérica de cloud computing que permite a los usuarios y desarrolladores hacer uso de todas las ventajas de los modelos expuestos anteriormente sin ninguna de sus desventajas.

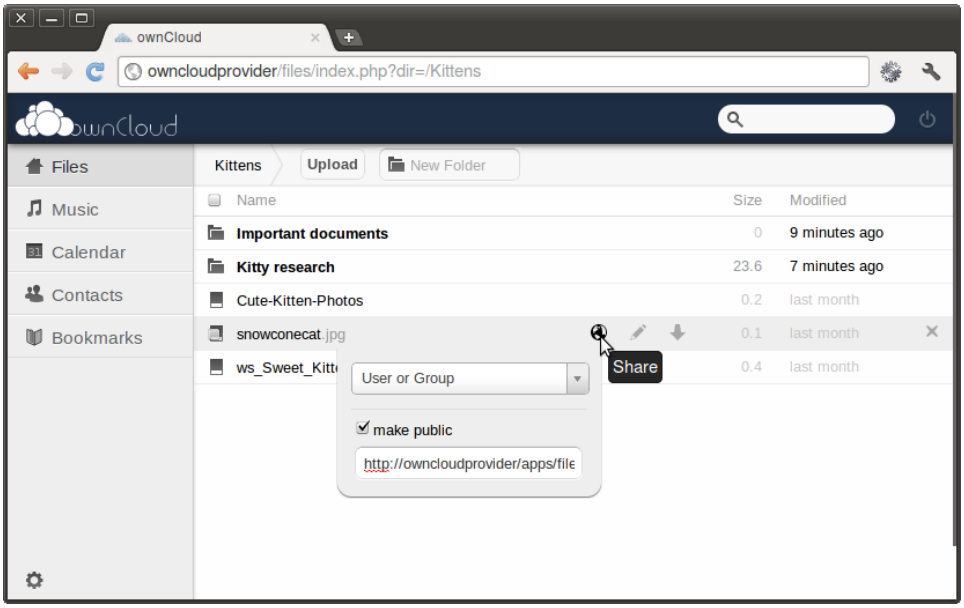

Figura 3.5: Interfaz de ownCloud alpha 2

Esto es, owncloud ofrece un API para desarrolladores, permitiendo a los desarrolladores hacer aplicaciones y plugins para dotar a owncloud de cualquier funcionalidad. Por sí mismo, ownCloud no es más que un API y algunas aplicaciones core distribuidas junto a él (navegador de ficheros, reproductor de música, visualizador y editor de los tipos de achivos más populares entre otras). Este API está disponible para que lo utilicen los desarrolladores, y permite operaciones comunes a todas las aplicaciones, tales como la autenticación, un sistema de ficheros virtual a utilizar por éstas (la idea detrás de este sistema es encriptar los ficheros de los usuarios) etc.

Esto en la práctica, permite programar para ownCloud cualquier tipo

de aplicación que requiera acceso a una *cloud*. Y al ser una plataforma de software libre, y que puede ser alojada en cualquier hosting con unos mínimos requisitos. Hay preparada una versión que puede ser desplegada en Amazon S3 con un mínimo esfuerzo por parte del usuario.

Ejemplos de aplicaciones que pueden ser desarrolladas para ownCloud son:

- Aplicación que integre un servidor firefoxSync con el fin de que el usuario se lo instale con un sólo click y tenga sus marcadores, contraseñas y demás datos en ownCloud
- Cliente de sincronización genérica de archivos. (Parte de este proyecto).
- Aplicación que sincronice los datos de configuración de una aplicación de escritorio.
- Aplicación para utilizar ownCloud como un servidor de Diáspora
- Editor de textos web y plugin para LibreOffice para sincronizarse entre ambos

Posteriormente al lanzamiento de ownCloud (Junio de 2010) Apple anunció iCloud, el cual estará disponible a finales de 2011 para los usuarios finales, y desde Junio de 2011 para desarrolladores.

El concepto detrás de iCloud es el misma que el de ownCloud, salvo que iCloud será un servicio privativo centrado en Apple y su ecosistema.

Figura 3.6: Esquema conceptual de iCloud

3.2.2. Gestión de información personal

3.2.2.1. Gestores de información personal como aplicaciones aisladas.

Estos programas fueron las primeras aproximaciones para informatizar el tratamiento de información personal. Inicialmente, y en la mayoría de los casos, no son muy útiles, ya que requieren que el usuario esté en contacto con la máquina, normalmente relativamente grande y pesada, para acceder a esos datos y modificarlos, careciendo de la versatilidad y portabilidad de los clásicos blocs o agendas.

Sin embargo, su utilidad se multiplica si se necesitan manejar cantidades ingentes de información, que no podrían almacenarse eficientemente en papel, así como su búsqueda y consulta sería muy costosa si no estuviese informatizada.

Un pequeño rebrote de este modelo ha surgido a raíz del auge de los dispositivos móviles. Los dispositivos móviles, incluso los más primitivos, cuentan con la potencia necesaria y el interfaz adecuado para poder gestionar de forma práctica esta información, paliando el defecto de la portabilidad. Sin embargo, la información sigue residiendo sólo en el dispositivo, otorgando muy pocas ventajas con respecto a un bloc de papel si la información

almacenada es pequeña.

Una característica surgida en estos programas a raíz de esa desventaja es la capacidad de exportar la información en archivos más o menos estándar, para importarlos en otras. Sin embargo, conservar la coherencia entre dos aplicaciones es un proceso tedioso y pesado, ya que al no tener ningún tipo de conectividad ha de hacerse manualmente.

Figura 3.7: Newton de Apple. Una de las primeras computadoras de mano avanzadas

3.2.2.2. Aplicaciones de escritorio sincronizadas

El ejemplo más representativo de este tipo de aplicaciones son los gestores de correo electrónico para escritorio. Éstos sincronizan datos personales, normalmente correos electrónicos, con un servidor, sin embargo acceder a esos correos electrónicos de manera ubicua utilizando únicamente un navegador web no es posible. Cabe clarificar esto último, ya que la mayoría de los servidores de correo electrónico comerciales modernos incluyen un cliente web, de esta manera el usuario puede utilizar su cliente de escritorio y además tiene disponible el interfaz web para poder utilizarlo en cualquier lugar. También cabe resaltar que el correo electrónico no es una solución completa de gestión de información personal.

3.2.2.3. Aplicaciones web

Son aquellos gestores que sólo disponen de un interfaz web para visualiazar y editar los datos, no siendo posible su sincronización con aplicaciones de terceros o de escritorio. La principal ventaja de este tipo de aplicaciones es su ubicuidad, el usuario sólo necesita un navegador web para utilizar la aplicación. Sin embargo, esto significa que el usuario sólo tiene acceso a estos datos cuando está conectado a internet. Las aplicaciones más comunes que siguen este modelo son los groupwares.

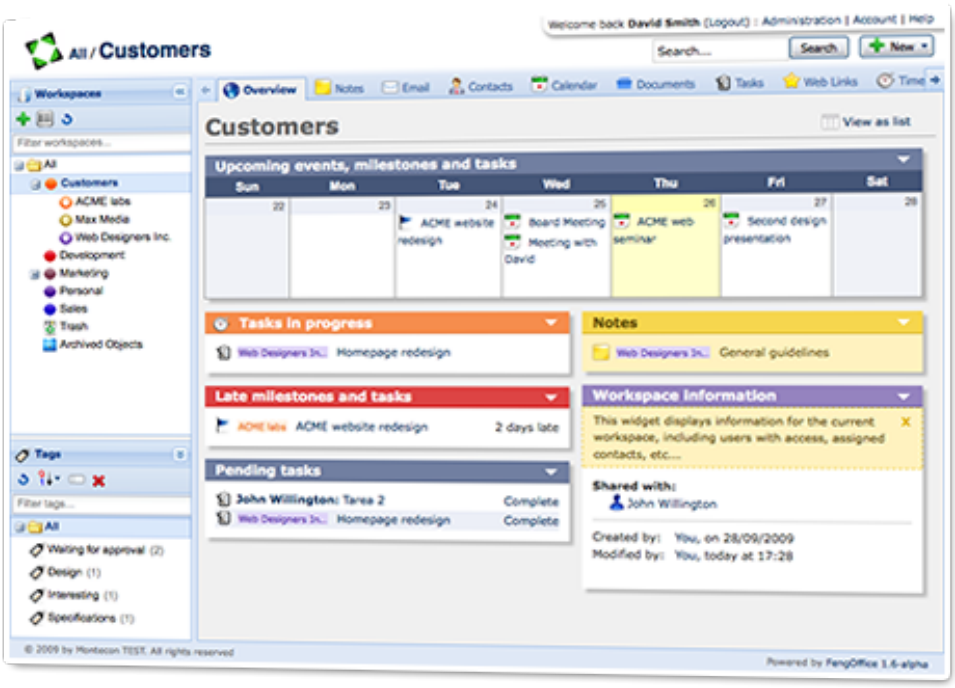

Figura 3.8: Interfaz web de fengOffice

3.2.2.4. Aplicaciones web sincronizables con clientes de escritorio

Este tipo de aplicaciones son aquellas que además de tener un interfaz web para que la información que manejan sea editable desde cualquier lugar, también ofrezca la posibilidad de sincroizarse con clientes de escritorio. Dentro de este conjunto podemos hacer varias distinciones

Aplicaciones web sincronizables con clientes de escritorio de manera completamente cerrada : Aplicaciones que sincronizan información personal entre distintos clientes y la nube, pero no permite de ninguna manera el uso de clientes de terceros. Este es el caso de mobileMe de Apple, que

únicamente permite el acceso y modificación de cierta información personal como los contactos a través de un número muy limitado de clientes (iPhone, AdressBook, iCal, Microsoft Outlook)

Aplicaciones web sincronizables con clientes de escritorio de terceros mediante un API : Son aquellas aplicaciones de gestión de información personal, que permite la sinocronización con aplicaciones de terceros a través de un API cerrada. Es el caso de Google Contacts, que permite la sincronización con éste a través de un API.

Aplicaciones web sincronizables con clientes de escritorio mediante estándares : Son el conjunto de aplicaciones que permiten la sincronización de información personal a través de un estándar, por lo que cualquier cliente que soporte el estándar podrá acceder y modificar dicha información. Actualmente existen tres protocolos estándar utilizados para la sincronización de información personal

- calDav: calDav es una extensión de webDav para sincronizar calendarios con un servidor remoto. El primer borrador de su especificación data del 2003. Finalmente fue estandarizado y especificado por el Internet Engineering Task Force en el documento RFC 4791 en Marzo de 2007.

  Utiliza archivos en formato iCalendar para los datos. Esto permite sincronizar de manera sencilla y estándar todos los datos contenidos en archivos de tipo iCalendar, lo que implica sincronizar, entre otros, tareas, calendario y notas.

  calDav se está convirtiendo en un estándar ampliamente utilizado y soportado.

  Múltiples aplicaciones web también soportan la funcionalidad de ser un servidor calDav.

  En aplicaciones de escritorio, calDAV es soportado por aplicaciones como iCal de Apple, o iPhone

- cardDav: cardDav es una extensión de webDav para sincronizar archivos vCard, de manera análoga a calDav. Sin embargo, su especificación aún es un borrador. cardDav utiliza ficheros vCard para sincronización de contactos.

  A pesar de que su especificación final aún no está disponible, el soporte a cardDav es cada vez mas amplio. Akonadi, el servidor de información personal de KDE, soporta cardDAV tanto como servidor como cliente.

- GroupDav: GroupDAV es un protocolo basado en webDAV que pretendía aunar las características de cardDAV y calDAV de manera mas

sencilla, utilizando también para el intercambio de información ficheros vCard e iCalendar. Sin embargo, no tuvo mucho éxito, y su desarrollo ha sido abandonado.

- SyncML: SyncML es un protocolo genérico para sincronización de información, esto es, sirve para sincronizar cualquier tipo de archivo. Sin embargo, su uso mas común es la sincronización de información personal.

  El principal problema de SyncML es que su especificación es bastante vaga en algunos aspectos, sin embargo esa característica es necesaria para utilizar el protocolo para sincronizar cualquier tipo de información. Por ejemplo, un cliente y un servidor han de utilizar el mismo tipo de archivo vCard e iCalendar, las mismas versiones y almacenarlo en el mismo directorio.

  A pesar de esto, SyncML es ampliamente utilizado. Probablemente, el servidor SyncML más conocido y utilizado sea Funambol, quien además tiene clientes para Android, iPhone, Windows, etc, lo que permite sincronizar la información personal entre todos esas plataformas. Sin embargo, el uso que hace el software oficial de Funambol de SyncML es únicamente la sincronización de información personal a través de iCalendar, vCard y además algunos contenidos multimedia.

  Existen también clientes de SyncML para Akonadi (el servidor de información personal de KDE), o varias series de teléfonos Nokia por ejemplo.

3.2.3. Clasificación de aplicaciones de sincronización genérica de archivos

3.2.3.1. Software de control de revisiones

Este tipo de software surgió para cubrir la necesidad de los programadores de trabajar en un proyecto común de manera eficiente.

Hasta entonces, la manera de escribir software de manera colaborativa era muy engorrosa. Requería mucha disciplina por parte de los programadores para etiquetar sus cambios, resultando en múltiples instancias de ficheros similares de las que cada programador tenía una copia.

Sin embargo, el engorro de manejar manualmente mútiples versiones de un fichero es mucho menor al que supone cruzar los cambios realizados por otros programadores. Para cubrir estas necesidades, nacen lo sistemas de control de revisiones. Permiten versionar ficheros, recuperar cambios previos y sincronizar nuevas modificaciones

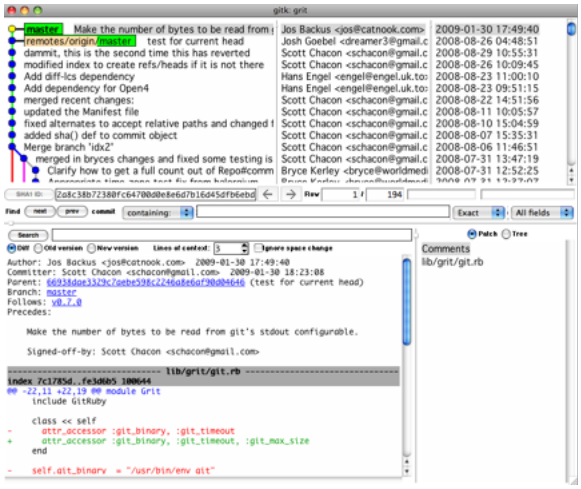

Figura 3.9: Interfaz gráfico de Git, uno de los sistemas de control de revisiones más utilizado

Aunque los sistemas de control de revisiones estén pensados para sincronizar código, pueden utilizarse de manera genérica para sincronizar cualquier tipo de información. Su gestión de los cambios en los ficheros de texto y la propagación de los mismos suele ser extremadamente eficiente, ya que utilizan *delta-encoding* para transmitir los cambios. Esto es, propagan sólamente los cambios entre versiones, no los ficheros enteros.

Sin embargo estos sistemas al estar ideados para el desarrollo de software, su desempeño con grandes archivos binarios no es óptimo.

3.2.3.2. Software de sincronización genérica de archivos

De manera muy similar a los sistemas de control de revisiones, este tipo de software se encarga de sincronizar cambios en un conjunto determinado de ficheros, detectar los cambios y propagarlos por medio de *delta-encoding*.Sin embargo, al estar diseñados para cualquier tipo de archivo, su desempeño con archivos genéricos de mayor tamaño es mejor que en el software de control de versiones.

Muchos de los ejemplos de este tipo de software además implementan una interfaz web para ejecutar operaciones básicas con este tipo de archivos (eliminaros, renombrararlos, descargarlos...)

El ejemplo más representativo de este tipo de software es de nuevo Dropbox. Una alternativa de software libre reseñable es SparkleShare. Sin embargo, SparkleShare utiliza git como backend para la gestión de ficheros, y por tanto adolece del problema de manejo de grandes ficheros binarios.

Algunos de los programas implementados con esta finalidad incorporan un API para permitir a programadores escribir herramientas que trabajen

con los ficheros de los usuarios.

En nuestro trabajo, la aproximación que tomamos es esta última. Como protocolo de transimisión de datos, utilizamos SyncML, pero todo el resto de la funcionalidad ha sido implementada por nosotros. Es decir, SyncML nos indica cómo transmitir la información, pero no dice nada acerca de cómo gestionarla. (Versionamiento, delta-encoding, etc)

Aun así, nuestro servidor tendrá una ventaja con respecto a los modelos comentados anteriormente. Nuestro servidor SyncML funcionará directamente con los clientes SyncML destinados a sincronizar información personal.

En el próximo capítulo...

En el próximo capítulo, abordaremos de manera detallada el proceso de implementación de las aplicaciones descritas: El servidor SyncML, la aplicación web para ver y editar contactos y el cliente de escritorio para sincronizar información genérica

Capítulo 4

Desarrollo del proyecto

The competent programmer is fully aware of the strictly limited size of his own skull; therefore he approaches the programming task in full humility, and among other things he avoids clever tricks like the plague

W.Dijkstra (1972)

Resumen: Durante este capítulo se detallará el desarrollo de las aplicaciones objeto de este proyecto. Primero se describirán las herramientas utilizadas en la implementación del mismo, y luego se esquematizarán y se explicarán las aplicaciones desarrolladas.

4.1. Herramientas utilizadas

Primero se hablará de KDE, la comunidad de software libre en cuyo marco se ha desarrollado el proyecto. Luego se procederá a dar una explicación de la plataforma donde el proyecto se ha implementado, ownCloud. Más tarde se explicarán en relativa profundidad las tecnologías utilizadas para la implementación del proyecto: El protocolo de sincronización SyncML, el formato de tarjetas electrónicas vCard y el formato de calendarios electrónicos iCalendar.

4.1.1. KDE

Figura 4.1: Logo de KDE

KDE se autodefine como *KDE is an international team co-operating on development and distribution of Free, Open Source Software for desktop and portable computing. Our community has developed a wide variety of applications for communication, work, education and entertainment. We have a strong focus on finding innovative solutions to old and new problems, creating a vibrant, open atmosphere for experimentation* KDE (2011a).

Fue fundada el 1996 por Matthias Ettrich, entonces estudiante, con el fin de mejorar ciertos aspectos del escritorio de Unix. La principal deficiencia que pretendía solventar es la falta de un *look and feel* congruente entre ellas.

Su primer trabajo fue la creación de un conjunto congruente de aplicaciones y un entorno de escritorio, en cual los usuarios encontrasen una experiencia unificada. Además, pretendía que el entorno de escritorio fuese usable.

La primera versión estable de KDE fue lanzada en Julio de 1998 KDE (2011b).

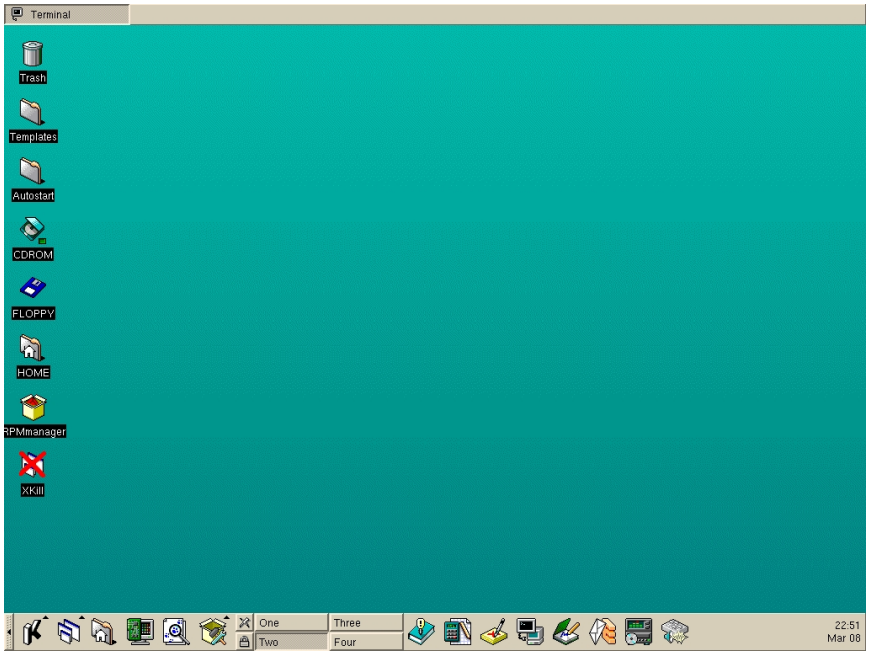

Figura 4.2: KDE 1.0

Hoy en día, KDE representa una de las mayores y más influyentes comunidades de software libre. También cuenta con una de las mayores bases de usuarios, fruto de su objetivo de hacer un entorno de escritorio usable, y de colaboraciones con universidades, instituciones gubernamentales y empresas.

Por ejemplo, es el único entorno de escritorio utilizado en las escuelas de Brasil, donde es usado por más de 52 millones de estudiantes.

La última versión mayor de KDE, KDE 4, fue lanzada en el año 2008 KDE (2009)

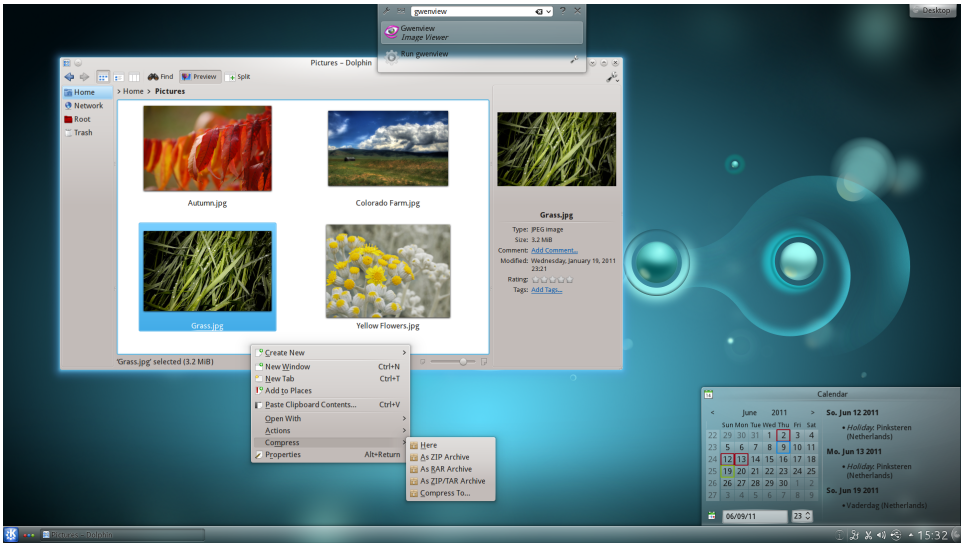

Figura 4.3: KDE SC 4

Esta nueva versión fue reescrita completamente, y su arquitectura fue rediseñada desde cero.

En Noviembre de 2009 (KDE (2011c)), KDE decide reposicionar su marca. Los cambios que adoptan quedan resumidos en *KDE is no longer software created by people, but people who create software*. Esto implica que ahora a partir de ese momento la palabra KDE no se refiere de un conjunto de software, sino de la comunidad que hay detrás de él.

En este contexto nace Social Desktop en KDE. Social Desktop es un proyecto destinado a integrar comunidades e interacción social en el escritorio.

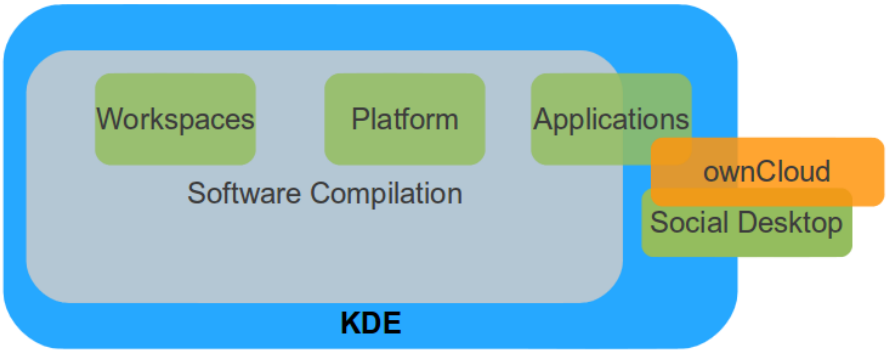

Figura 4.4: Esquema de organización de la marca KDE y del papel desempeñado por ownCloud y Social Desktop en ella

ownCloud, cuya primera versión es lanzada por primera vez en Junio de 2010 es desarrollada principalmente como un proyecto dentro de Social Desktop.

Sin embargo, en estos momentos la mayoría de sus desarrolladores no tienen relación con Social Desktop, y la relación con KDE es debida a razones históricas y al soporte que recibe el proyecto de la comunidad de KDE.

4.1.2. ownCloud

Figura 4.5: Logo de ownCloud

ownCloud es una plataforma de auto-hosting que forma parte de (social desktop) KDE. Su objetivo es proporcionar acceso universal a los datos del usuario a través de un interfaz web o WebDAV. También deberá proveer de una plataforma para editar y sincronizar fácilmente los contactos, calendario y marcadores a través de todos los dispositivos, y habilitar un sistema de edición básico a través de la web. (ownCloud 2011)

Su instalación tiene requsitos mínimos para el servidor, y no necesita permisos especiales. Esto es, cualquier servidor LAMP estándar deberá ser

suficiente para una instalación de ownCloud. ownCloud deberá proveer de un robusto y simple API para la instalación de aplicaciones y plugins. Además ownCloud puede desplegarse en muchos servicios populares de cloud computing tales como Amazon S3.

La primera versión de owncloud fue lanzada en Junio de 2010. Incluía funcionalidades básicas para subir archivos y gestionarlos, acceso WebDAV a los ficheros y algunas opciones básicas de configuración.

La versión actual de ownCloud (1.2) incluye además un sistema de plugins, gestión de usuarios y diversas pequeñas mejoras.

En la futura version de ownCloud (2.0), se prevee entre otras cosas crear un cliente de sincronización, versionamiento automático de ficheros y sincronización de información personal, siendo algunas de estas características el objetivo de este proyecto.

ownCloud utiliza en su arquitectura el patrón modelo-vista-controlador, y se espera que los programadores utilicen también este modelo en la implementación de sus aplicaciones ya que La estructura del API está pensada para utilizar este modelo de manera natural.

El patrón aísla la lógica de negocio de la interfaz de usuario, permitiendo alto grado de independecia en el desarrollo, testeo y mantenimiento de ambas. (Reenskaug (1979))

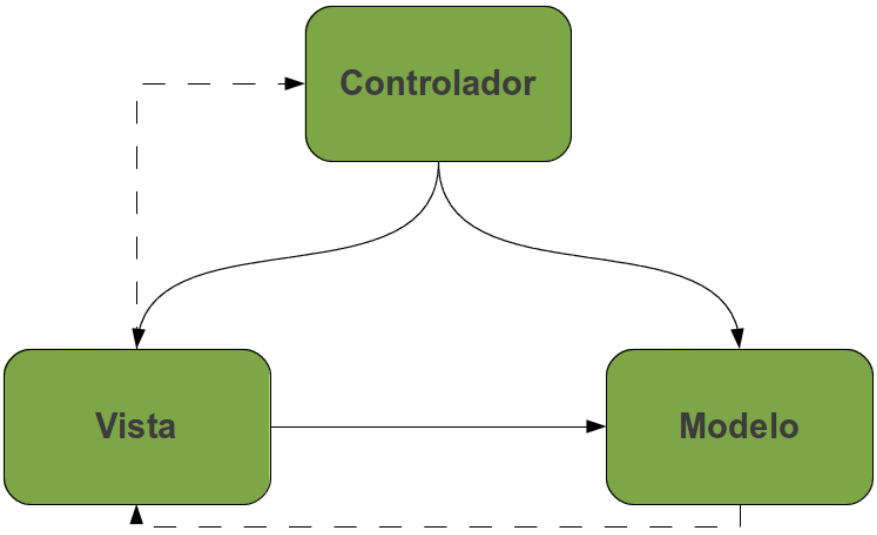

Figura 4.6: Diagrama simplificado del patrón de programación modelo-vista-controlador

- El modelo:

El modelo contiene los datos a ser mostrados y la lógica del negocio. Es independiente de la vista y del controlador. El aviso de que nuevos cambios en datos relevantes del modelo se hace a través del patrón de diseño *observador* normalmente a la vista, para que ésta gestione la nueva información.

- La vista:

  La vista es la responsable de presentar los datos requeridos al modelo, y la encargada de gestionar la interacción con el usuario.

- El controlador:

  El controlador recibe la entrada del usuario a través de la vista y prepara una respuesta haciendo consultas al modelo.

El patrón modelo-vista-controlador es un patrón ampliamente utilizado en aplicaciones web, donde la vista es el código HTML renderizado por la aplicación. El controlador recibe información a través de GET o POST y se prepara para procesarla, delegando en el modelo para esta operación, ya que es el que contiene la lógica de negocio. Más tarde, cuando la información ya ha sido procesada, se le envía a la vista que la muestra de una manera apropiada al usuario.

Este es el patrón de programación utilizado en la propia implementación de ownCloud y en las aplicaciones de este proyecto exceptuando el cliente de sincronización genérico.

4.1.3. SyncML

4.1.3.1. El protocolo SyncML

SyncML (Synchronization Markup Language) es el nombre de un protocolo estándar de intercambio de información.

El propósito de SyncML es ofrecer un estándar abierto para implementar soluciones de sincronización de datos, ya que actualmente, casi todas las soluciones que existen para este problema son específicas para cada aplicación.

La aplicación directa más común de SyncML es para el intercambio de información personal, es decir, contactos y calendario entre dos dispositivos tales como un ordenador y un teléfono móvil. Para esto, se hace uso de dos formatos estándar para la sincronización de los dos elementos más representativos de la información personal. vCard, para contactos e iCalendar, para calendarios y tareas. Sin embargo, el estándar está ideado para sincronizar cualquier tipo de información.

El protocolo está basado en XML, y puede ser transmitido, entre otros, mediante métodos POST de HTTP, lo que permite utlizarlo fácilmente con cualquier dispositivo conectado a internet.

SyncML ha sido desarrollado por la *Open Mobile Alliance* en el año 2001, en colaboración con empresas como Samsung, Motorola, Nokia e IBM (entre otras) y cuenta con un gran apoyo en el mercado, existiendo clientes para prácticamente todas las plataformas. Cabe resaltar, que la práctica totalidad de estos clientes tienen como fin la sincronización de información personal, y no el intercambio genérico de información.

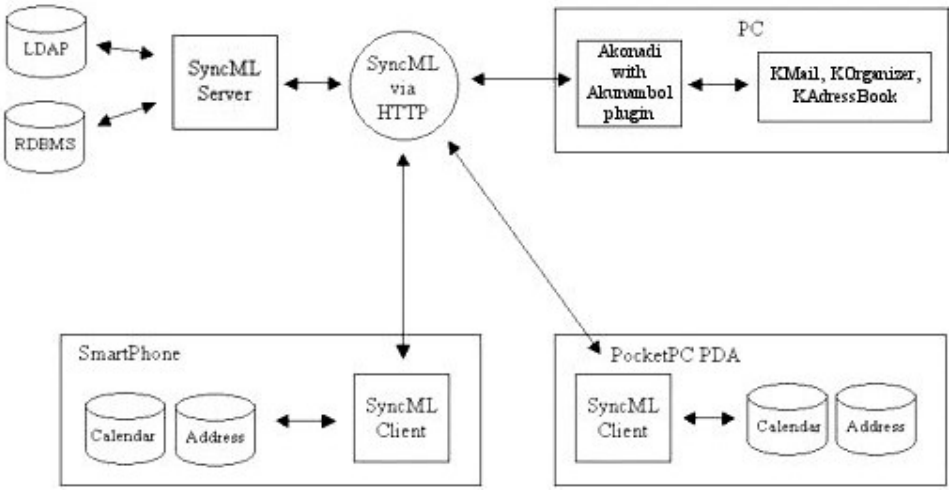

Figura 4.7: Esquema simplificado de la arquitectura de SyncML

Como podemos apreciar en la figura 4.7, la arquitectura de SyncML es una arquitectura típica de cliente-servidor, transmitida a través de un protocolo como HTTP (pueden ser otros).

Un posible inconveniente de la implementación de SyncML, es que su especificación es vaga en algunos aspectos. Esto implica, que por ejemplo, no existe ningún mecanismo de resolución de conflictos de sincronización especificado en el protocolo, y cada implementación podría hacerlo de una manera distinta.

El funcionamiento del protocolo, su representación y su transmisión a través de HTTP pueden consultarse en syn (2009)

4.1.3.2. PHPSyncML

Para la implementación del procolo SyncML en ownCloud, no hemos partido desde cero. Para empezar a trabajar se ha utilizado una implementación GPL muy rudimentaria de SyncML: PHPSyncML. PHP (2011) Esta elección ha sido determinada por varios aspectos:

- Implementación rudimentaria de SyncML: Algunas tareas tediosas pero sencillas ya implementadas (aunque a veces con errores) y nada más.

- GPL
- Implementado en PHP. Imprescindible, ya que de otra manera no podría ejecutarse como aplicación estándar en ownCloud.

Sin embargo, dado su temprano estado de desarrollo, muchas de las funcionalidades básicas presentaban errores de fondo, y corregirlos ha resultado en algunos casos más complicado de lo que hubiese costado reimplementar esas funciones. Esto es debido a que la única manera sencilla y directa de depurar este tipo de software es con un analizador de paquetes.

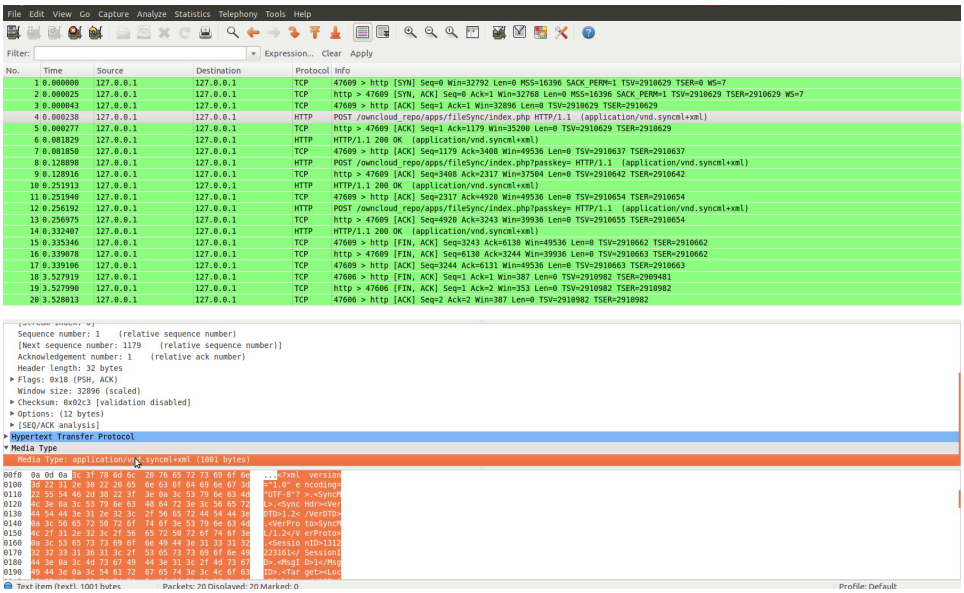

Figura 4.8: Utilizando el analizador de paquetes wireShark para depurar el servidor SyncML.

Cabe resaltar también el proceso de adaptación de la finalidad del servidor. PHPSyncML estaba escrito como un servidor SyncML diseñado únicamente para sincronizar información personal, y no archivos genéricos como era nuestro objetivo.

Además, el proyecto ha sido abandonado por sus autores, por lo que han declinado incorporar ninguna de las correcciones o mejoras realizadas al servidor.

4.1.3.3. Funambol

Para testear y probar el servidor SyncML desarrollado y las características únicas de éste (en concreto, la de sincronizar cualquier tipo de informa-

ción), se ha utilizado el SDK de Funambol para construir un pequeño cliente genérico.

Funambol es una compañía con un modelo de negocio de licencia dual. Esto incluye software comercial y software de sincronización open source basado en SyncML (Funambol project).

Funambol project consta de dos partes principales: Un servidor SyncML basado en Java (el cual en un primer momento se pensó en utilizar para dotar a ownCloud de esta característica, pero que posteriormente fue desechado debido a que una de las exigencias de desarrollo en ownCloud es que sea capaz de correr en un servidor con úncamente Apache, PHP y MySQL instalados) y un SDK (plataforma de desarrollo) con APIs disponibles en C++ y Java.

Como ya hemos mencionado en este proyecto, hemos utilizado el SDK de Funambol para desarrollar un pequeño cliente en C++ genérico que se sincronice con el servidor. Esto ha tenido un doble propósito, testear el servidor y crear una base desde la que partir para implementar un cliente de escritorio más sofisticado.

4.1.4. vCard e iCalendar

Como hemos mencionado ya anteriormente, el uso más común que se hace del protocolo SyncML es la sincronización de información personal. Para esto, se hace uso de dos formatos estándar para la sincronización de los dos elementos más representativos de la información personal. vCard, para contactos e iCalendar, para calendarios y tareas.

4.1.4.1. vCard

vCard es un formato estándar de archivo para tarjetas de negocio electrónicas, desarrollado por la *Internet Mail Consortium* en el año 1998 con el apoyo de empresas como Apple o IBM entre otras.

Los ficheros vCard contienen información tal como el nombre, el número de teléfono, la dirección de correo electrónico etc. de una persona.

vCard, permite, además, la adición de campos no estándar mediante un prefijo, para dotar al estándar de la flexibilidad necesaria para ser adoptado ampliamente.

Actualmente, vCard es el estándar de facto para la gestión de contactos, y es soportado por la práctica totalidad de las aplicaciones que de alguna manera gestionan contactos.

En vca (2011) se detalla la especificación del formato.

4.1.4.2. iCalendar

Con un objetivo similar a vCard, pero con el fin de gestionar información relativa al calendario, se desarrolló el estándar vCalendar. Además, su

estructura es similar.

iCalendar es la evolución del formato vCalendar, desarrollado y mantenido esta vez por el *Internet Engineering Task Force*.

iCalendar permite, entre otros, describir un evento que ocupa una determinada franja de tiempo dentro de un tenerminado día, gestión de tareas, diario, etc. Además, de manera similar a vCard, contempla la creación de extensiones para ampliar el estándar para dotarlo de flexibilidad ante las distintas necesidades de distintas aplicaciones.

En ica (2009) se detalla la especificación del formato.

4.2. Descripción de las aplicaciones desarrolladas

Podemos dividir las aplicaciones desarrolladas en dos partes principales; el servidor SyncML, y la parte que trabaja con los ficheros vcf sincronizados y que interactúa con el usuario para realizar todas las tareas de configuración. A estas dos partes, a partir de ahora las llamaremos Servidor SyncML y `contact viewer & editor`.

También describiremos brevemente el cliente de sincronización genérico desarrollado.

4.2.1. Contact viewer & editor

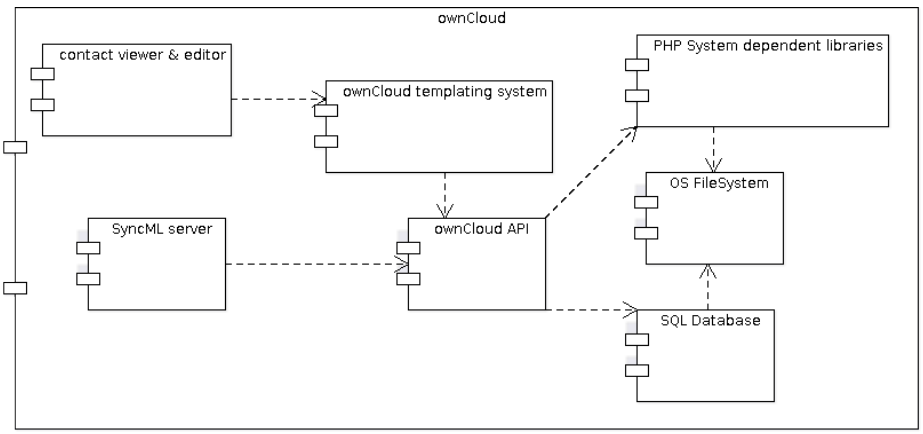

Figura 4.9: Diagrama de componentes de ownCloud y de las aplicaciones desarrolladas

La interfaz de esta aplicación es muy sencilla. y consta de dos partes, la primera es el editor de contactos, accesible desde un enlace con el nombre

de la aplicación, y una sección de configuración donde podemos editar la configuración del servidor SyncML.

Como se puede observar en la figura 4.9, aunque empaquetadas como una misma aplicación, el grado de independencia entre el servidor SyncML y `contact viewer & editor` es alto. El servidor SyncML mantiene una URL de escucha, y empieza a trabajar en cuanto recibe una petición de un cliente SyncML mientras que la interfaz sirve para configurar todos los parámetros del servidor y visualizar la información sincronizada, todas estas operaciones realizadas a través del API.

`contact viewer & editor` es una aplicación web programada mediante el patrón modelo-vista-controlador que hace uso del API para utilizar el sistema de templating de ownCloud. Este sistema de templating (modelo-vista-controlador) es muy similar al que se realizaría nativamente en PHP.

También, como podemos ver en la figura 4.9 `contact viewer & editor` hace uso del ownCloud de API para todas las operaciones que sean nativas del sistema de ficheros. Esto es así, porque ownCloud mantiene su propio sistema de ficheros. Esta abstracción del sistema de ficheros permite, entre otras cosas, independencia completa de los archivos de los distintos usuarios que hay dentro de ownCloud, versionamiento automático de estos ficheros, encriptación del sistema de ficheros de los usuarios, etc.

`contact viewer & editor` trabaja con un directorio del directorio del usuario que ha iniciado sesión en ownCloud. Este es el directorio que utiliza el servidor SyncML para guardar los contactos sincronizados, así que cada vez que accedamos al editor accederemos a una versión actualizada de éstos. Cabe recordar que los contactos no son más que ficheros vcf (vCard). Podremos editar éstos ficheros de manera amigable mediante la interfaz, y guardar los cambios. El servidor SyncML será el encargado de cuando reciba una petición de sincronización, detectar los cambios realizados a los contactos, y gestionarlos junto a la información proveniente del cliente.

Una vez se ha accedido a dicho enlace, nos muestra una lista de contactos, con información relevante para identificarlos.

Esa información relevante son datos almacenados del contacto, y se muestra el dato más prioritario que exista. El orden de prioridad es: Nombre del contacto, direcciones de correo electrónico y números de teléfono. El resto de información poco relevante tiene la misma prioridad, y se tomará para listar en el orden que aparezca en el fichero vCard.

Cuando el usuario pulsa sobre un contacto de la lista, se abre un editor del contacto. En este editor es visible toda la información del contacto, y se muestra de una manera editable. Podremos añadir, editar o eliminar información, y al pulsarse el botón *Done* toda la información será guardada y estará lista para ser sincronizada

4.2.2. Servidor SyncML

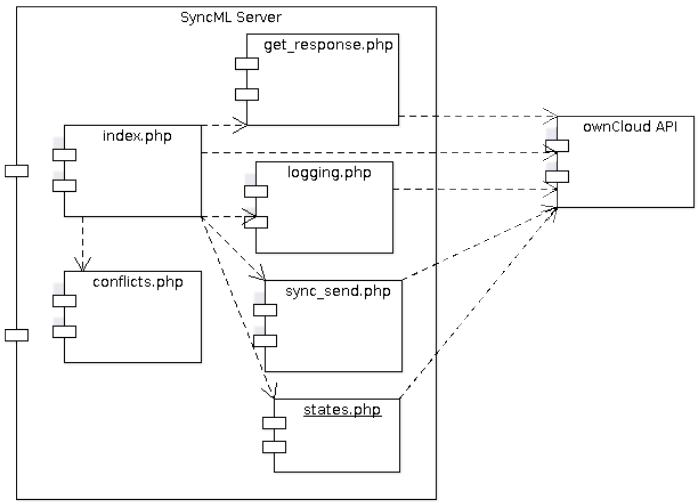

Figura 4.10: Diagrama de componentes del servidor SyncML implementado

A continuación se pasarán a describir los componentes del servidor SyncML

- Index.php Este script es el fichero principal y es el encargado de gestionar las peticiones por HTTP que se le hacen al sevidor, comprueba que las credenciales de inicio de sesión son válidas, parsea los ficheros, responde al cliente, etc. haciendo uso del API de ownCloud y del resto de ficheros del servidor que se detallarán más abajo. Para ello hace uso de llamadas a los ficheros indicados en la figura 4.10. Este fichero también carga la configuración del servidor de la base de datos de ownCloud.
- conflicts.php Este script es el único que no hace uso del API de ownCloud, ya que se encarga únicamente de la resolución de conflictos, y no necesita acceder a ficheros del sistema. En este fichero se implementan las políticas de sincronización y resolución de conflictos del estándar SyncML. (Server wins, client wins...)

- get_response.php Este script se encarga de interpretar, gestionar y producir la información referente a los estados de la transmisión.
- logging.php Este script se encarga de registrar las operaciones del servidor en la base de datos de ownCloud através del API en función de un grado de detalle determinado en la configuración del servidor
- sync_send.php Este script se encarga básicamente de generar el mensaje de sincronización que se intercambian cliente y servidor. En resumen, es el que se encarga de generar la información que pararece entre la etiqueta <Sync>del servidor en el estándar SyncML
- states.php Se encarga de parsear y procesar y actualizar la información contenida en los ficheros `.state` contenidos en las carpetas de los usuarios. Estos ficheros contienen el mapeo de relación entre los ficheros sincronizados (en el servidor y en cliente), y un hash de éstos para saber si han sido modificados.

La aplicación utiliza el API de ownCloud para almacenar, ficheros, autenticar las peticiones del servidor SyncML, guardar logs, guardar y cargar la configuración del sistema, etc.

De manera esquemática en el trabajo realizado sobre el servidor PHP-SyncML se han llevado a cabo las siguientes tareas:

- Sustituir todas las llamadas a funciones nativas de PHP para el tratamiento de ficheros y demás funciones a bajo nivel por llamadas al API de owncloud relativas a su sistema virtual de ficheros
- Utilizar el API de OC para guardar los logs en lugar de guardarlos en ficheros
- Utilizar el API de OC para la autenticación de peticiones de clientes SyncML
- Utilizar el API de OC para configurar automáticamente el servidor al instalar la aplicación, y guardar dicha configuración en la base de datos de OC mediante el API
- Utilizar el API de OC para leer la configuración cada vez que el servidor SyncML arranca
- Corregir la funcionalidad básica del servidor. La implementación del protocolo tenía múltiples errores de base, que tras depurarlos y detectarlos han sido corregidos.

4.2.3. Cliente de escritorio

Como ya hemos mencionado anteriormente, hemos utilizado el SDK de Funambol para crear un pequeño cliente de sincronización que nos permita testear nuestro servidor y además servir como base para crear un cliente más sofisticado y robusto.

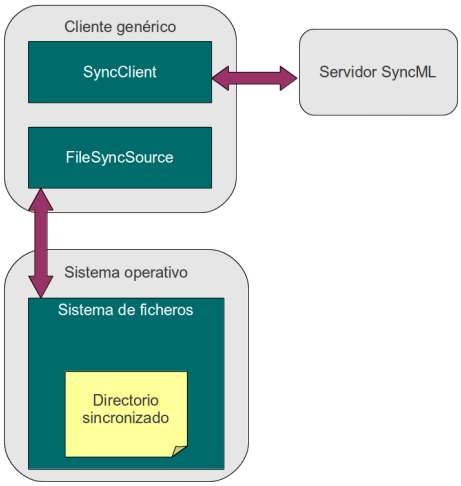

Figura 4.11: Esquema del cliente de escritorio desarrollado

Como puede apreciarse en la figura 4.11, `SyncClient` es el componente encargado de implementar el protocolo de sincronización, y de comunicarse con el servidor. La instancia básica de este componente ya viene implementada en el SDK, pero en el futuro será necesario modificarla para adaptarla a los cambios en la resolución de conflictos de sincronización entre cliente y servidor.

`FileSyncSource` hereda de la interfaz definida en el SDK como SyncSource, y es la clase utilizada por SyncClient para interactuar de forma transparente con las distintas maneras de almacenar la información. En este caso, utilizamos el sistema de ficheros del sistema operativo, e implementamos `FileSyncSource` con este fin.

En el próximo capítulo...

Durante el próximo capítulo describiremos de manera sencilla y visual los casos de uso de las aplicaciones. Su instalación, configuración, uso...

Capítulo 5

Casos de estudio

I mean, if 10 years from now, when you are doing something quick and dirty, you suddenly visualize that I am looking over your shoulders and say to yourself "Dijkstra would not have liked this", well, that would be enough immortality for me

W.Dijkstra (1995)

Resumen: Durante este capítulo se detallarán los casos de estudio de las aplicaciones desarrolladas en este trabajo

5.1. Instalación de la aplicación en ownCloud

Para instalar la aplicación, sólo hay que descomprimir la aplicación en el directorio `apps` de una instalación de ownCloud como podemos apreciar en la figura 5.1

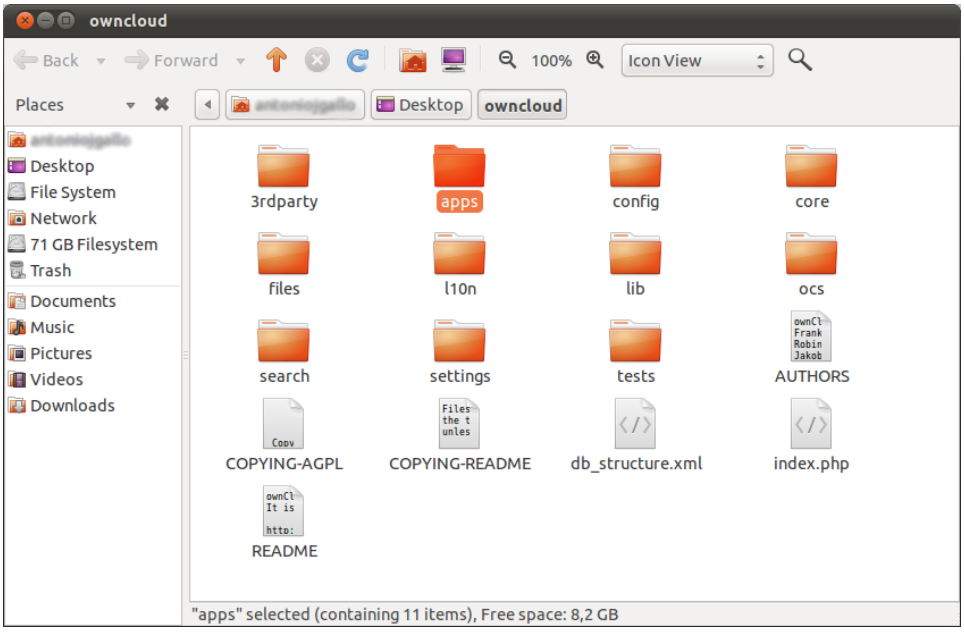

Figura 5.1: Directorio de ownCloud

Y dentro del directorio `apps`, descomprimimos la aplicación (La aplicación de ownCloud que hemos desarrollado se llama `fileSync`) como se representa en la figura 5.2.

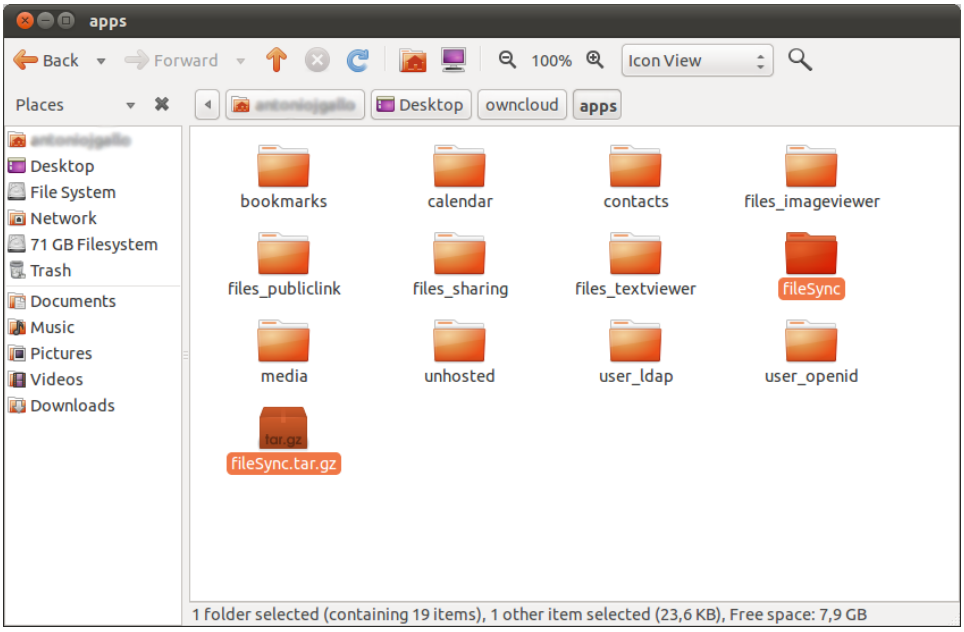

Figura 5.2: Directorio de aplicaciones de ownCloud

Ahora, vamos hasta la página de configuración de ownCloud y habilitamos la aplicación como muestra la figura 5.3. Ya está lista para usar.

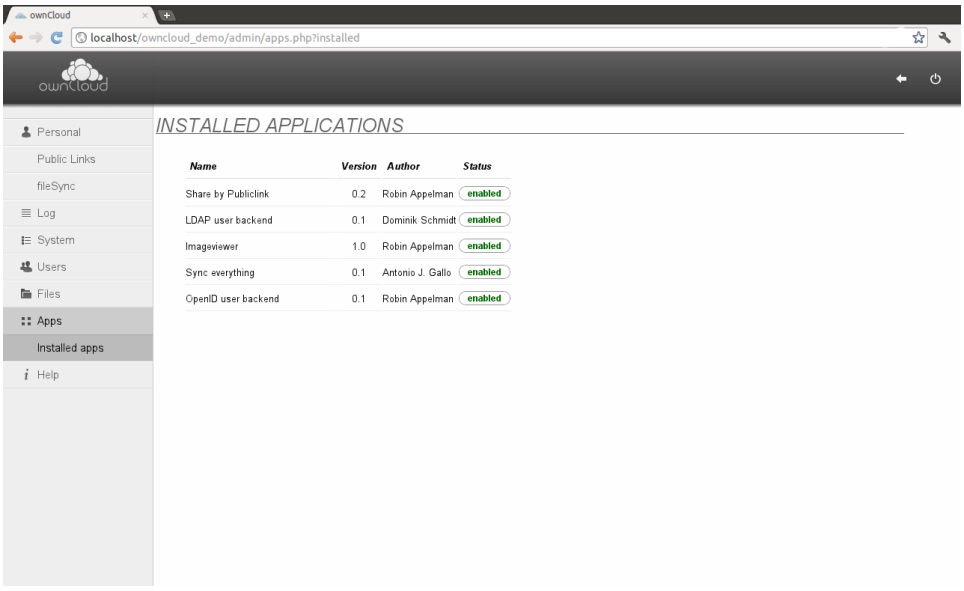

Figura 5.3: Habilitando la aplicación desde el interfaz de ownCloud

5.2. Sincronización de contactos

Para la sincronización de contactos, en este ejemplo se utilizará la harramienta Akonadi, y el plugin akunambol. Akonadi es un servidor de información personal de KDE. A grosso modo, Akonadi se encarga de implementar *conectores* para obtener distinta información personal de distintas fuentes, y las ofrece a los programas del usuario de manera unificada y sencilla a través de un API. También permite al usuario modificar esta información también mediante este API . Akunambol es uno de estos *conectores*, un plugin para Akonadi, que permite sincronizar parte de la información contenida en Akonadi (Tareas, Calendario y Contactos) con un servidor SyncML.

Abrimos Akunambol como muestra 5.4.

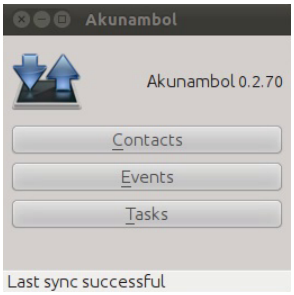

Figura 5.4: Interfaz de Akunambol

Ahora configuramos Akunambol. En `Account` ponemos en `Username` nuestro nombre de usuario de ownCloud (La aplicación de ownCloud usa las credenciales de usuario de ownCloud para autenticar) y en `password` nuestra contraseña de nuestro usuario de ownCloud.

En `Sync URL` la url de sincronización de la aplicación. Esta url estará compuesta por `dirección base de ownCloud` (por ejemplo `http://miDominio.com/owncloud`) concatenado con `/apps/fileSync/index.php`. Por ejemplo, una Sync URL válida podría ser `http://miDominio.com/owncloud/apps/fileSync/index.php`

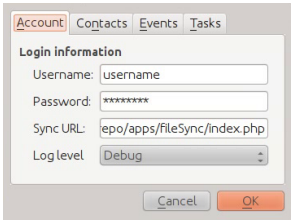

Figura 5.5: Interfaz de configuración de Akunambol

Pulsamos en lo que queramos sincronizar como muestra la figura 5.4 (Contactos, Eventos o Tareas) y ya tenemos sincronizado lo que hayamos querido sincronizar.

Resaltar que aunque sólo hay interfaz web para editar contactos, la sincronización funciona con toda suerte de información personal.

5.3. Visualización y sincronización de contactos

Al realizar la instalación de la aplicación, aparecerá un nuevo enlace en área de aplicaciones de ownCloud llamado `fileSync` (Figura 5.6). Este es el enlace de nuestra aplicación para ownCloud Pulsando en este enlace, veremos un listado de todos los contactos sincronizados con ownCloud.

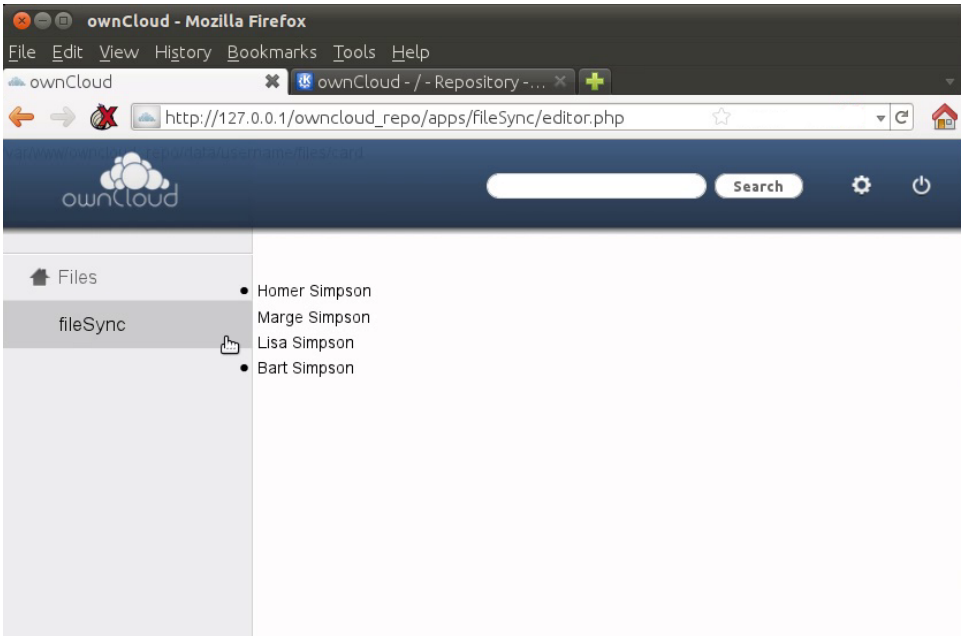

Figura 5.6: Interfaz web para la visualización y edición de contactos

Pulsando sobre alguno de estos enlaces de los contactos (Figura 5.6, accederemos a la información sobre dicho contacto 5.7 . Esta información será editable. Pulsando en `Done` para guardar los cambios realizados o en `Delete` para borrar el contacto, se ejecutará la acción procedente y volveremos al listado de contactos.

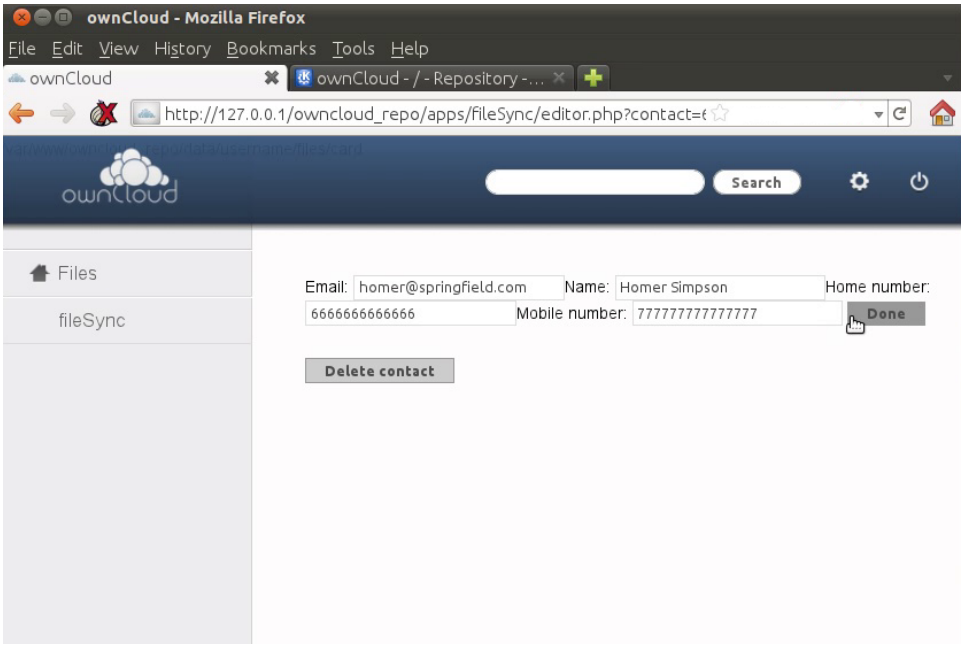

Figura 5.7: Editando un contacto

Cuando el cliente SyncML se resincronice con el servidor, los cambios realizados en los contactos se propagarán el cliente, así como si ha habido modificaciones en el cliente, éstas también se propagarán al servidor y se mostrarán en el interfaz web.

5.4. Sincronización de archivos genéricos

Nuestro cliente genérico SyncML no tiene interfaz, y debe ser lanzado desde un terminal. El nombre del programa es `gsync`

Los parámetros que debemos introducir en la llamada son los siguientes:

- `-d` para especificar el directorio a sincronizar con el servidor
- `-s` para especificar la dirección del servidor (de la misma forma que hacíamos con el cliente akunambol en el caso de uso anterior)
- `-u` para especificar el usuario de ownCloud
- `-p` para especificar la contraseña del usuario elegido de ownCloud.

Una llamada correcta sería por ejemplo

```
./gsync -d ./test -s http://miDominio.com/owncloud/apps/fileSync/index.php
```

`-u usuario -p contraseña` Al ejecutar la llamada con los parámetros correctamente introducidos, el directorio elegido se sincronizará con el servidor. ownCloud incorpora una explorador de ficheros, y éstos ficheros sincronizados se podrán ver en dicho explorador.

En el próximo capítulo...

Durante el próximo capítulo se explicarán las conclusiones obtenidas del desarrollo del proyecto, y los posibles trabajos futuros realizables partiendo de éste proyecto

Capítulo 6

Conclusiones y trabajo futuro

Now, if someone tries to monopolize the Web, for example pushes proprietary variations on network protocols, then that would make me unhappy.

Tim Berners-Lee en Herbst (1994)

Resumen: Durante este capítulo se hablará de las posibles ampliaciones a este proyecto, y sobre las conclusiones obtenidas después de la realización del mismo.

6.1. Conclusiones

La mayor parte de las retribuciones positivas de este proyecto provienen de haber trabajado en un proyecto de software libre de forma coordinada con el resto de desarrolladores.

Muchas de las decisiones tomadas durante el desarrollo de este proyecto han tenido que ser consensuadas con el resto de desarrolladores, lo cual en un principio aparentemente puede limitar la creatividad. Pero las ventajas de contar con las opiniones del resto de desarrolladores y su experiencia ha evitado más de una vez que durante la ejecución de este proyecto, se tuviese que desechar mucho trabajo hecho por no haber preveído lo que algunos desarrolladores.

Cabe resaltar que la experiencia de trabajar con este equipo pequeño-mediano de desarrolladores ha sido muy gratificante y muy instructiva, ya que durante el estudio de la carrera sólo había trabajado en grupos muy pequeños.

Aunque las aplicaciones desarrolladas durante este proyecto no poseen una gran complejidad conceptual, el haberlas desarrollado consensuadamente

con el resto de los desarrolladores implica que el código generado en este trabajo será distribuido en las siguientes versiones de ownCloud. El que en un proyecto de fin de carrera sea utilizado en un escenario práctico, real y potencialmente por miles de personas es algo muy poco común, y desde luego inmensamente satisfactorio.

En mi humilde opinión, el que los alumnos trabajasen en proyectos de software libre debería ser algo mucho mas usual de lo que es ahora mismo, así como la utilización de este tipo de software durante el estudio de la carrera. Opino que debido a la transparencia durante el desarrollo de aplicaciones, la total disponibilidad del código y al carácter colaborativo de este tipo de proyectos les hacen idóneos para ser estudiados y utilizados en la docencia.

6.2. Trabajo futuro

Con respecto a las aplicaciones realizadas, aunque funcionales, su extensión y mejora es bastante directa y evidente. A continuación iremos repasando las aplicaciones desarrolladas durante el proyecto, y las posibles líneas que pueden seguirse para continuarlas o expandirlas.

Aparte de las posibles mejoras listadas de las aplicaciones de este proyecto, son pertinentes las operaciones de mantenimiento del software típicas en cualquier proyecto de cierta envergadura. Corrección de bugs no detectados, adaptación de las aplicaciones a las futuras versiones de ownCloud, etc.

6.2.1. Servidor SyncML

El servidor SyncML ha sido implementado como una aplicación de ownCloud. Esto implica que se ha utilizado el API de ownCloud. Sin embargo, la versión de ownCloud utilizada durante el desarrollo no es una versión estable, y existe la posibilidad de que parte del API cambie hasta la versión final.

También, otro objetivo muy relacionado al desarrollado por este trabajo es la sincronización de contactos, calendario y tareas mediante CalDAV y CardDAV. Una pequeña adaptación del código de la aplicación sería muy interesante ya que la finalidad de CalDAV y CardDAV es la sincronización de información personal, y que tanto CalDAV y CardDAV utilizan los mismo ficheros estándar que los que utilizan los clientes SyncML y nuestro servidor (iCalendar para calendario y tareas y vCard para contactos). Esta modificación es tan sencilla como modificar los directorios utilizados para compartirlos con los de CalDAV y CardDav. mal escrito

Este pequeño cambio supondría que podremos sincronizar nuestra información personal, indistintamente mediante SyncML o CalDAV y CardDAV, manteniendo la congruencia de la información y aprovechando aplicaciones como el visor y editor de contactos (ya que trabaja con ficheros vCard)

6.2.2. Visor y editor de contactos

El visor y editor de contactos programado durante este proyecto con el fin de visualizar y editar los contactos sincronizados con SyncML dentro de ownCloud es sencillo, pero perfectamente funcional. Sin embargo, mejoras a la interfaz utilizando tecnologías web asíncronas soportadas por ownCloud (AJAX) son posibles y mejorarían en gran parte la experiencia del usuario.

También, tal y como se ha desarrollado el visor para contactos, es pertinente el desarrollo de editores similares el resto de información personal sincronizada: tareas y calendario.

6.2.3. Cliente de escritorio

El cliente de escritorio es la aplicación en la que más mejoras pueden implementarse. Durante este proyecto, se ha implementado un pequeño cliente que es capaz de sincronizar con el servidor cualquier archivo genérico, dividiéndolo en varios fragmentos y enviándolos a través de SyncML. Las mejoras posibles son, entre otras:

- Creación de una interfaz. Ahora mismo el programa se utiliza mediante línea de comandos
- Monitorizar determinado conjunto de directorios para detectar los cambios realizados automáticamente en esos directorios y propagarlos al servidor
- Mejorar la robustez de las transmisiones para soportar interrupciones prolongadas durante una transmisión, detección y corrección de corrupción de datos, encriptación de las transmisiones, etc

Parte II

Apéndices

Apéndice A

Autorización de difusión

Los abajo firmantes autorizan a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Antonio José Gallo Sánchez

Apéndice B

Lista de palabras

owncloud, kde, syncml, sincronización, cloud computing, pim, computación en la nube

Bibliografía

iCalendar specification. *The Internet Engineering Task Force*, Disponible en `http://tools.ietf.org/html/rfc5546`.

OMA Data Synchronization V1.2.2. *Open Mobile Alliance Ltd.*, Disponible en `http://www.openmobilealliance.org/Technical/release_program/ds_v1_2_2.aspx`.

Repositioning the KDE brand. Disponible en `http://community.kde.org/Promo/Branding/Rebranding_KDE_v1.1.0`.

About KDE. Disponible en `http://kde.org/community/whatiskde/`.

KDE Release 1.0. Disponible en `http://www.kde.org/announcements/announce-1.0.php`.

KDE Release 4.0. Disponible en `http://www.kde.org/announcements/4.0/index.php`.

ownCloud site. Disponible en `http://www.owncloud.org`.

PHPSyncML site. Disponible en `http://phpsyncml.sourceforge.net/`.

vCard specification. *Internet Mail Consortium*, Disponible en `http://www.imc.org/pdi/`.

Abelson y Hal. *Architects of the Information Society, Thirty-Five Years of the Laboratory for Computer Science at MIT*. MIT Press, 1999. ISBN 978-0262071963.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I. y Zaharia, M. Above the Clouds: A Berkeley View of Cloud Computing . *University of California at Berkeley*, 2009.

Farber, D. Oracle's Ellison nails cloud computing. *CBS Interactive*, Disponible en `http://news.cnet.com/8301-13953_3-10052188-80.html?part=rss&subj=news&tag=2547-1_3-0-5`.

FOSTER, I. y KESSELMAN, C. *The grid: blueprint for a new computing infrastructure*. Elsevier Inc., 1999. ISBN 1-55860-933-4.

HERBST, K. Interview with Tim Berners Lee. *Internet World*, Disponible en http://www.w3.org/People/Berners-Lee/FAQ.html.

HICKSON, I. Web Storage W3C Working Draft 01 September 2011. *The World Wide Web Consortium*, Disponible en http://www.w3.org/TR/webstorage/.

JOHNSON, B. Cloud computing is a trap, warns GNU founder Richard Stallman. *Guardian News and Media Limited*, Disponible en http://www.guardian.co.uk/technology/2008/sep/29/cloud.computing.richard.stallman.

MACASKILL, D. Amazon S3: Show me the money. Disponible en http://don.blogs.smugmug.com/2006/11/10/amazon-s3-show-me-the-money/.

MELL, P. y GRANCE, T. The NIST Definition of Cloud Computing. *National Institute of Standards and Technology*, 2009.

NEWTON, I. Letter to Robert Hooke. 1976.

REENSKAUG, T. The original MVC reports. *University of Oslo*, Disponible en http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html.

SOTOMAYOR, B., MONTERO, R. S., LLORENTE, I. M. y FOSTER, I. Virtual Infrastructure Management in Private and Hybrid Clouds. *IEEE Computer Society*, 2009.

W.DIJKSTRA, E. The Humble Programmer. 1972.

W.DIJKSTRA, E. Introducing a course on calculi. Disponible en http://www.cs.utexas.edu/users/EWD/ewd12xx/EWD1213.PDF.