Universitat Autònoma
de Barcelona

SISTEMA DE SINCRONIZACIÓN Y CONTROL DE VERSIONES DE DOCUMENTOS ON-LINE

Memoria del proyecto de Ingeniería Técnica en Informática de Sistemas.

Realizado por:

Javier Carreño Izquierdo

Y dirigido por:

Xavier Verge

El sotasignat, **Xavier Verge**

professor de l'Escola Universitària d'Informàtica de la

UAB,

CERTIFICA:

Que el treball al que correspon la present

memòria ha estat realitzat sota la seva

direcció per en **Javier Carreño Izquierdo**

I per a que consti firma la present.

Sabadell, Juliol de 2011

Presentación

Proyecto:

Sky Net: sistema de sincronización y control de versiones de documentos on-line

Autor:

Javier Carreño Izquierdo

Director:

Xavier Verge

Departamento:

Economía de la Empresa

Resumen de la memoria:

Esta memoria describe el proceso de desarrollo de “Sky Net”, un sistema de control de versiones y sincronización de documentos *On-Line*

La función principal de la aplicación será la de ofrecer al usuario un conjunto de herramientas capaces de establecer una comunicación cliente-servidor de forma transparente utilizando carpetas del disco duro. De ésta forma el usuario final podrá tener todos sus documentos guardados en un servidor (que puede ser o no conocido), poderlos recuperar en el momento que se desee, y tener los mismos documentos en todas las estaciones de trabajo que estén conectadas a la aplicación. Además se le añade una función de control de versiones que permitirá guardar el contenido de una carpeta bajo un identificador (nombre o versión), y poderlo recuperar más adelante aunque ya hayamos hecho modificaciones en los documentos de dicha carpeta.

Para lograr este objetivo, el proyecto se compone de dos aplicaciones: Sky Catcher, la aplicación cliente, y Sky Node, la aplicación servidor. La aplicación servidor estará instalada y configurada en una máquina servidora, convenientemente con alta capacidad de disco duro, mientras que la aplicación cliente será instalada en tantas máquinas como desee el usuario final.

Material entregado: Memoria en formato papel, dispositivo digital (CD-R) que incluye el código fuente de la aplicación y la memoria en formato digital.

ÍNDICE

CAPÍTULO 1

INTRODUCCIÓN

1. Introducción

1.1 Objetivo del proyecto

El proyecto se realiza bajo el objetivo de crear una aplicación de uso muy intuitivo que sea capaz de ofrecer un servicio de sincronización y control de versiones sobre los documentos que tendremos en ciertas carpetas definidas por el programa.

Lo que se pretende conseguir con este proyecto es que el usuario final sea capaz de olvidarse completamente de tener que llevarse los documentos importantes de un lugar a otro de forma física (utilizando un lápiz USB, o discos duros portátiles, etc.), y sea capaz de tener una sincronización perfecta desde cualquier estación de trabajo. Esto será posible siempre y cuando haya conexión a internet, y bajo unas condiciones concretas.

Para llevar esto a cabo se utilizará un esquema de cliente-servidor en la aplicación desarrollada. La aplicación cliente (llamada Sky Catcher), se encargará de conectarse automáticamente a un servidor que habremos configurado en el primer inicio, y se comunicará con él para detectar los cambios de archivos en la carpeta definida. Dichos cambios causarán que cliente y servidor se intercambien datos, y así el servidor siempre tendrá una copia exacta de dicha carpeta en su disco duro.

El servidor por su parte, se encargará de dar este servicio a más de un cliente a la vez, bajo el mismo o diferente usuario. Esto quiere decir que si hay más de un cliente conectado bajo el mismo usuario, los cambios que haga uno de ellos sobre un documento se enviarán a los demás clientes, y así los documentos estarán sincronizados siempre. Además el servidor tendrá un servicio de control de versiones básico capaz de guardar el estado y las modificaciones de una carpeta en un momento determinado, y el cliente podrá consultar (no modificar) los documentos en ese momento.

El control de usuarios se realizará utilizando el siguiente esquema de grupo-usuario:

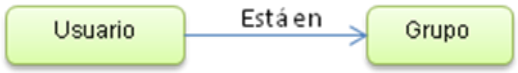


Fig 1. Relación Usuario-Grupo

Así pues, nos podremos encontrar usuarios sin grupo, usuarios con grupo, e incluso grupos sin usuarios. Más adelante explicaremos la utilidad de estas tres variantes, y cómo afecta a la sincronización de documentos.

1.2 Descripción del proyecto

El proyecto consta de tres partes muy diferenciadas entre sí:

- Diseño e implementación del protocolo y librería de comunicaciones entre cliente / servidor
- Diseño e implementación de la aplicación cliente, así como de la pequeña base de datos para la gestión de conexiones a servidores de Sky Net.
- Diseño e implementación de la aplicación servidor, así como de la base de datos para la gestión de usuarios y grupos.

En el siguiente esquema podemos ver un ejemplo de cómo se comunicaría un cliente Sky Catcher con un servidor Sky Node, al detectar que el archivo “Nuevo.jpg” ha sido modificado.

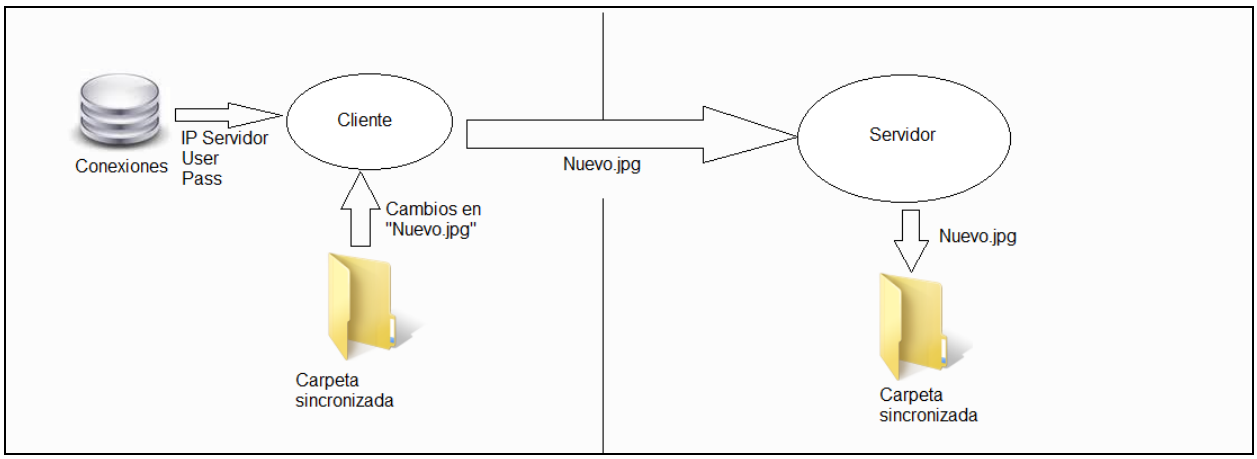


Fig 2. Comunicación básica cliente - servidor

La aplicación cliente detectará automáticamente un cambio en alguno de los archivos que están dentro de la carpeta sincronizada, y enviará una petición de actualización al servidor, enviándole el archivo. El servidor reconocerá al usuario, encontrará su carpeta sincronizada en su sistema de archivos, y cambiará el archivo antiguo por el nuevo.

Cabe destacar que esta detección de cambios en los archivos será dinámica para aliviar la carga de trabajo del servidor, y la carga de datos de la comunicación. Esto quiere decir que en vez de enviar el archivo por cada cambio que haya distinguiremos cuatro tipos de cambios en un archivo: Crear, Modificar, Renombrar y Eliminar. El comportamiento del cliente será diferente en estos cuatro casos, en especial en el Renombrar. El servidor no tiene por qué necesitar el archivo en caso de renombrar y eliminar, simplemente con una instrucción de control será suficiente. Sin embargo, en crear y modificar si será necesario.

Como podemos ver, las dos aplicaciones son muy diferentes entre sí, y necesitan de un diseño e implementación totalmente por separado, aún usando ambas el mismo protocolo de comunicaciones.

1.3 Motivaciones personales

La motivación principal que me ha llevado a realizar éste proyecto tan ambicioso es la falta de opciones que existen actualmente en el mercado con la sincronización de documentos on-line. Actualmente existen muchas aplicaciones que proporcionan un servicio muy similar al que propongo (Podemos nombrar varios, pero el que más destacaría en el momento de realizar ésta memoria se llama *Dropbox*).

Todos estos programas proporcionan una capacidad de poder sincronizar tus documentos en un servidor que funciona en internet. Pero todas estas aplicaciones carecen de algunas de las funcionalidades que voy a aplicar en mi proyecto, que son:

- Mono usuario. Muchos de estos programas únicamente permiten tener un usuario sincronizado a la vez, con una carpeta. Si cambias de usuario, las carpetas se mezclan, o simplemente, se borra el contenido.

- Mono servidor. La gran mayoría de estos programas son de pago. Y como son de pago, utilizan servidores propietarios. No puedes conectarte a otro servidor que no sea el suyo (aunque luego internamente sean varios), y estas obligado a ello.

- Seguridad. Estas aplicaciones ofrecen un servicio en sus propios servidores. Tú no eres capaz de saber qué uso se dan a tus documentos una vez almacenados en ellos. Por lo tanto, aunque por ley tus datos pueden estar protegidos (en según qué estado no existen leyes orgánicas de protecciones de datos), siempre podrán acceder a ellos.

Así pues, como podemos ver, las diferentes soluciones que encontramos en el mercado pueden ser efectivas, pero tienen estos pequeños fallos que el proyecto que se ha realizado no tienen. La aplicación que quiero ofrecer al mercado será multiusuario (varias cuentas, carpetas separadas), multiservidor (nos podemos conectar a tantos servidores como queramos), y la seguridad estará supeditada por el servidor al que nos conectemos, pudiendo ser desde un organismo público, hasta uno que instales en tu propio domicilio.

1.4 Estructura de la memoria

Esta memoria contiene toda la documentación, diseño, pasos realizados, y problemas encontrados y su resolución acerca del proyecto Sky Net, sistema de sincronización y control de versiones de documentos on-line.

La memoria se divide en siete capítulos, de los cuales los cuatro primeros tratan sobre la explicación del proyecto, sus funciones y la realización. Los siguientes capítulos tratan sobre los juegos de pruebas realizados sobre el proyecto para asegurar su correcto funcionamiento, los objetivos cumplidos, y la bibliografía utilizada como soporte durante la realización del proyecto.

En el primer capítulo encontraremos la introducción al proyecto, sobre qué trata, y las motivaciones personales que me han llevado a realizar ésta aplicación.

En el segundo capítulo encontraremos el estudio de viabilidad de nuestro proyecto, dónde analizamos la situación actual de mercado, las alternativas a nuestra aplicación, si podemos llevarlo a cabo, o no, y también los requisitos de la aplicación. En este apartado encontraremos la planificación temporal estimada al inicio, y el gasto económico.

El tercer y cuarto capítulo trata sobre la fase de diseño de la aplicación y de su implementación. En este capítulo podremos encontrar todos los esquemas y diagramas utilizados para determinar todas las funcionalidades de la aplicación, así como de los problemas encontrados en la fase de implementación, como se han resuelto, y qué tecnologías y librerías han hecho falta para poder realizar la aplicación.

En el quinto capítulo podremos encontrar los juegos de pruebas realizados durante y después de la implementación del proyecto. En este apartado podremos encontrar diferentes tablas de resultados y el comportamiento de la aplicación bajo situaciones de estrés.

Por último, en los dos últimos capítulos encontraremos las conclusiones finales, los objetivos que hemos cumplido y los que no, qué ha sido necesario para que la aplicación funcione correctamente, toda la bibliografía que he utilizado como soporte técnico para poder llevar a cabo, y un pequeño agradecimiento a personas ajenas al proyecto pero que han aportado sugerencias y observaciones que han mejorado el producto final.

1.5 Metodología de desarrollo

Para el desarrollo de este proyecto se ha seleccionado una metodología de desarrollo evolutiva, estableciendo en la primera versión un prototipo básico del protocolo de comunicaciones y de la aplicación cliente-servidor, y posteriormente añadiendo funcionalidades a los tres módulos de la aplicación. Siguiendo la estructura del proyecto, la primera fase constará del desarrollo de los siguientes módulos:

- Protocolo de comunicaciones (llamado Sky Net Protocol)
- Aplicación cliente (llamada Sky Catcher)
- Aplicación servidor (llamada Sky Node)

La primera fase del proyecto es la de realizar un diseño y una implementación básica con todo lo necesario para establecer una comunicación entre cliente-servidor, capaz de reconocer los cambios en un documentos y comunicárselo al servidor con tal de almacenar el documento y mantener una sincronización. A partir de esta fase, se irán añadiendo funcionalidades, tales como reconocer archivos duplicados, renombrar archivos, permisos de grupos, etc., que explicaremos mas adelante con detalle. Así pues, las fases serán las siguientes:

1. Diseño e implementación inicial del protocolo de comunicaciones, con juego de pruebas.
2. Diseño e implementación inicial de la aplicación cliente, con juego de pruebas,
3. Diseño e implementación inicial de la aplicación servidor, con juego de pruebas.
4. Unificar los tres módulos en un conjunto, y realizar juego de pruebas.
5. Añadir funcionalidades más específicas a los tres módulos.: Control de usuarios y grupos, detección de errores, detección de duplicados de archivos, etc.

Así pues, el esquema utilizado para este desarrollo sería el siguiente:

Fig 3. Esquema de desarrollo del proyecto

CAPÍTULO 2

ESTUDIO DE VIABILIDAD

2. Estudio de viabilidad

2.1. Introducción

Este proyecto se ha realizado con la intención de aportar un nuevo punto de vista a las aplicaciones de sincronización de documentos en internet, añadiendo funcionalidades que pocos o ninguno de los softwares existentes aportaban en el momento de la realización del proyecto. La realización del proyecto se hará en formato de dos aplicaciones (cliente-servidor), y la descripción de las funcionalidades sería la siguiente:

Aportar un control de usuarios/grupos en un entorno de sincronización on-line, permitiendo a la aplicación cliente ser capaz de gestionar documentos en uno o varios servidores a la vez, y usando una o varias configuraciones distintas por cada servidor conectado. La aplicación será capaz de sincronizar todos los documentos con el servidor de forma transparente, de forma que dichos documentos se puedan crear, modificar, eliminar y ser recuperados en máquinas distintas a donde se instaló originalmente la aplicación. Además, permitirá la creación de grupos públicos donde los administradores podrán colocar documentos para ser visualizados, pero no editados.

Lo que se pretende con esta aplicación es mejorar la usabilidad y flexibilidad en los entornos de sincronización de archivos on-line, que en su gran mayoría dependen de grandes empresas y en ocasiones de servicios contratados de pago.

A día de hoy la gran mayoría de éstas aplicaciones tienen versiones gratuitas y de pago, y casi ninguna de las aplicaciones de pago ofrece un sistema de gestión de grupos / usuarios efectiva, ya que la idea principal y básica es la de que un usuario tenga los documentos sincronizados con un servidor, que a su vez es servidor propietario. Una empresa, o ciertos usuarios, no pueden permitirse el tener los documentos almacenados en servidores ajenos a su negocio, y se necesita una solución a este problema.

2.2. Tipología y palabras clave

Tipología: Desarrollo de una aplicación de sincronización de documentos On-Line multiusuario y multiservidor.

Palabras clave: Sincronización On-Line, Gestión de grupos y usuarios, control de versiones.

2.3. Descripción

Mediante esta aplicación de sincronización de documentos On-Line, cualquier usuario que esté dado de alta en un servidor será capaz de tener sus documentos guardados de forma segura y manteniendo los cambios en cualquier lugar, pudiendo recuperarlos en cualquier momento siempre y cuando se tenga conectividad con el servidor. Además el usuario podrá visualizar los documentos publicados en los grupos públicos sin necesidad de estar registrado en el servidor, y podrá realizar versiones de sus carpetas para más adelante poder visualizar los documentos en la fecha que se realizó la versión.

2.4. Objetivos

- O1. Sincronizar los documentos de la máquina cliente con la máquina servidora de forma transparente al usuario final.
- O2. Permitir la creación y gestión de grupos públicos, así como la asignación de usuarios a grupos.
- O3. Permitir la creación de versiones y la visualización de ellas al margen de las carpetas sincronizadas.
- O4. Realizar una interfaz de usuario amigable para el cliente, permitiendo que el usuario más inexperto sea capaz de conectarse a un servidor de Sky Net y sincronizar carpetas.

	Crítico	Prioritario	Secundario
O1	**X**		
O2		X	
O3			X
O4		X	

Tabla 1. Criticidad de los objetivos

2.5. Definiciones, acrónimos y abreviaciones.

Sincronización On-Line: La sincronización on-line es el hecho de que un documento, o archivo que se encuentra en la máquina cliente esté duplicado en tiempo real en una máquina servidora, permitiendo su recuperación en caso de eliminación, o poderlo visualizar desde cualquier otra estación de trabajo.

Control de Versiones (CVS): El control de versiones permite versionar un documento (ya sea un archivo, o carpeta) de forma que guarda su estado en el momento que se realiza la versión. Un control de versiones sencillo permite visualizar esa versión, y los más complejos permiten re-editar esa información y guardarla como si de otra versión se tratara.

Protocolo de comunicación: Conjunto de instrucciones definidas que permiten a dos extremos (en nuestro caso, cliente y servidor) comunicarse e intercambiar información, ya sean instrucciones de control, o archivos enteros.

Aplicación cliente: Aplicación que se ejecuta en una estación de trabajo y que establece comunicación con un servidor, con el fin de intercambiarse información.

Aplicación servidor: Aplicación que se ejecuta en segundo plano en una estación servidora. Se encarga de dar un servicio a las aplicaciones cliente que se conectarán. Suelen ser aplicaciones pesadas y de mucha carga de procesamiento. También necesitan de una cantidad moderada de espacio en el disco duro.

2.6. Partes interesadas

Existen dos grupos bien diferenciados de usuarios que usarán esta aplicación.

Usuario básico - doméstico

El usuario básico será aquél que utilizará la aplicación para uso doméstico. Utilizará solamente la función de sincronización con el servidor, y creará usuarios que puedan utilizar él y el grupo social que le rodea. Por lo tanto, solamente utilizará la aplicación para tener sus documentos sincronizados y poderlos recuperar en caso de pérdida de documentos o compartirlos con otros usuarios / ordenadores.

Usuario avanzado – profesional

El usuario avanzado o profesional será aquél que utilizará la aplicación para hacer seguimiento de sus proyectos. Utilizará el servicio que le proporciona la aplicación para mantener un control de versiones sobre él y poder recuperar los documentos en caso de pérdida. Por lo tanto, hará uso del control de versiones y de la sincronización de archivos.

Usuario empresarial

El usuario empresarial es aquél que utilizará todas las funciones del programa. Utilizará la sincronización de ficheros para asegurarse su recuperación en caso de pérdida, para sincronizar grupos de trabajo con los mismos ficheros (varios trabajadores utilizando a la vez el mismo proyecto), y utilizará el control de versiones para mantener un seguimiento de los trabajos. A la vez podrá utilizar el programa para preservar sus documentos personales.

Usuario público / publicidad

Un usuario público, o un usuario empresarial que desee dar publicidad, pueden usar este sistema para proporcionar documentación o control de documentos utilizando la opción de grupos públicos. Estos grupos pueden mantener documentación e ignorar los cambios que realicen otros usuarios que los visualicen. Así estos documentos publicados estarán al alcance de todos, y siempre estarán actualizados en tiempo real. Un ejemplo seria un campus virtual de una universidad, donde cada grupo se puede considerar una asignatura, y todos los alumnos pueden acceder a su documentación.

2.6. Diseño / desarrollo de la aplicación

Los diferentes usuarios que llevarán a cabo el proyecto serán:

Javier Carreño Izquierdo, que se encargará del desarrollo y de la documentación del proyecto

Xavier Verge, tutor del proyecto, que se encargará de la revisión y coordinación del proyecto por parte de la universidad (UAB), así como de las pruebas de aceptación del software.

Nombre	Descripción	Responsabilidad
Xavier Verge	Tutor del proyecto UAB	Realizar el seguimiento del proyecto, marcar al alumno los plazos previstos para las distintas fases del proyecto, supervisión del proyecto, recomendar una metodología de trabajo, realizar las pruebas de aceptación.
Javier Carreño Izquierdo	Alumno que realiza el proyecto	Alumno que bajo el seguimiento de **Xavier Verge** se encargará de diseñar e implementar el proyecto, la corrección de errores, y la realización de la documentación.

Tabla 2. Usuarios de desarrollo del proyecto

2.7. Referencias

La bibliografía y el material utilizado para la realización de ésta práctica, así como la colaboración por parte de otras personas se encontrará en el capítulo X (Bibliografía y colaboraciones).

2.8. Producto y documentación del proyecto

- Se elaborará una memoria del proyecto que contendrá un estudio del objetivo y de la metodología utilizada para desarrollar el proyecto, con una explicación de las herramientas y de la documentación necesaria para realizarlo.

- Se elaborará un manual de instrucciones tanto para el usuario cliente como para el usuario administrador, explicando cómo utilizar la aplicación y qué posibilidades tiene.

2.10. Estudio de la situación actual

2.10.1. Contexto

Actualmente existen muchas aplicaciones que nos ofrecen un servicio similar al que propongo en mi proyecto, y en algunas ocasiones nos ofrecen un servicio *"Premium"* (de pago), que nos ofrecen nuevas funcionalidades, mas espacio de almacenamiento, etc.

Todas y cada una de estas aplicaciones están orientadas a un sector muy específico empresarial, o de usuario doméstico, que es aquél que quiere tener sus documentos almacenados en internet y acceder a ellos de forma automática y fácil. Sin embargo, en la gran mayoría de aplicaciones se les olvida que muchos de estos usuarios quieren utilizar más de una cuenta de usuario, o dentro de una misma empresa necesitan varias cuentas (por ejemplo, usuario por proyecto), y es aquí donde quieren hacer negocio: hacer pagar por cuenta.

La situación actual es que realmente existen pocas aplicaciones realmente versátiles en este campo. Normalmente nos encontramos con aplicaciones muy sencillas de utilizar, pero de alto coste, o aplicaciones totalmente gratuitas pero que necesitan un mantenimiento y una configuración muy laboriosos. Por lo tanto, las empresas normalmente optan por soluciones más sencillas, como serían las *VPN*[1], o acceso por control remoto a servidor. Estas dos opciones son prácticamente de coste nulo, pero si el servidor se queda sin conexión el sistema deja de funcionar.

[1] **VPN**: Virtual Private Network. Es una red virtual establecida entre un ordenador y un servidor a través de internet, y se puede trabajar en red como si de una red local se tratara.

2.10.2 Alternativas

Actualmente en el mercado tenemos varias posibles alternativas al proyecto que se quiere realizar, de entre las cuales destacaremos tres: Dropbox, Microsoft Live Mesh, y SubVersion.

- **Dropbox**: Aplicación desarrollada por la empresa *Dropbox*, que permite crear una cuenta de usuario gratuita hasta 2 Gb de capacidad, y sincroniza todo lo que contenga la carpeta "My Dropbox" creada en la carpeta de documentos del usuario. Se puede conseguir hasta 10 Gb de capacidad consiguiendo invitar a 5 personas más invitando desde tu cuenta de Dropbox, pero si se quiere más espacio se debe contratar los servicios Premium: 50 Gb por 9.90 $ al mes o 99$ al año, y 100 Gb por 19.99 $ al mes o 199 $ al año.

  Aún así, la aplicación es mono usuario, y si se desea añadir un usuario las carpetas internas se mezclarán, no tiene separación de diferentes cuentas de usuario.

- **Microsoft Live Mesh**: Aplicación desarrollada por Microsoft, y que forma parte del conjunto de herramientas "Live" que desarrolló en 2010. Es un software que permite tener las carpetas sincronizadas con la cuenta de correo Live, creada de forma gratuita, y la cual dispone de 5 Gb para sincronizar archivos on-line. Esta alternativa es más eficiente que Dropbox en cuanto a gestión de cuentas de usuario, ya que separa las cuentas en distintas carpetas (carpeta por cuenta), aunque se debe configurar esto manualmente. En el momento de la realización del proyecto no existían mejores planes de almacenamiento a excepción de los planes extra contratados para Windows Live Empresas, los cuales son muy caros y costosos.

  Se integra a la perfección con Windows Live Messenger (también para empresas), y con la cuenta de Microsoft Live, haciendo de ésta alternativa una alternativa cómoda y fácil de utilizar.

- **SubVersion:** Esta aplicación no es una aplicación de sincronización de archivos on-line, es un sistema de control de versiones. Sin embargo, existen clientes para Windows, como por ejemplo **Tortoise** que se integra con el explorador de Windows y sincroniza una carpeta con un proyecto versionado. La sincronización no es inmediata, debe hacerse manualmente siempre (o configurar que se haga cada X tiempo). El control de usuarios y de grupo es muy complejo y amplio, y el control de versiones es muy completo.

Sin embargo, se necesita de una configuración muy precisa y exhaustiva, que solamente usuarios con experiencia en el campo, o utilizando manuales complejos se puede realizar. De las tres alternativas ésta es la más difícil de utilizar y la menos intuitiva.

2.10.3. Lógica del sistema

Esta aplicación está orientada a ser utilizada en dos máquinas distintas (cliente y servidor), sin embargo como las comunicaciones funcionan por el protocolo TCP/IP se puede llegar a utilizar en una misma máquina (que haga de cliente y servidor a la vez). Aún así, la lógica en ambos casos es la misma. La lógica seguirá el siguiente esquema:

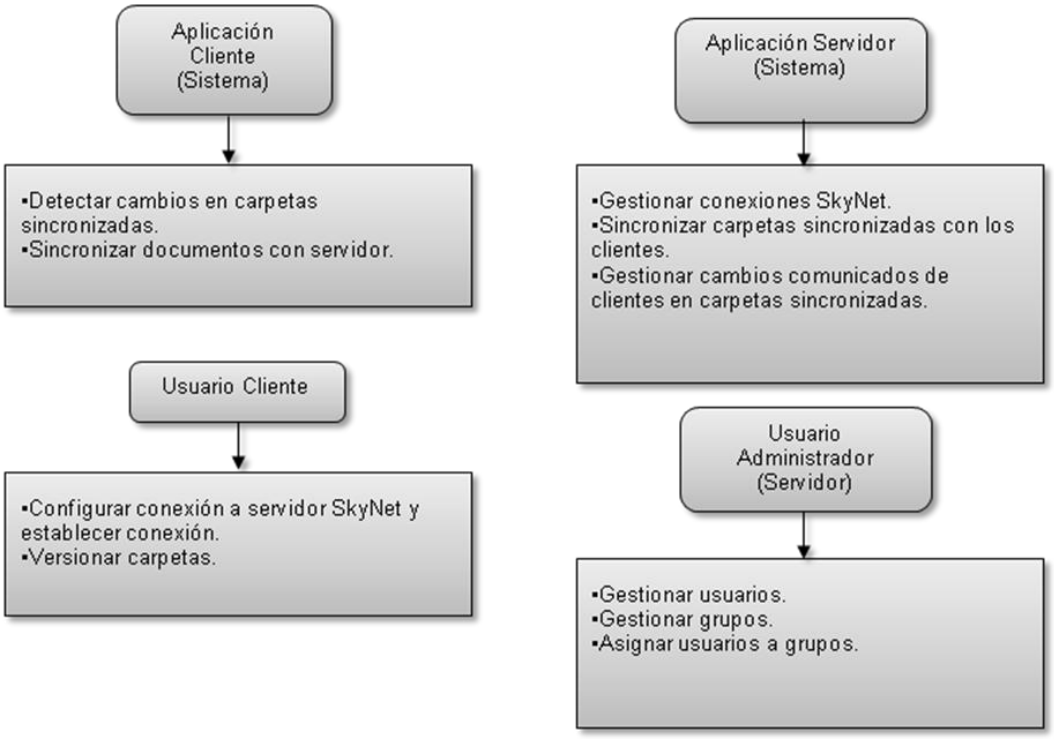


Fig 4. Lógica del sistema

2.10.4. Descripción física.

La infraestructura de este proyecto se basa en una aplicación cliente-servidor, por lo tanto, únicamente se necesitarán de máximo dos ordenadores para funcionar. La aplicación cliente tendrá una pequeña base de datos para almacenar los datos de conexión con un servidor de Sky Net, y la aplicación servidor tendrá una base de datos que almacenará los usuarios y sus contraseñas, los grupos y su contraseña de administrador, y una relación de usuarios-grupos.

Por lo tanto, el esquema quedaría así:

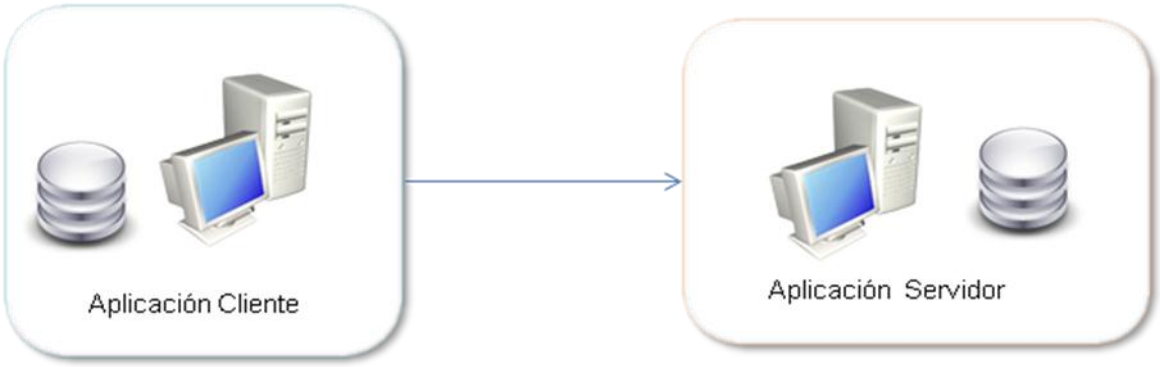


Fig 5. Descripción física del sistema

2.10.5. Usuarios del sistema

Nombre	Descripción	Responsabilidad
Usuario	Usuario estándar	Usuario que utiliza la aplicación cliente y puede conectarse a servidores Sky Net.
Administrador	Administrador	Administrador de la aplicación servidor Sky Net que puede gestionar los usuarios, grupos y asignar usuarios a grupos, así como configuraciones del sistema (puertos, carpeta raíz del servidor, etc.).
User	Usuario Sky Net	Usuario de Sky Net que representa una carpeta sincronizada. Se podría considerar como una cuenta de usuario de un Usuario estándar.
Group	Grupo Sky Net	Grupo de Sky Net, al cual puede acceder cualquier persona sin necesidad de estar registrada en el servidor. Su comportamiento es similar al del usuario Sky Net, pero se necesita de una contraseña para poder modificar los documentos (no visualizarlos). Puede contener usuarios.

Tabla 3. Usuarios del sistema.

2.10.6. Diagnóstico del sistema

Deficiencias actuales:

- Actualmente las soluciones que podemos encontrar nos ofrecen un servicio de almacenamiento on-line con sincronización de documentos, pero la gran mayoría solo nos permiten utilizar una cuenta de usuario (al menos de forma gratuita).
- Los servidores de dichas aplicaciones son completamente desconocidos, y los documentos pueden estar comprometidos.
- Las aplicaciones que podemos encontrar que subsanan estos problemas suelen ser caras o con mucho coste de implementación, así como de posibles incompatibilidades con sistemas de trabajo concretos.

Mejoras:

- Implementar un sistema de gestión de usuarios – grupos utilizando una estructura de carpetas, sin necesidad de complejas relaciones entre sí.
- Permitir que la aplicación cliente se conecte a varios servidores y a varias cuentas a la vez, separándolas en diferentes carpetas que trabajan por separado.
- Añadir un control de versiones simple y fácil de utilizar.

2.10.7. Normativas y legislación

- Normativa de proyectos de final de carrera de la EIS (Escuela de Informática de Sabadell)

- Ley de la propiedad intelectual. El proyecto es propiedad única y exclusivamente del alumno y de la universidad.

2.11. Requisitos del proyecto

2.11.1. Requisitos funcionales

1. Gestión de usuarios/grupos y asignación de usuarios a grupos.
2. Gestión de las conexiones entre cliente y servidor.
3. Sincronización entre cliente-servidor de carpetas sincronizadas.
4. Creación y almacenamiento de versiones de las carpetas sincronizadas

2.11.2. Requisitos no funcionales

1. Encriptación de datos de usuario: contraseñas
2. Utilización de sistemas de hash estándares para la identificación de documentos (MD5, SHA1)
3. Únicamente un usuario administrador del sistema operativo podrá hacer cambios en la aplicación servidor.

2.11.3. Restricciones del sistema

1. La aplicación únicamente funcionará bajo entorno Windows, con .NET FRAMEWORK 4.0 mínimo (Microsoft Windows Vista SP1, Windows 7, Windows Server 2008 R2)
2. Margen de finalización de proyecto 28 de junio del 2011
3. Para el desarrollo de la aplicación se utilizará el Microsoft Visual Studio 2010 Profesional (licencia MSDN de la UAB)
4. Para el desarrollo de ambas bases de datos (cliente y servidor) se utilizará ADO.NET 4.0 SQLite (Licencia Freeware).

2.11.4. Priorización de requisitos

	RF1	RF2	RF3	RF4
Esencial		X	X	
Condicional	X			
Opcional				X

Tabla 4. Prioridades de requisitos funcionales

	RNF1	RNF2	RNF3
Esencial	X		
Condicional	X		
Opcional			X

Tabla 5. Prioridades de requisitos no funcionales

	RF1	RF2	RF3	RF4	RNF1	RNF2	RNF3
O1		X				X	
O2	X				X		X
O3		X		X			
O4			X		X		

Tabla 6. Relación entre objetivos y requisitos.

2.12. Recursos

La clasificación de los recursos necesarios para realizar el proyecto es la siguiente:

- **Recursos humanos**: Personas que aportan los conocimientos y la mano de obra para la realización de este proyecto
- **Recursos de hardware:** La maquinaria y sus características necesarias para llevar a cabo este proyecto.
- **Recursos de software:** Software que necesitaremos para el desarrollo y la ejecución del proyecto.

2.12.1. Recursos humanos

El recurso humano principal es la persona encargada de la realización del proyecto. Se encargará de todas las tareas, desde la fase de análisis hasta la fase de documentación, pasando por diseño, desarrollo y juego de pruebas.

2.12.2. Recursos de hardware

Para este proyecto se utilizaran un total de 3 ordenadores distintos, una estación de trabajo profesional para el desarrollo de la aplicación, una estación de trabajo normal y corriente (portátil) para las pruebas de la aplicación cliente, y un ordenador servidor para la comprobación de la aplicación servidor.

	Estación de desarrollo	**Estación de trabajo portátil**	**Ordenador Servidor**
CPU	Intel Core2 Quad Q6600 2.4 Ghz	Intel Centrino Duo 2.4 Ghz	Intel XEON 3060, 2.40 Ghz
RAM	8 Gb RAM DDR2 1200 mhz	2 GB RAM DDR2 800	32 GB RAM DDR2 1200 Mhz
HDD	800 GB 7200 RPM SATA-2	650 GB 7200 RPM SATA-2	256 GB SSD / 2 TB SATA2 7200
Grafica	ATI RADEON 2400 PRO	nVidia GeForce 8500 GTX	ATI ES1000
SO	Windows 7 Professional	Windows 7 Professional	Windows Server 2008 R2

Tabla 7. Equipos utilizados y sus características.

2.12.3. Recursos de software

Distinguiremos dos partes: el software de desarrollo del proyecto, y el software utilizado para la documentación.

- Herramientas de desarrollo utilizadas en el proyecto:
  - **Microsoft Visual Studio 2010 Professional**: Entorno de desarrollo integrado (IDE) de pago realizado por Microsoft. Utilizado bajo la licencia de desarrollo de la UAB. Existe una versión gratuita (Express) que también sirve. Permite desarrollar aplicaciones utilizando el .NET FRAMEWORK, que es un conjunto muy extenso de librerías de Microsoft que puede utilizar el usuario de forma gratuita.
  - **SQLite ADO.NET 4.0:** Sistema gestor de base de datos gratuito y muy liviano. Se integra en el proyecto utilizando únicamente una librería y utilizando un archivo como base de datos. No contiene tanta seguridad y definiciones como otros sistemas gestores de bases de datos, pero para éste proyecto ya es suficiente.
  - **Modelio:** Diseñador de diagramas UML completo y gratuito. Permite la creación de todo tipo de diagramas que engloban todo el UML: Casos de uso, de secuencia, de estad, etc.

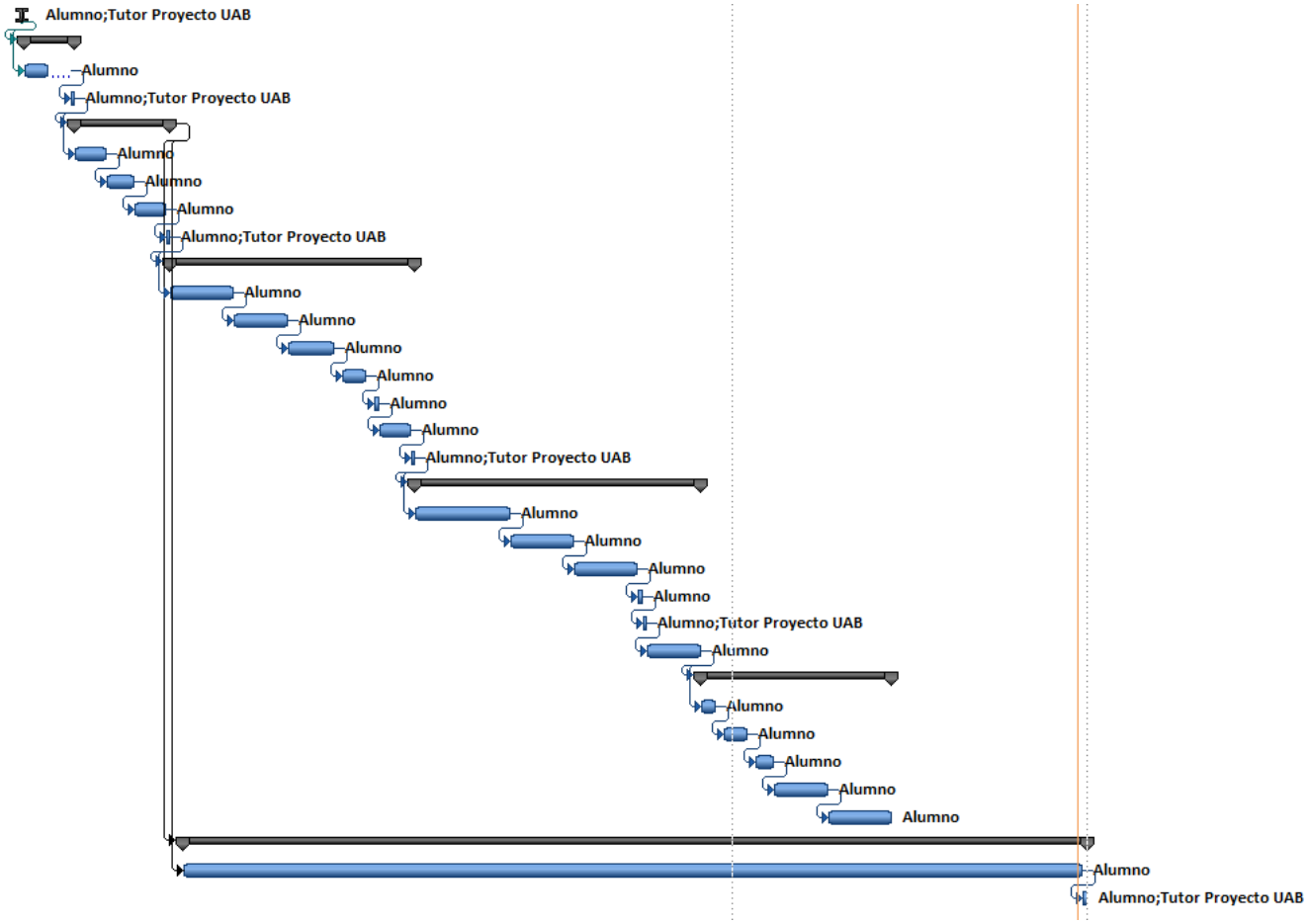
- Herramientas para la documentación del proyecto:
  - **Microsoft Office © Word 2010:** Utilizado para la creación del documento principal y sus anexos
  - **Microsoft Office © Powerpoint 2010:** Utilizado para la creación de la presentación en diapositivas del proyecto.
  - **Microsoft Office Project © 2010:** Utilizado para elaborar la planificación temporal del proyecto del estudio de viabilidad
  - **Microsoft Paint © 6.1:** Utilizado para la elaboración de algunos gráficos de la memoria.

2.13. Planificación del proyecto

2.13.1. Planificación Temporal

Nombre de tarea	Duración	Comienzo	Fin	Nombres de los recursos
Inicio del proyecto: Asignación del tutor y matricula vía WEB	**2 horas**	**dom 31/10/10**	**dom 31/10/10**	**Alumno; Tutor Proyecto UAB**
Planificación	**9 días**	**lun 01/11/10**	**jue 11/11/10**	
Estudio de viabilidad	5 días	lun 01/11/10	mié 10/11/10	Alumno
Verificaciones y correcciones por parte del tutor	8 horas	jue 11/11/10	jue 11/11/10	Alumno; Tutor Proyecto UAB
Análisis del proyecto	15 días	vie 12/11/10	jue 02/12/10	
Análisis de requisitos	5 días	vie 12/11/10	jue 18/11/10	Alumno
Análisis del modelo de datos	4 días	vie 19/11/10	mié 24/11/10	Alumno
Documentación de los requisitos y del modelo de datos	5 días	jue 25/11/10	mié 01/12/10	Alumno
Verificación del análisis por parte del tutor	8 horas	jue 02/12/10	jue 02/12/10	Alumno; Tutor Proyecto UAB
Diseño de la aplicación (diagramas, modelos de uso, etc.…)	**38 días**	**vie 03/12/10**	**mar 25/01/11**	
Diseño del protocolo de comunicaciones	10 días	vie 03/12/10	jue 16/12/10	Alumno
Diseño de la aplicación cliente	8 días	vie 17/12/10	mar 28/12/10	Alumno
Diseño de la aplicación servidor	8 días	mié 29/12/10	vie 07/01/11	Alumno
Diseño de la detección de archivos	5 días	lun 10/01/11	vie 14/01/11	Alumno
Diseño de la base de datos	1 día	lun 17/01/11	lun 17/01/11	Alumno
Documentación del diseño	5 días	mar 18/01/11	lun 24/01/11	Alumno
Verificación del diseño por parte del tutor	8 horas	mar 25/01/11	mar 25/01/11	Alumno; Tutor Proyecto UAB
Implementación	**45 días**	**mié 26/01/11**	**mar 29/03/11**	
Implementación del protocolo de comunicaciones	15 días	mié 26/01/11	mar 15/02/11	Alumno
Implementación de la aplicación cliente	10 días	mié 16/02/11	mar 01/03/11	Alumno
Implementación de la aplicación servidor	10 días	mié 02/03/11	mar 15/03/11	Alumno
Implementación de la base de datos	1 día	mié 16/03/11	mié 16/03/11	Alumno
Verificación del código por parte del tutor	8 horas	jue 17/03/11	jue 17/03/11	Alumno; Tutor Proyecto UAB
Corrección de errores	8 días	vie 18/03/11	mar 29/03/11	Alumno
Test de pruebas	**30 días**	**mié 30/03/11**	**mar 10/05/11**	
Pruebas de la comunicación cliente-servidor	3 días	mié 30/03/11	vie 01/04/11	Alumno
Pruebas de estrés	5 días	lun 04/04/11	vie 08/04/11	Alumno
Pruebas de sincronización en banda estrecha	4 días	lun 11/04/11	jue 14/04/11	Alumno
Pruebas de CVS	8 días	vie 15/04/11	mar 26/04/11	Alumno
Corrección de errores	10 días	mié 27/04/11	mar 10/05/11	Alumno
Documentación	**143 días**	**lun 06/12/10**	**mié 22/06/11**	
Elaboración de la documentación	142 días	lun 06/12/10	mar 21/06/11	Alumno
Validación de la documentación	1 día	mié 22/06/11	mié 22/06/11	Alumno; Tutor Proyecto UAB

2.13.2. Diagrama de Gantt



2.14. Evaluación de riesgos

Existen dos riesgos principales en este proyecto: el tiempo y la posibilidad de que se cree algún software similar en el transcurso del proyecto.

El riesgo del tiempo consta básicamente del desconocimiento real de un diseño efectivo con la gestión y detección de cambios en documentos en la máquina cliente. Se sabe que un sistema Windows genera cuatro tipos de cambios en un documento: crear, modificar, renombrar y eliminar, pero también existen combinaciones entre ellos, y los programas de terceros pueden crear un archivo utilizando varias de estas combinaciones. Por lo tanto, lo que va a consumir más tiempo es en entender y diseñar una buena gestión de documentos para Windows. Para solucionar este riesgo se utilizará la librería de ayuda gratuita de Microsoft MSDN, que siempre está actualizada con los últimos cambios en los sistemas operativos y las librerías utilizadas.

Por otro lado, las varias aplicaciones alternativas a éste proyecto están en continuo proceso de actualización durante el transcurso del curso académico del proyecto, con el fin de aumentar y ofrecer más funcionalidades. Por lo tanto podría ocurrir que al finalizar el proyecto éstas alternativas incluyeran cambios similares al proyecto, lo cual reduciría la originalidad de ésta aplicación.

2.15. Presupuesto

Para la realización de éste proyecto realmente no habrá ningún tipo de inversión económica en él. Todos los equipos informáticos utilizados son de propiedad del alumno, y el software utilizado lo proporciona la entidad universitaria (UAB) bajo la licencia de Microsoft MSDN.

Sin embargo, si éste proyecto se realizara fuera de ámbito universitario, por ejemplo para una empresa o desarrollo personal, el presupuesto sería el siguiente:

Material Hardware	
Equipo de desarrollo, marca DELL	480 €
Estación de trabajo (portátil), marca DELL	550 €
Estación Servidor	1800 €
Material Software	
Microsoft Office 2010 (Microsoft Word ©, Microsoft Excel ©, Microsoft PowerPoint ©, Microsoft Project ©) Versión estudiantes	90€
SQLite ADO.NET 4.0	gratuito
Microsoft Visual Studio 2010 Professional	550€
Diseño y Desarrollo	
400 h de diseño e implementación (16€/h, precio estándar)	6400 €
TOTAL	**9870 €**

Pero como ya hemos comentado, el hardware en este proyecto es propiedad del alumno, y el software lo proporciona la entidad UAB con la licencia de Microsoft MSDN. Únicamente se podrían considerar las horas de diseño e implementación, en cuyo caso el presupuesto sería de **6400 €.**

2.16. Conclusiones

Una vez vista la situación actual, y las alternativas que existen en el mercado podemos decir que aunque existen alternativas que pueden ofrecernos un servicio similar a lo que proponemos, dichas alternativas contienen un alto pago, o alto mantenimiento monetario. Además, nos quedará siempre la duda de la seguridad de nuestros documentos, ya que se almacenarán en servidores a los que no podemos acceder directamente, o podemos supervisar su seguridad.

Por otro lado, las herramientas utilizadas para la realización de éste proyecto son prácticamente gratuitas, gracias a la licencia de la entidad UAB, y cómo la aplicación se realiza como proyecto de final de carrera, el coste es prácticamente nulo.

Por lo tanto, estamos ante una aplicación que ofrece unos servicios que pueden llegar a mejorar a las alternativas que actualmente existen en el mercado, y en la práctica, no existe ninguna limitación en cuanto a hardware se refiere. El único requisito que tendrá la aplicación es que los ordenadores donde se ejecuten dispongan de conexión a internet para poder funcionar. La conclusión del estudio de viabilidad, entonces, es que el proyecto es totalmente ***Viable***.

CAPÍTULO 3

DISEÑO DE LA APLICACIÓN

3. Diseño de la aplicación

3.1. Introducción

Como se ha explicado anteriormente, ésta aplicación constará de tres partes distintas: el protocolo de comunicaciones, la aplicación cliente, y la aplicación servidor. La aplicación cliente y servidor utilizará el protocolo para enviarse mensajes de control de la una a la otra, y en caso de necesitar el archivo, se enviará como dato adjunto al mensaje de control (utilizando un puerto distinto).

Los datos que se intercambiarán el cliente y el servidor serán el log-in de usuario o grupo (o combinación de ambos), un esquema de la situación actual de las carpetas sincronizadas (para que el servidor sepa qué archivos tiene el cliente, lo veremos más adelante), los cambios en las carpetas sincronizadas en la maquina cliente, y los archivos que sean necesarios para que dichos cambios queden registrados en el servidor.

El esquema de comunicación será el siguiente:

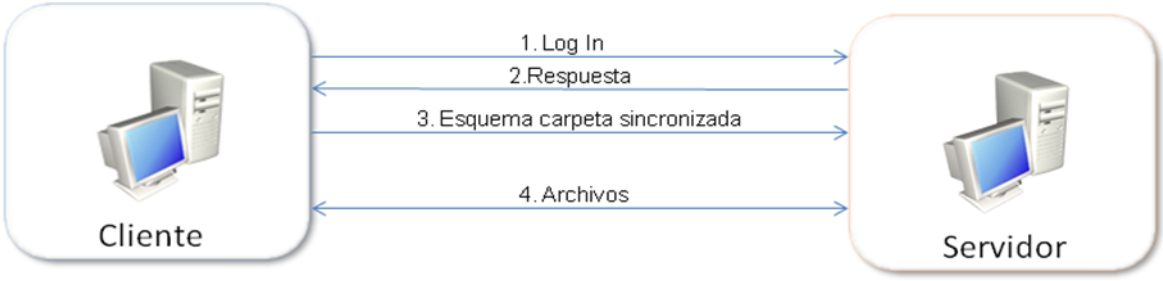


Fig 6.Tipos de comunicaciones entre cliente y servidor

3.2. Definiciones y características de los módulos.

Analizaremos por separado el protocolo de comunicaciones, la aplicación cliente, y la aplicación servidor.

3.2.1. Protocolo de comunicaciones

Los casos de uso del protocolo de comunicaciones son los siguientes:

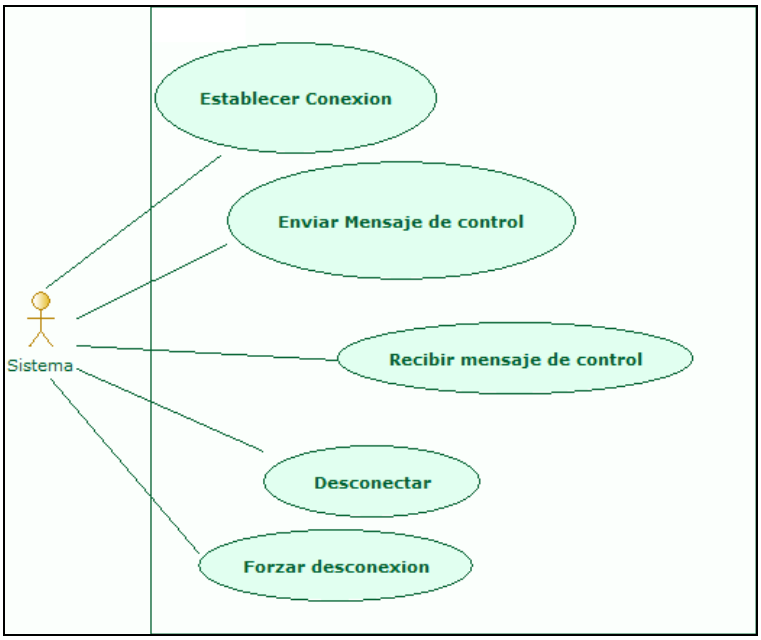


Fig 7.Diagrama de casos de uso del protocolo de comunicaciones

Definimos los posibles casos de uso que el sistema puede tener con el protocolo de comunicaciones:

Establecer Conexión

1) El sistema intentará conectarse vía socket[2] a la dirección IP y puerto proporcionados por la aplicación
    a. En caso de error, saltará evento de error de conexión
2) Se establece el estado de la conexión en “Activa”.
3) Se ejecuta un thread[3] paralelo que se quedará leyendo el socket hasta que el estado de la conexión pase a “Desconectado”.

[2] Socket: mecanismo lógico (en lenguaje de programación, es una librería) que permite establecer una conexión de datos entre dos máquinas usando protocolo TCP/IP e intercambiar datos punto a punto.

Enviar Mensaje de control

1) Poner el estado del protocolo en “Enviando mensaje”
    a. Si el estado ya estaba en Enviando mensaje, poner el mensaje en cola, y finalizar.
2) Enviar el mensaje de control por el socket establecido en el establecimiento de conexión.
3) Si el mensaje es de tipo “Enviar archivo”:
    a. Comprobar que la ruta proporcionada por parámetro es correcta, y se puede acceder al archivo en modo lectura.
    b. Se envía después del mensaje de control el archivo byte a byte por el socket establecido.
4) Comprobar la cola de mensajes
    a. Si la cola de mensajes está vacía, finalizar y poner el estado en “Disponible”
    b. Si la cola de mensajes tiene mensajes por enviar, volver al paso 2 con el siguiente mensaje de la cola.

Desconectar

1) Enviar comando de detención al thread de lectura del socket.
2) Enviar mensaje de control de desconexión por el socket de envío.
3) Una vez recibido el evento de cierre del thread, desconectar el socket de mensajes de control.,
4) Desconectar el socket de ficheros.
5) Establecer estado en “desconectado”
6) Lanzar evento de desconexión realizada con éxito.

[3] Thread: Un Thread es una aplicación ejecutándose en segundo plano o en paralelo a la aplicación principal. Permite que el procesador sea capaz de ejecutar ambas aplicaciones (o trozos de aplicación) simultáneamente

Recibir Mensaje de control

1) Leer los bytes que hay en el búfer[4] del socket establecido.
2) Reconstruir mensaje de control a partir de los bytes
    a. En caso de que no haya mensaje de control, seguir leyendo del socket hasta encontrar mensaje.
    b. Si no se encuentra mensaje después de varios intentos, hacer saltar evento de error de lectura y vaciar búfer.
3) Interpretar el mensaje de control:
    a. Si el mensaje no es del tipo “enviar archivo”, hacer saltar evento de mensaje recibido con el mensaje de control como parámetro
    b. Si el mensaje es del tipo “enviar archivo”:
        i. Intentar abrir la ruta de archivo temporal establecida como parámetro de la función, en modo escritura.
            a. En caso de no poder abrir la ruta de archivo temporal, lanzar excepción de error de apertura de archivo.
        ii. Establecer variable temporal de bytes de fichero leídos a 0.
        iii. Mientras los bytes de fichero leídos sean menores a los bytes de ficheros esperados (pasados como parámetro en el mensaje de control de enviar fichero):
            a. Recibir bytes de fichero en bloques de 1024 bytes. (1Kb).
            b. Escribir bytes de fichero en bloques de 1024 bytes (1Kb).
        iv. Lanzar evento de fichero recibido con éxito, enviando la ruta del archivo temporal como parámetro.
4) Si el socket sigue en estado “conectado”, volver al paso 1.
    a. En caso contrario, finaliza el thread.

[4] Búfer: Espacio de memoria reservado donde se guardan datos que posteriormente se irán leyendo. En .NET FRAMEWORK 4.0 los búferes de los sockets son teóricamente ilimitados, pero a la práctica pueden llegar a ser tan grandes como memoria RAM disponible se tenga en el ordenador.

Forzar desconexión (En caso de detección de funcionamiento incorrecto del sistema).

1) Enviar comando de detención inmediata al thread de lectura del socket.
2) Ignorar los mensajes de error de thread interrumpido bruscamente.
3) Forzar la desconexión del socket de mensajes de control.
4) Forzar la desconexión del socket de ficheros.
5) Si los búferes de ambos sockets están parcialmente llenos, vaciarlos.
6) Enviar evento de desconexión forzosa realizada con éxito.

3.2.2. Aplicación cliente

En la aplicación cliente distinguiremos los diferentes casos de uso que podrá tener el usuario físico, de los casos de uso que podrá tener el sistema (la aplicación).

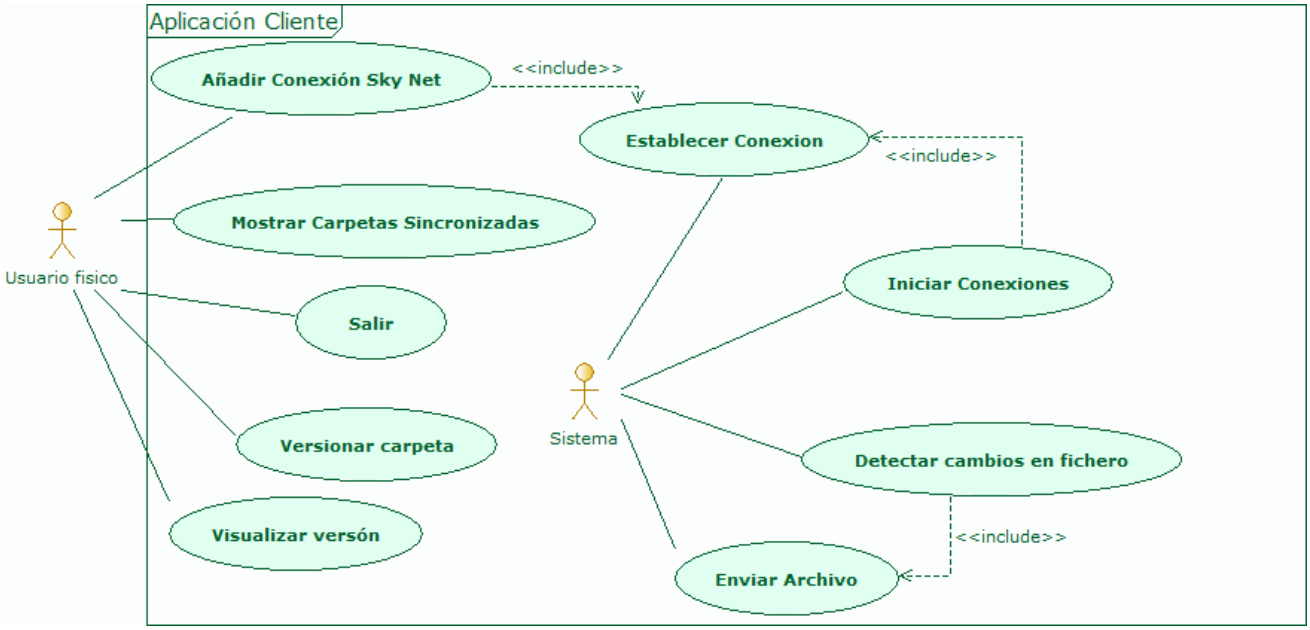


Fig 8. Diagrama de casos de uso de la aplicación cliente

Usuario Físico

Añadir Conexión Sky Net

1) Mostrar formulario pidiendo los siguientes datos de conexión: Usuario/grupo, contraseña, dirección IP del servidor, puerto de conexión, puerto de ficheros.
2) Guardar en la base de datos la configuración de conexión.
    a) La representación del formato de conexión será: “user_grupo@servidor”.
3) Crear una carpeta que será la carpeta sincronizada. La carpeta se llamará user_grupo@servidor.
    a) En caso de que exista la carpeta, no hacer nada, ya se utilizará esa.
4) Establecer conexión Sky Net (caso de uso del sistema).

Mostrar carpetas sincronizadas

1) Mostrar formulario con una lista de las carpetas sincronizadas que tiene el programa.
    a) En caso de que la conexión esté establecida correctamente, mostrar un icono de carpeta normal.
    b) En caso de que la conexión no esté establecida correctamente, mostrar un icono de carpeta con una cruz o símbolo que represente que la carpeta no está siendo sincronizada.

Salir

1) Interrumpir todas las conexiones y el tráfico de datos / mensaje de control.
2) Desconectar todas las conexiones de Sky Net activas.
3) Salir del programa.

Versionar carpeta

1) Mostrar formulario que pedirá al usuario un nombre o un número de versión para la carpeta seleccionada.
2) Se enviará una petición al servidor con el mensaje de control de versionar carpeta, y el identificador del usuario.
3) Se mostrará un mensaje de versión completado correctamente.
    a) En caso de que el servidor no pueda versionar por algún motivo (nombre o versión ya existente), se mostrará un mensaje de error con el error devuelto.

Mostrar versión

1) Pedir al servidor una lista de versiones de la carpeta.
2) Mostrar formulario al usuario con una lista de las diferentes versiones de la carpeta
    a) En caso de que no exista aún ninguna versión, mostrar mensaje de que no hay versiones.
3) Una vez el usuario seleccione la versión, se envía una petición al servidor de ver la versión.
4) Se mostrará un formulario al usuario pidiéndole dónde desea guardar la versión
5) Se mostrará un mensaje de versión completado correctamente.
    a) En caso de que el servidor no pueda versionar por algún motivo (nombre o versión ya existente), se mostrará un mensaje de error con el error devuelto.

Sistema (aplicación)

Iniciar Conexiones (Esto ocurre justo cuando inicia el programa)

1) Conectar a la base de datos.
2) Recuperar la información de las conexiones almacenadas en la base de datos.
3) Por cada conexión:
    I. Establecer conexión

Establecer conexión

1) Establecer conexión utilizando el protocolo de comunicaciones.
    a) En caso de que la conexión no sea realizada con éxito, mostrar mensaje de error al usuario.
2) Recuperar la estructura de la carpeta sincronizada que pertenece a la conexión. Los datos que se deben recuperar de cada archivo son: Nombre, tamaño, fecha de creación, fecha de modificación, Hash[5].
3) Enviar la estructura de la carpeta sincronizada mediante un mensaje de control al servidor.
4) Añadir un visualizador de cambios a la carpeta sincronizada, con tal de detectar cualquier tipo de cambio dentro de la carpeta.

Detectar cambios en fichero

1) Identificar el tipo de cambio.
2) Enviar un mensaje de control dependiendo del tipo de cambio:
    a) Archivo Creado.: Enviar archivo al servidor utilizando el mensaje de control de archivo creado.
        i. Si el archivo creado tiene un tamaño de 0 bytes se interpretará como archivo vacío y no se enviará.
    b) Archivo Modificado.: Enviar archivo al servidor utilizando el mensaje de control de archivo modificado.
    c) Archivo renombrado: Se enviará un mensaje de control al servidor de archivo renombrado, con la ruta antigua y la nueva ruta.
    d) Archivo borrado: se enviará un mensaje de control al servidor de archivo eliminado, con la ruta del archivo.

Enviar archivo

1) Enviar mensaje de control de envío de archivo con la ruta del archivo.

[5] Hash: Utilizando un algoritmo matemático con el contenido del fichero se obtiene un identificador teóricamente único del fichero. Se utiliza para hacer diferencias entre ficheros en caso que no sea posible identificarlos por otros medios.

3.2.3. Aplicación servidor

En la aplicación servidor volvemos a tener el esquema de usuario físico (ésta vez llamado administrador), y sistema, que actuará automáticamente respondiendo a los eventos de mensajes recibidos desde el protocolo de comunicaciones.

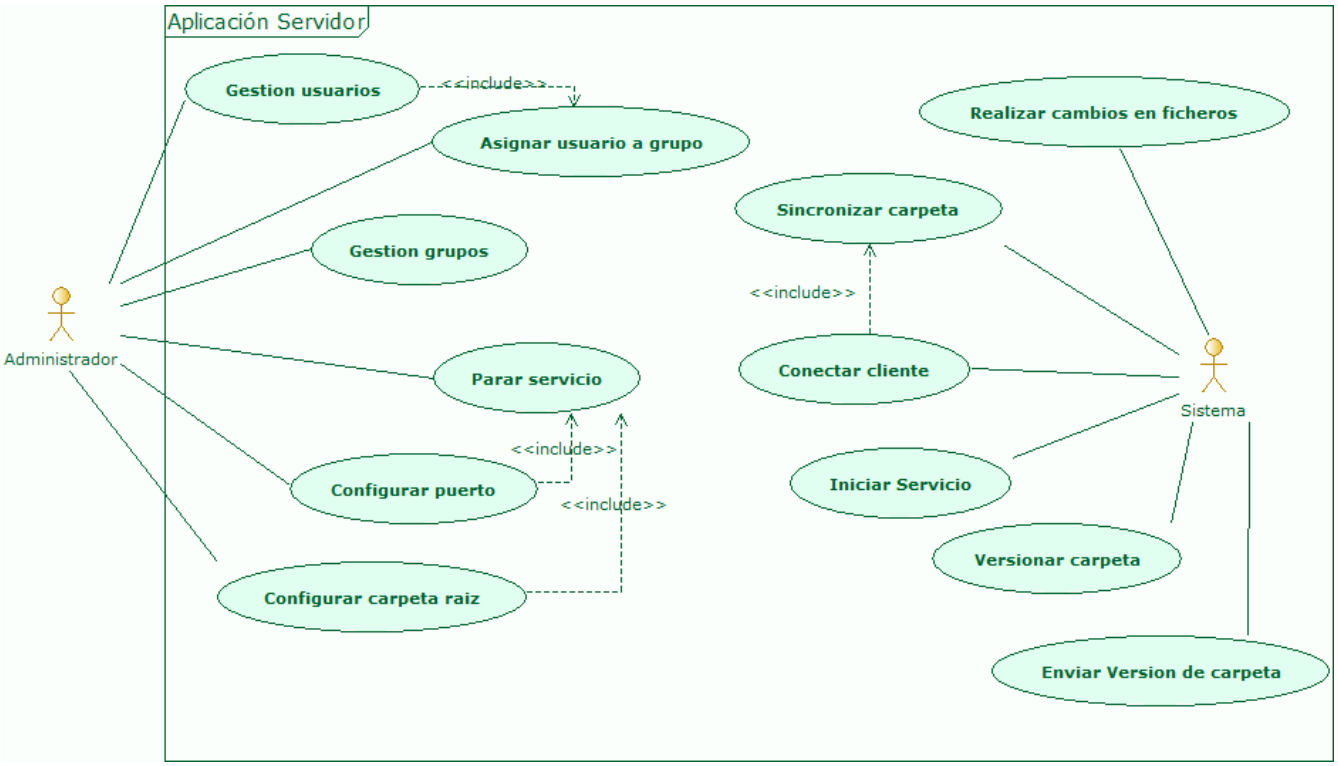


Fig 9. Diagrama de casos de uso de la aplicación servidor

Usuario Administrador

Gestión de Usuarios

1) Mostrar formulario que contiene los usuarios y las opciones crear, modificar parámetros y eliminar.
    a) Crear Usuario: Guardar los datos establecidos en la base de datos y crear la carpeta de usuario
    b) Modificar usuario: Guardar los nuevos datos en la base de datos, y desconectar los usuarios conectados con este nombre de usuario
    c) Eliminar usuario: Pedir confirmación de eliminación
        i. En caso afirmativo: Desconectar los usuarios que tengan este nombre de usuario, borrar datos de la base de datos y eliminar la carpeta de usuario.
        ii. En caso negativo: no hacer nada.

Gestión de Grupos

1) Mostrar formulario que contiene los grupos y las opciones crear, modificar parámetros y eliminar.
    a) Crear grupo: Guardar los datos establecidos en la base de datos y crear la carpeta del grupo.
    b) Modificar grupo: Desconectar a los usuarios conectados como rol administrador de grupo. Guardar los nuevos datos en la base de datos.
    c) Eliminar grupo: Pedir confirmación de eliminación.
        i. En caso afirmativo: Desconectar los usuarios que estén conectados a este grupo, borrar datos de la base de datos y eliminar la carpeta del grupo.
        ii. En caso negativo: no hacer nada.

Asignar usuario a grupo

1) Mostrar formulario que contiene una lista de usuarios y sus grupos asignados, con las opciones de añadir grupo y eliminar grupo al usuario.
    a) Añadir al grupo: Guardar la relación en la base de datos y crear una carpeta de usuario dentro de la carpeta del grupo.
    b) Eliminar del grupo: Desconectar a los usuarios que esté conectados, borrar la carpeta del grupo, y eliminar la relación de usuario-grupo de la base de datos.

Parar servicio

1) Pedir confirmación de que se desea parar el servicio
    a) En caso negativo, no hacer nada.
2) Desconectar todos los usuarios conectados al servidor con un mensaje de desconexión.
3) Detener el servicio de escucha de sockets.

Configurar puertos

1) Mostrar formulario de configuración de puertos, con un campo de texto para poner el puerto y botón para guardar configuración
2) En caso de cambiar el puerto y tener encendido el servicio, parar servicio.
3) Guardar puerto.
4) Encender servicio de escucha del socket.

Configurar carpeta raíz

1) Mostrar formulario de selección de carpeta
2) En caso de que el servicio esté activo, pedir confirmación de cambiar la carpeta raíz
    a) En caso negativo, no hacer nada
3) Parar servicio.
4) Guardar posición de carpeta raíz
5) Mover todos los archivos de la antigua carpeta raíz a la nueva carpeta raíz.
6) Encender el servicio de la escucha de socket.

Usuario Administrador

Realizar cambios en fichero

1) Analizar el mensaje de control:
    a) Crear Archivo: Leer el fichero adjunto del evento y guardarlo en la carpeta sincronizada del usuario
    b) Modificar archivo: Leer el fichero adjunto del evento y guardarlo en la carpeta sincronizada del usuario.
    c) Renombrar archivo: Renombrar el fichero de la carpeta sincronizada al nombre especificado en el evento
    d) Eliminar archivo: Eliminar el archivo especificado en el parámetro del evento
2) En cualquiera de los 4 casos anteriores: Reenviar el mensaje de control y el archivo adjunto si hay a los demás clientes conectados con el mismo nombre de usuario o grupo

Sincronizar carpeta

1) Utilizando la tabla de contenidos de carpeta sincronizada que nos ha enviado el cliente, comparar archivo a archivo:
    a) Si la carpeta sincronizada contiene el archivo, y el cliente no, marcarlo como archivo a enviar a cliente.
    b) Si la carpeta sincronizada no contiene el archivo, y el cliente si, marcarlo para pedirlo al cliente.
    c) Si ambos contienen el archivo, distinguirlos usando la fecha de modificación:
        i. Si el del cliente es más nuevo, marcarlo para pedirlo al cliente.
        ii. Si el de la carpeta es más nuevo, marcarlo para enviarlo al cliente.
        iii. Si ambas fechas son las mismas, repetir comparación con hash MD5 de archivo. (caso muy poco frecuente).
        iv. Si no se puede distinguir, ignorar el archivo.
2) Por cada archivo que tenemos que pedir al cliente:
    a) Enviar mensaje de control de recuperar archivo, con la ruta del archivo, al cliente que ha enviado la tabla de contenidos.
3) Por cada archivo que tenemos que enviar al cliente:
    a) Enviar mensaje de control de enviar archivo con el archivo adjunto al cliente de la tabla de contenidos.

Conectar cliente

1) Aceptar la petición de conexión por parte de una aplicación cliente.
2) Esperar mensaje de log-in
    a) En caso de que no llegue mensaje de login durante un tiempo determinado, desconectar.
3) Comprobar si el usuario – grupo y su contraseña corresponden con los almacenados en la base de datos.
    a) En caso negativo, enviar mensaje de login error al cliente.
4) Pedir tabla de contenidos al cliente.
5) Añadir la conexión al cliente en la colección de clientes conectados, identificándolo con el nombre de usuario.

Iniciar servicio

1) Comprobar que tenemos configurada la carpeta raíz
    a) En caso contrario, mostrar mensaje al administrador de que debe configurar la carpeta raíz.
2) Iniciar el servicio de escucha del socket, utilizando el puerto configurado (o predefinido: 5000)
    a) En caso de error, informar de él mediante un mensaje en pantalla al administrador
3) Iniciar la aceptación de conexiones vía socket.

Versionar carpeta

1) Poner todos los clientes conectados que utilicen este nombre de cliente en modo de espera (los mensajes enviados y recibidos se quedan bloqueados hasta nuevo aviso, utilizando semáforos en threads[6]).
2) Copiar la carpeta sincronizada y su contenido en una nueva carpeta que contiene el nombre dado por el cliente (nombre o versión) codificado en el nombre.
3) Cuando se finalice la copia de la carpeta, se enviará una señal al semáforo de los threads para que continúen con el proceso.

Enviar Versión de carpeta

1) Localizar la carpeta versionada dentro del sistema de archivos de la carpeta de usuario
    a) En caso de no localizarla, enviar mensaje de que esa versión no existe al usuario
2) Comprimir el contenido en un archivo ZIP[7]
3) Enviar el archivo Zip por socket mediante un mensaje de control de enviar versión, adjuntando el archivo Zip al mensaje.

[6] Semáforo en thread: es un bloqueo lógico (escrito en código) que detiene la ejecución del código hasta nuevo aviso.

[7] Archivo ZIP: Archivo único que contiene más archivos dentro, utilizando un estándar de compresión. De ésta forma podemos enviar muchos archivos compactados y ocupando menos espacio.

3.2. Definiciones del modelo de datos.

En nuestro proyecto utilizaremos dos bases de datos, la base de datos de la aplicación cliente, y la base de datos de la aplicación servidor.

Las entidades de las bases de datos en cuestión serán:

APLICACIÓN	NOMBRE	TIPO	DESCRIPCION
CLIENTE	Conexión	TABLA	Tabla que contiene la información de una conexión a servidor Sky Net.
SERVIDOR	Usuario	TABLA	Tabla que contiene la información de configuración de los usuarios.
SERVIDOR	Grupo	TABLA	Tabla que contiene la información de configuración de los grupos.
SERVIDOR	UsuarioGrupo	TABLA	Tabla que contiene la relación de grupo-usuario.

Tabla 8. Entidades de las bases de datos de aplicación y cliente

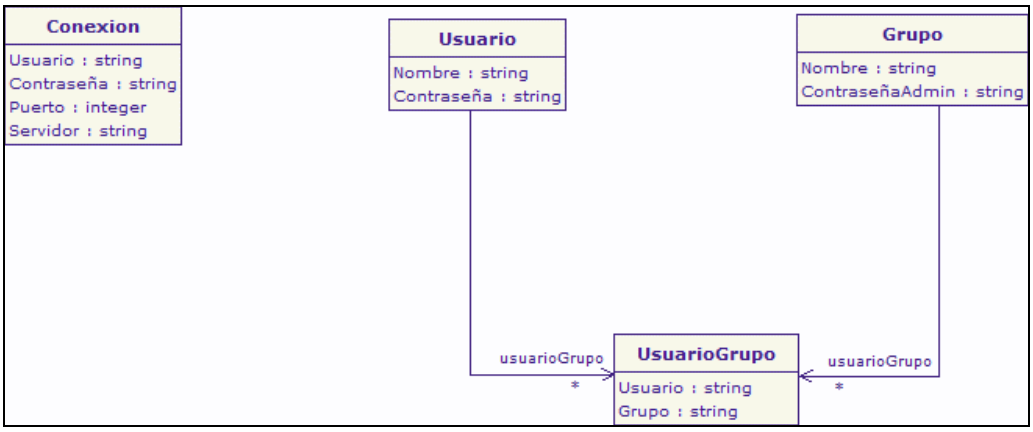


Fig 10. Diagrama entidad-relación del modelo de datos almacenado

3.2.1 Tabla Conexión

En esta tabla guardaremos los datos de configuración por parte del usuario físico de las múltiples conexiones a servidores Sky Net.

Campo	Tipo	Descripción
Usuario	TEXT	Nombre del usuario/grupo que identifica la conexión
Contraseña	TEXT	Contraseña almacenada codificada con MD5
Puerto	INTEGER	Número del Puerto. Por defecto 5000.
Servidor	TEXT	IP o nombre del servidor.

Nota: en SQLite los cambios de texto (tipo STRING) son TEXT. No existe VARCHAR.

3.2.2 Tabla Usuario

En esta tabla simplemente guardaremos el nombre de usuario y su contraseña de conexión. La información sobre ubicación de carpeta sincronizada no es necesaria, ya que la aplicación trabajará directamente sobre el sistema de archivos.

Campo	Tipo	Descripción
Nombre	TEXT	Nombre del usuario que identifica la conexión. No puede haber duplicados.
Contraseña	TEXT	Contraseña almacenada codificada con MD5

3.2.2 Tabla Grupo

En esta tabla simplemente guardaremos el nombre del grupo y su contraseña de administrador. La información sobre ubicación de carpeta sincronizada no es necesaria, ya que la aplicación trabajará directamente sobre el sistema de archivos.

Campo	Tipo	Descripción
Nombre	TEXT	Nombre del grupo que identifica la conexión. No puede haber duplicados.
ContraseñaAdmin	TEXT	Contraseña de administrador almacenada codificada con MD5.

3.2.2 Tabla UsuarioGrupo

En esta tabla guardaremos la relación existente de un grupo y un usuario (que un usuario pertenece a un grupo). Un mismo usuario puede estar en varios grupos, y un grupo puede tener varios usuarios.

Campo	Tipo	Descripción
Usuario	TEXT	Nombre del usuario de la relación
Grupo	TEXT	Nombre del grupo de la relación

3.3. Definiciones del comportamiento de la aplicación

3.3.1 Protocolo de comunicaciones

3.3.1.1 Comportamiento

La primera fase del comportamiento del protocolo de comunicaciones es la del establecimiento de conexión de dos máquinas (conexión punto a punto). Para ello se utilizará la clase socket definida en el .NET FRAMEWORK 4.0. En éste conjunto de librerías de Microsoft la clase socket se utilizan dentro de las clases “TcpClient” y “TcpServer” , que aportan más métodos de control a la clase socket nativa.

Si el servicio de comunicaciones del protocolo se inicia en una maquina servidora, el protocolo aceptará conexiones ya establecidas (objetos de la clase TcpClient), mientras que si el servicio se inicia en una maquina cliente, se creará utilizando la dirección IP y el puerto. De esta manera podemos distinguir si es cliente o servidor.

Una vez establecida la comunicación (en caso contrario se lanzaría una excepción de error de comunicación), el protocolo iniciará un thread con el módulo de lectura. Éste thread se quedará a la espera indefinidamente (hasta que la conexión se interrumpa de forma forzosa, o acordada por parte de ambos extremos) a que el otro extremo envíe datos.

El módulo de envío puede enviar dos tipos de mensajes: mensaje de control y archivo.

- En el caso del mensaje de control, se enviará por el canal de datos del socket establecido. Como sólo se pueden enviar bytes por un socket, el mensaje de control (que es un objeto de una clase definida) se serializará[8] en una cadena de bytes, y se enviará por el canal de datos.
- En el caso de enviar archivos, primero se generará un mensaje de control informando del tamaño del archivo y de la ruta del archivo (utilizando el anterior método). Una vez enviado, se enviará por el canal de socket los bytes del archivo en bloques de 1024 bytes (1 Kb), para facilitar la tarea del receptor de ir guardando el archivo en una ruta temporal y no sobrecargar la memoria RAM del receptor con el archivo (Un mal diseño sería ir guardando los bytes recibidos en la memoria, ya que un archivo puede llegar a tener un tamaño superior a ello).

El módulo de recibir datos se estará ejecutando siempre en segundo plano, y únicamente se activará en el momento que el búfer de entrada del socket reciba datos. Cuando se lean los bytes del canal de datos del socket, se intentará de serializar los bytes recibidos, con tal de reconstruir el mensaje.

Una vez reconstruido el mensaje, si el mensaje es del tipo enviar archivo, leeremos la cantidad de bytes especificada en el parámetro del mensaje de control y guardaremos el archivo en la ruta temporal. Una vez hayamos recibido el archivo, generaremos un evento de fichero recibido. En caso de no ser un envío de fichero, simplemente generaremos un evento de mensaje de control recibido.

El protocolo incluirá métodos para finalizar una conexión de forma segura y forzada. En la desconexión segura ambos extremos se enviarán mensajes de desconexión y procederán a la desconexión cuando los búferes se vacíen. En el caso de la desconexión forzosa, se detendrá el thread de lectura de forma inmediata, y se ignorarán todos los errores de interrupción brusca de threads y conexión.

[8] Serializar: proceso en el cual un objeto que está almacenado en la memoria RAM del ordenador que está ejecutando el programa, se transforma en una cadena de bytes. De ésta forma se pueden compartir objetos entre programas, o guardarlos en ficheros en el disco duro. La operación inversa de serializar se llama deserializar, y consta de pasar de una cadena de bytes a un objeto en memoria.

3.3.1.2 Diagrama de clases

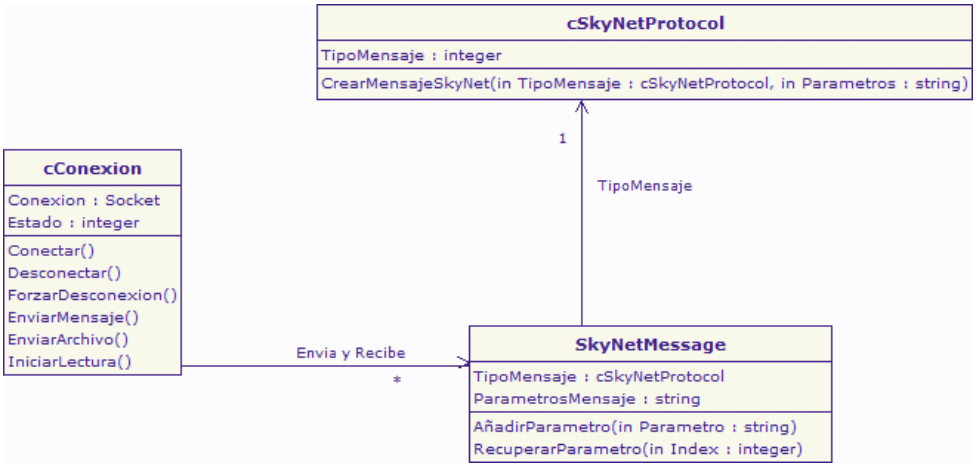


Fig 11. Diagrama de clases del módulo de protocolo de comunicaciones.

3.3.1.3 Definición de clases

El módulo de protocolo de comunicaciones contiene 3 clases:

- **cConexion:** Clase principal del protocolo de comunicaciones. Ésta clase se encarga de establecer conexión vía socket con otra máquina, y de enviar y recibir los mensajes de control definidos dentro del Sky Net Protocol.
- **cSkyNetProtocol:** Clase que define los diferentes tipos de mensajes de control posibles (los enumera), y se encarga de construir los mensajes de control, añadiéndoles los parámetros.
- **SkyNetMessage:** Clase que representa un mensaje de control del protocolo de comunicaciones Sky Net. Contiene el tipo de mensaje que es, y los parámetros asociados a él. En sí simplemente es una estructura de datos, no tiene ninguna función ni trabaja con los parámetros.

3.3.1.4 Tipos de mensaje en el protocolo Sky Net.

Estarán predefinidos dentro del protocolo varios tipos de mensaje, para facilitar la identificación de los mensajes de control dentro de las aplicaciones cliente y servidor. Los mensajes están divididos en control lógico y control de archivos. Serán los siguientes:

Mensajes de control lógico

- **LOGIN**: Petición de login por parte del cliente, y tiene como parámetros el nombre de usuario o grupo, y el de la contraseña, siendo éste último opcional en el caso del grupo
- **LOGIN_RESPONSE**: Respuesta por parte del servidor a una petición de login. Tendrá como parámetro el mensaje ACCEPT, o DENIED, dependiendo de si se acepta el login o no, y en el caso de DENIED puede llevar opcionalmente un segundo parámetro indicando el motivo.
- **FILESYNC_RETRIEVE**: Mensaje generado por el servidor una vez aceptado el login. Es un mensaje que indica al cliente que le envíe su estructura de la carpeta sincronizada, explicando la situación de cada uno de los archivos.
- **FILESYNCTABLE**: Respuesta del cliente a un mensaje de tipo *FILESYNC_RETRIEVE.* Contiene como dato adjunto un objeto que representa la estructura de la carpeta sincronizada.
- **MESSAGE***:* Este mensaje contiene como parámetro un mensaje que se envía un extremo a otro. Se utilizará en tareas de mantenimiento para avisar al cliente de que va a ser desconectado en un intervalo de tiempo corto.
- **DISCONNECT:** Este mensaje indica al otro extremo que se va a proceder a la desconexión. Se utilizará para parar todos los threads que estén escuchando el socket, y para realizar una desconexión limpia, sin errores. Puede contener como parámetro un mensaje a mostrar.
- **DO_VERSION:** Este mensaje se envía de cliente a servidor. Indica que empiece el proceso de realizar una versión de su carpeta sincronizada. Viene acompañado del nombre o del número de la versión como parámetro.
- **RETRIEVE_VERSION:** Este mensaje se envía de cliente a servidor. Indica el deseo del cliente de recuperar una versión determinada de su carpeta sincronizada. Como parámetro viene el nombre o el número de la versión.
- **WHAT_VERSIONS:** Este mensaje se envía también de cliente a servidor. Es una petición del cliente para conocer qué versiones dispone el servidor de su carpeta sincronizada.
- **VERSION_LIST:** Este mensaje contiene una lista con todos los nombres y numeraciones de las versiones de la carpeta sincronizada.

Mensajes de control de archivos

- **FILE_RETRIEVE:** Mensaje que enviará mayormente el servidor al cliente. Este mensaje indica al otro extremo que desea que le envíe un archivo en concreto.
- **FILE_SEND:** Por definición del proyecto, será el mensaje más utilizado en el protocolo de comunicaciones. Indica que se está enviando un archivo. Como parámetros contiene la ruta del archivo, y el tamaño de éste. Justo después de este mensaje llegarán bytes sin codificar en el socket, y será el archivo adjunto enviado.
- **FILE_RENAME:** Mensaje que indica que el archivo ha sido renombrado. El primer parámetro indica la ruta del antiguo archivo, y el segundo parámetro indicará la nueva ruta.
- **FILE_DELETE:** Mensaje que indica que un archivo ha sido borrado. El parámetro indica la nueva ruta.
- **VERSION_ZIP:** Mensaje similar al FILE_SEND, pero que contendrá un archivo comprimido como adjunto. El cliente deberá descomprimir el archivo y colocarlo en la ruta especificada. Como parámetro incluye el nombre o la numeración de la versión.

3.3.2 Aplicación cliente

3.3.2.1 Comportamiento y entorno gráfico de usuario.

La aplicación cliente constará de dos elementos básicos: un icono de mensajes en la barra de tareas (con el icono de la aplicación), dónde saldrán todos los mensajes de error, y de un formulario principal que se desplegará cuando el usuario haga doble clic en el icono.

En dicho formulario se mostrará un resumen en forma de listado de todas las carpetas sincronizadas que disponemos. El icono de la carpeta será un icono de carpeta plano en caso de que la conexión esté activa y funcionando, y el de una carpeta con una cruz marcada en caso de que la conexión de dicha carpeta no esté conectada o tenga algún error.

Fig 12. Iconos de carpeta sincronizada activa o no activa.

En dicho formulario si el usuario hace doble clic en los iconos de la carpeta, se abrirá una ventana del explorador de Windows directamente en la ubicación de dicha carpeta, para facilitar la localización de la ruta exacta de la carpeta sincronizada.

El usuario podrá desplegar un menú emergente pulsando el botón derecho en todo el formulario. Si el puntero del ratón estaba situado encima de uno de los iconos de las carpetas sincronizadas, el menú también tendrá las opciones de realizar versión, ver versiones, y editar conexión. El menú tendrá las siguientes opciones:

- Añadir conexión
- Editar conexión (deshabilitado si no se está encima de icono de carpeta)
- Versionar carpeta (deshabilitado si no se está encima de icono de carpeta)
- Ver versiones (deshabilitado si no se está encima de icono de carpeta)

Cuando se selecciona la opción de añadir conexión, o editar conexión, aparecerá un formulario dividido en dos partes: la parte izquierda tendrá una lista con las conexiones existentes, dónde las conexiones inactivas saldrán en color rojo. También tendrá un botón para añadir nueva conexión. La parte de la derecha, que se activará al añadir una conexión nueva o editarla, tendrá una zona con las configuraciones: usuario, contraseña, puerto e IP, así como botones de guardar datos, conectar o eliminar conexión.

La opción de ver versiones desplegará un nuevo formulario con una lista con las versiones disponibles de dicha carpeta sincronizada. Al seleccionar una de las versiones, se desplegará un formulario donde se pedirá al usuario en qué carpeta de su unidad de disco duro desea guardar ésta versión. Cuando se seleccione la carpeta, ambos formularios (el de selección de carpeta y visualizar versión) se cerrarán, y aparecerá un mensaje en el icono de la aplicación avisando que la versión se está descargando del servidor, y se abrirá automáticamente la ubicación cuando esté descargada.

La opción de crear versión desplegará un formulario donde habrá un selector de modo de versión: por nombre o por número de versión. Si el usuario selecciona por nombre, aparecerá un campo de texto donde introducir el nombre. Por el contrario, si el usuario selecciona por versión, aparecerán 4 campos de texto pequeños donde poner la numeración de la versión (por ejemplo, 1.0.0.1). Una vez seleccionado el modo de versionar, y aceptarlo, saldrá un mensaje de aviso desde el icono de la aplicación avisando que se está versionando la carpeta. Cuando termine de versionar, aparecerá un mensaje desde el icono avisando de que la versión se ha realizado correctamente y ya se puede visualizar.

El comportamiento interno de la aplicación constará de la creación de la carpeta sincronizada al crear una nueva conexión, y de supervisarla de forma transparente para el usuario. Cada vez que el usuario cree, modifique, renombre o elimine un archivo dentro de esa carpeta, la aplicación comunicará al servidor dicho cambio y será responsabilidad del servidor de tener una copia exacta de la carpeta sincronizada.

3.3.2.2 Diagrama de clases

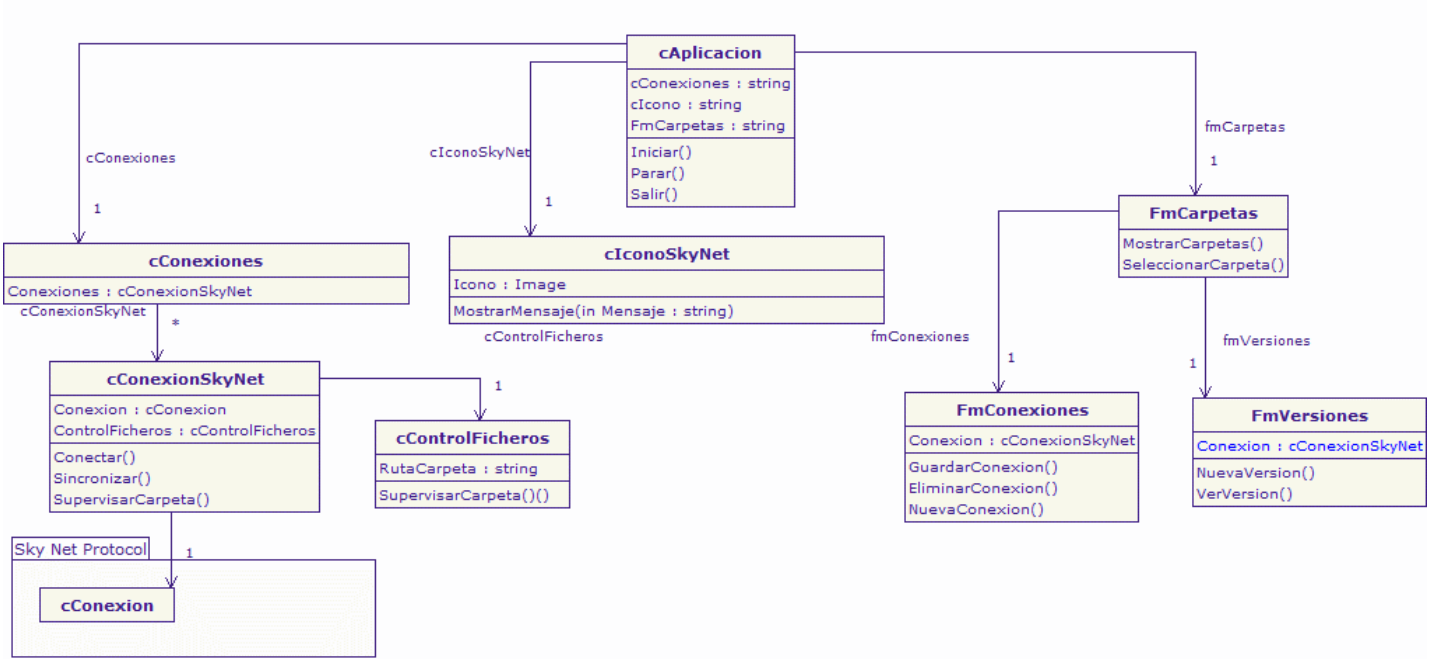


Fig 13. Diagrama de clases de la aplicación cliente.

3.3.2.3 Definición de clases

A continuación definiremos brevemente las clases que compondrán la aplicación cliente y su función:

- **cAplicacion**: Clase principal que se inicia junto a la aplicación. Contiene una instancia de las clases *cConexiones*, *cIconoSkyNet* y *FmCarpetas*, ya que serán únicas en toda la aplicación. Se encargará de iniciar todas las posibles conexiones, o pararlas en caso de que el usuario lo solicite.
- **cConexiones**: Clase que contendrá todas las conexiones activas e inactivas hacia servidores *Sky Net*. Se encargará de administrar todos los eventos que puedan producir las conexiones.
- **cConexionSkyNet**: Clase que representa la implementación de una conexión *SkyNet* del protocolo de comunicaciones. Contiene un objeto de la clase *cConexion* y se encargará de interpretar los mensajes de control recibidos, y enviar los pertinentes.

- **cControlFicheros**: Clase que supervisará la carpeta sincronizada y generará eventos distintos dependiendo del tipo de modificación que se realiza en el sistema de ficheros del ordenador.
- **cIconoSkyNet**: Clase que representa el icono en la barra de tareas de la aplicación. Mostrará los mensajes en un globo de información emergente, y mostrará el formulario de carpetas en caso de hacer doble clic
- **FmCarpetas**: Clase que representa el formulario visible donde se representan las carpetas sincronizadas con diferentes iconos. Tendrá un menú emergente con el que poder acceder a la edición de conexiones y versiones
- **FmVersiones**: Clase que representa el formulario visible donde se pueden crear y solicitar versiones.
- **FmConexiones**: Clase que representa el formulario visible donde se pueden administrar las conexiones activas e inactivas, así como crear nuevas.

3.3.3 Aplicación servidor

3.3.3.1 Comportamiento y entorno gráfico de usuario.

La aplicación servidor se diferencia de la aplicación cliente en gran parte por la interfaz de usuario. En vez de centrar la interfaz de usuario en las carpetas sincronizadas, ésta aplicación se centrará en la administración de usuarios y grupos.

La aplicación se iniciará similarmente a la aplicación cliente. El inicio de la aplicación será silencioso y con un icono en la barra de tareas. Sin embargo, si es la primera vez que se inicia la aplicación en la máquina en la que se encuentra, saldrá un mensaje de aviso a modo informativo que hará saber al usuario que debe configurar una carpeta raíz para empezar a almacenar las carpetas sincronizadas. Ésta carpeta raíz será donde se van a alojar todas las carpetas de todos los clientes que se conecten al servidor, por lo tanto deberá estar en un disco duro amplio y con buena seguridad. Más adelante se podrá cambiar la carpeta.

Al hacer doble clic sobre el icono, aparecerá una pequeña barra de herramientas en una de las esquinas de la pantalla. Dicha barra de herramientas será un menú desplegable que contendrá las siguientes opciones de configuración:

- Configurar carpeta raíz
- Configurar puerto de escucha
- Gestión de clientes
- Gestión de grupos
- Parar / Encender servicio.

La opción de configurar carpeta raíz nos mostrará un formulario de selección de carpeta, dónde el usuario seleccionará dónde quiere situar todo el contenido del servidor de Sky Net. Una vez seleccionada una nueva carpeta, si no es la primera configuración, se le pedirá nuevamente confirmación, avisando que cambiar la carpeta raíz desconectará inmediatamente a todos los clientes conectados y se moverán todos los archivos de una carpeta a otra, lo que comportará lentitud en el sistema, y posibles fallos si se interrumpe el proceso.

Si aun así el usuario decide cambiar la carpeta, se procederá a desconectar a los clientes enviando un mensaje de que el servidor va a proceder con labores de mantenimiento y permanecerá desconectado un intervalo de tiempo moderado, bloquear el proceso de aceptar conexiones de clientes y mover todo el contenido de un sitio a otro. Una vez finalizado el proceso, se vuelven a aceptar conexiones a clientes.

La opción de configurar el puerto de escucha cambiará el puerto por el cual se conectarán los clientes. En caso de guardar un valor distinto al anterior, se desconectarán a todos los clientes conectados con un mensaje de que el servidor ha cambiado la configuración y contacten con su administrador, se bloqueará el proceso de aceptar clientes, se guardarán los cambios, y se volverá a iniciar el proceso de aceptar clientes, esta vez con el puerto nuevo.

La opción de gestionar usuarios mostrará un formulario en el que puede interactuar el usuario administrador. El formulario tendrá dos zonas: una lista de usuarios existentes a la izquierda, y el panel de configuración a la derecha, que se activará solamente si queremos crear un usuario o modificarlo. Dentro del panel de configuración, si el usuario ya ha sido creado, se activará un sub panel con dos listas: grupos disponibles, y grupos a los que pertenece. Se podrán pasar elementos de una lista a otra con un botón central con dibujo de flecha que cambiará la orientación dependiendo de qué lista hayamos seleccionado.

Cualquier cambio producido en usuarios se guardará en la base de datos, en la tabla Usuario. Si el cambio es modificación o eliminación, se desconectará a todos los clientes conectados bajo ese nombre, enviándoles un mensaje de que la configuración de su usuario ha cambiado y contacten con el administrador. Si es un cambio de modificación, se actualizarán los datos en la conveniente base de datos. Si es eliminación, además de eliminar los datos de la tabla, se eliminará la carpeta del servidor.

La opción de gestionar grupos mostrará un formulario con un comportamiento idéntico al de la gestión de usuarios. Se podrán crear, modificar y eliminar grupos. Cuando se modifica un grupo, se desconectarán a todos los usuarios que estén conectados a ese grupo, con un mensaje de notificación. Si se elimina un grupo, además de desconectar a los usuarios y eliminar el registro de la base de datos, se eliminará la carpeta sincronizada.

Cabe destacar que en ambos casos, tanto de usuario como de grupo, si se eliminan se deberá eliminar también cualquier posible relación en la tabla UsuarioGrupo, eliminando las carpetas adjuntas de dichos usuarios o grupos.

La opción de Parar / Encender servicio actuará dependiendo de si el servicio de aceptar conexiones de clientes está encendido o no. En el caso de que esté parado, lo encenderá sin mostrar notificación alguna, de forma silenciosa. En caso de que esté encendido, se mostrará un formulario donde se podrá escribir un mensaje para enviar a todos los clientes. Si no se escribe ningún mensaje, se generará el mensaje de la parada del servicio por labores de mantenimiento por tiempo indefinido.

En la parte del comportamiento interno del servidor, nos encontramos con una estructura similar a la de la aplicación cliente. Controlaremos todas las conexiones de clientes en distintos threads, y recibiremos todos los mensajes que éstos nos hacen llegar.

El procedimiento de identificar a un cliente será el siguiente:

1. Establecer conexión con el cliente
2. Esperar mensaje de login
3. Comparar el login con lo almacenado en la base de datos
4. Pedirle su estructura y estado de la carpeta sincronizada
5. Actualizar la carpeta sincronizada del servidor y la del cliente. La del servidor con los archivos más recientes del cliente, y la del cliente con los archivos más recientes del servidor.
6. Quedarse a la espera de más mensajes de control por parte del cliente.

Todo este procedimiento se realizará con eventos que saltarán desde el módulo de protocolo de comunicaciones.

Cuando se recibe un mensaje de tipo LOGIN, comprobaremos en la base de datos si el login es correcto, y le contestaremos con un mensaje de LOGIN_DENIED, con el primer parámetro en ACCEPT o DENIED según si es correcto o no respectivamente.

Cuando se recibe un mensaje de tipo FILESYNCTABLE, compararemos archivo a archivo con los que tenemos en el servidor. Podemos tener cuatro posibles resultados:

- Si el archivo es distinto y más reciente que el del servidor, programaremos su petición.
- Si el archivo es distinto y menos reciente que el del servidor, programaremos su envío.
- Si el archivo no se encuentra en el servidor, programaremos su petición
- Si el archivo no se encuentra en el cliente, programaremos su envío.

Una vez se hayan comparado todos y cada uno de los archivos, se procederá a enviar mensajes de FILE_SEND (enviar archivo), o FILE_RETRIEVE (pedir archivo) según se haya programado.

Cuando se recibe un mensaje de tipo FILE_SEND, significará que el cliente ha tenido un cambio de archivo (o creación), y nos envía el nuevo archivo. Se deberá cambiar el archivo que tiene el servidor por el que nos está proporcionando el cliente.

Con el mensaje de tipo FILE_RENAME, simplemente se cambiará la antigua ruta del archivo por la nueva.

Si recibimos un mensaje tipo FILE_DELETE, borraremos el archivo en la ruta especificada en el parámetro.

En cualquiera de los anteriores tres mensajes, FILE_SEND, FILE_RENAME y FILE_DELETE, se deberá reenviar el mismo mensaje a todos los clientes conectados bajo ese mismo nombre de usuario o grupo. Cabe destacar que en una conexión a grupo (no a usuario), si no se proporcionó contraseña de administrador, se ignorarán absolutamente todos los cambios que el cliente envíe.

Si se recibe un mensaje de DO_VERSION, se bloquearán todos los threads provenientes de los clientes conectados bajo el mismo nombre, con el fin de que no se produzcan cambios en la carpeta mientras realizamos la versión. La versión se hará comprimiendo un archivo en ZIP y colocando dicho archivo en una subcarpeta llamada “VersionedFiles”, dentro de la carpeta del usuario.

El mensaje WHAT_VERSIONS será respondido con un mensaje de tipo VERSION_LIST, conteniendo una lista de los nombres de todos los archivos ZIP dentro de la subcarpeta “VersionedFiles”.

El mensaje de tipo RETRIEVE_VERSION, será respondido enviando un mensaje de VERSION_ZIP, enviando como dato adjunto el archivo ZIP conteniendo la versión.

Las carpetas de los usuarios se almacenarán siguiendo la siguiente estructura:
CarpetaRaiz/Users/NombreUsuario

Dentro de esa carpeta de usuario, se crearán dos carpetas, “OriginalFiles” y “VersionedFiles”. Los archivos sincronizados con los clientes se almacenarán dentro de OriginalFiles, y los ZIPS que contienen versiones serán almacenados dentro de VersionedFiles.

Las carpetas de los grupos se almacenarán siguiendo la siguiente estructura:
CarpetaRaiz/Groups/NombreGrupo

Dentro de esta carpeta de usuario se crearán dos carpetas mas: “PublicFiles” y “UserFiles”. Dentro de PublicFiles se crearán dos carpetas “OriginalFiles” y “VersionedFiles”, y dentro de la carpeta “UserFiles” se reproducirá exactamente la misma estructura que las carpetas de usuario. De esta forma lograremos tener separados los documentos públicos del grupo de los documentos privados de los usuarios del grupo.

3.3.3.2 Diagrama de clases

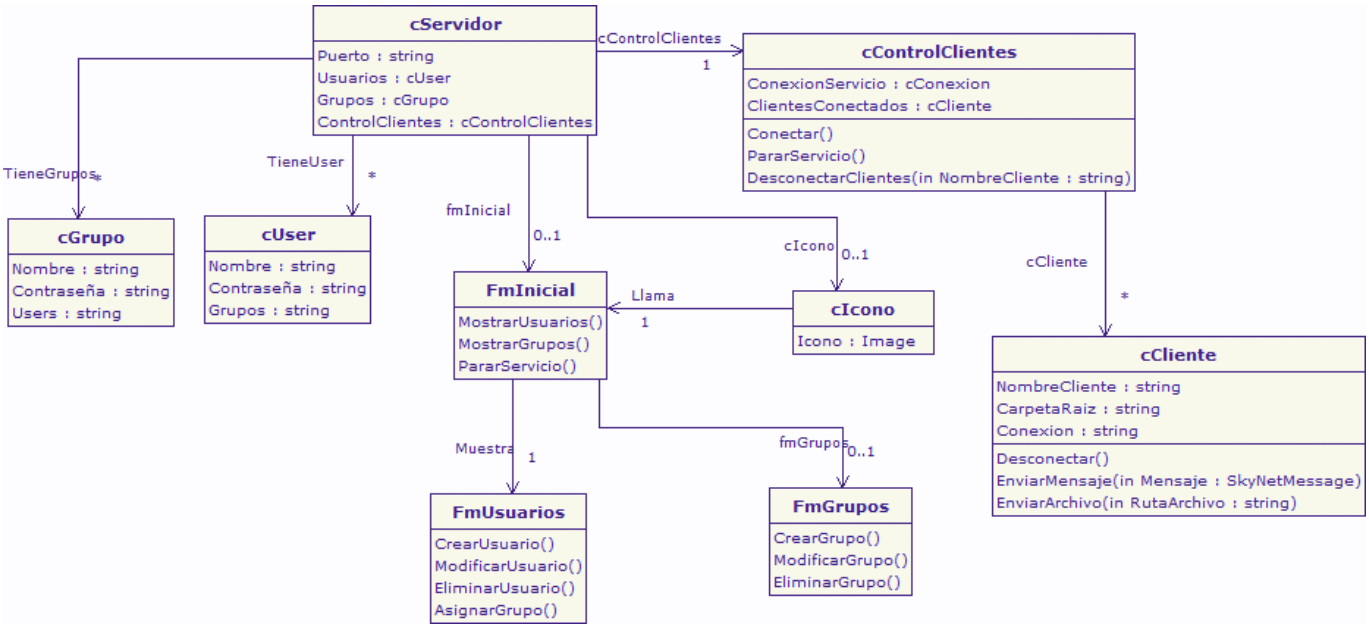


Fig 14. Diagrama de clases de la aplicación servidor.

3.3.3.3 Definición de clases

- **cServidor**: Clase que se inicia junto a la aplicación. Al iniciarse recuperará las configuraciones de usuarios y grupos y las almacenará internamente, para evitar posteriores consultas a la base de datos. Además iniciará el icono en la barra de tareas y el servicio de escuchar conexiones de clientes, siempre que se cumplan los requisitos.
- **cGrupo**: Clase que representa un grupo. Tiene nombre del grupo, su contraseña codificada con MD5, y una lista de los nombres de usuarios que él tiene, para facilitar la posterior búsqueda.
- **cUser**: Clase que representa un usuario. Tiene el nombre del usuario, una contraseña codificada con MD5, y una lista con todos los grupos a los que pertenece, para facilitar la posterior búsqueda.
- **cControlClientes**: Clase que tiene la conexión de servidor activa, y una lista con todos los objetos de cCliente que están conectados. Se encarga de recoger los eventos generados por los cClientes y actuar según sea necesario.
- **cCliente**: Clase que representa la conexión de un cliente con el servidor. Es la que se comunica directamente con el otro extremo. Genera eventos según lleguen mensajes de control, y envía mensajes de control al otro extremo.

- **cIcono**: Clase que representa el icono en la barra de tareas. Responderá ante eventos del ratón por parte del usuario físico, y mostrará el formulario inicial, o lo ocultará.
- **FmInicial:** Formulario visual que se muestra al usuario físico para que pueda gestionar los usuarios, los grupos, y las configuraciones del servidor. Consta simplemente de una barra de menús desplegable para acceder a las distintas opciones.
- **FmUsuarios:** Formulario que tiene una lista de usuarios y opciones para crearlos, modificarlos y borrarlos. Además aquí es donde se asignarán los grupos a los usuarios.
- **FmGrupos:** Formulario que tiene una lista de grupos y opciones para crearlos, modificarlos y borrarlos.

3.3.4 Definición usuario-grupo

El punto fuerte de este proyecto es la aparición de un nuevo tipo de usuarios en esta clase de aplicaciones (programas de sincronización on-line), que son los grupos.

Un grupo es una nueva definición de usuario, que representa al usuario público. Es un tipo de usuario al que podemos conectarnos sin necesidad de contraseña, y podemos visualizar los archivos que existen en esta carpeta sincronizada. Sin embargo, la diferencia de un usuario normal, es que cualquier modificación realizada en la carpeta sincronizada no va a afectar al contenido del servidor ni de los demás clientes conectados. Únicamente si se ha introducido contraseña de administrador, y ésta es correcta, funcionará como si fuera un usuario normal (guardando cambios).

Además se introduce la posibilidad de asignar un usuario a un grupo. Lo que aporta esto es que un usuario pueda tener su propia carpeta sincronizada al completo (guardando las modificaciones), utilizando su misma contraseña que la de su usuario. La aportación de ésta posibilidad, es que los usuarios con contraseña administradora podrán interactuar con los usuarios de estos grupos, modificando sus archivos. Los demás usuarios que estén conectados al grupo sin contraseña en ningún caso podrán visualizar los archivos de los usuarios del grupo.

De ésta forma conseguimos una carpeta colaborativa entre administrador-usuario, además de seguir teniendo sincronizados los archivos de la carpeta pública. Una conexión a grupo público será distinta a la de una conexión de usuario de grupo, por lo tanto, en la aplicación cliente se deberán configurar dos conexiones distintas (una para grupo, otra para usuario de grupo) si se quieren utilizar ambas ventajas.

Algunos ejemplos de utilidad de este sistema sería el de un campus virtual universitario. Los alumnos podrían conectarse con su usuario a un grupo (que sería una asignatura), y colgar ahí los trabajos. Paralelamente podrían recuperar la documentación de la asignatura con una conexión a un grupo público.

Otra aplicación al sistema sería una carpeta dedicada a la sincronización de sub proyectos de un proyecto mayor (por ejemplo, módulos de un programa). Cada carpeta de usuario de grupo representaría un módulo del programa (o sector de la empresa), y la carpeta pública sería donde se cuelga la documentación. Los administradores del grupo se encargarían de realizar comentarios y modificaciones en los diferentes módulos.

Como podemos ver, las posibilidades y la flexibilidad son mucho mayores que un programa de sincronización de documentos on-line sencillo.

CAPÍTULO 4

IMPLEMENTACIÓN

4. Implementación

4.1. Introducción

En este apartado hablaremos de la fase de implementación del proyecto, empezando por las tecnologías y librerías utilizadas para ello, y los problemas surgidos durante ésta fase y de cómo se han solucionado.

4.2.1 Tecnologías y librerías

Lenguajes de programación

- Visual Basic.NET 2010

Tecnologías de plataforma

- Microsoft .NET Framework 4.0

Gestores de bases de datos

- SQLite3, versión ADO.NET 4.0 (Adaptado a Visual Studio.NET 2010)

Protocolos utilizados

- Socket TCP/IP

Librerías estándares utilizadas

- Codificación MD5
- Codificación SHA1

4.2.1.1 Visual Basic.NET 2010

El lenguaje Visual Basic.Net es un lenguaje creado y diseñado íntegramente por Microsoft para ser el sucesor del antiguo Visual Basic. Es un lenguaje totalmente orientado a objetos y funciona sobre la plataforma de desarrollo .NET Framework. El lenguaje está orientado a crear aplicaciones de gestión y visuales, utilizando ventanas de Windows, aunque también se pueden crear fácilmente aplicaciones en modo consola, y más tipos de aplicaciones especificados dentro del .NET Framework.

La versión utilizada del Visual Basic.NET es la 2010, incluida dentro del Microsoft Visual Studio 2010, que a su vez utiliza el .NET Framework 4.0

Características:

- Lenguaje intuitivo y fácil de utilizar. Curva de aprendizaje corta.
- Lenguaje totalmente orientado a objetos.
- Sintaxis intuitiva y parecida al lenguaje normal hablado.
- Compilador y debugador completos y fáciles de aprender a utilizar.

Ventajas:

- Utiliza toda la potencia del .NET Framework 4.0, que brinda al lenguaje un conjunto de librerías capaces de realizar un extenso abanico de funciones.
- Máxima integración con el sistema operativo Microsoft Windows: El lenguaje ha sido diseñado para tratar con variables del sistema de forma fácil.
- Facilidad a la hora de debugar una aplicación y encontrar errores.
- Modo “Inmediato”: Acceso inmediato a variables en proceso de ejecución y poder cambiar los valores para analizar el comportamiento del programa.

Inconvenientes:

- Únicamente disponible (de forma oficial) para los sistemas operativos Microsoft Windows (XP SP3, Vista, 7).
- En fase de diseño y modo debugación, los recursos consumidos son de la orden de una veintena más que la simple ejecución del programa. Por ejemplo, el proyecto ejecutado consume aproximadamente 20 Mb de memoria RAM, mientras que en modo debugación consume alrededor de 400-500 Mb.
- Interfaz de diseño pesada, requiere de un ordenador con muchos recursos.

4.2.1.2 Microsoft .NET Framework 4.0

El Microsoft .NET Framework 4.0 es un conjunto muy potente de librerías, totalmente gratuito, que se incorpora al sistema operativo (muchas versiones ya lo traen preinstalado). Aporta un abanico de funciones, clases definidas, y control de recursos muy amplio y fácil de utilizar. Además, todas estas librerías están realizadas en formato estándar de Microsoft, lo cual significa que se pueden utilizar siempre en cualquier lenguaje que utilice el .NET Framework.

Gracias a esto, una aplicación realizada en un lenguaje de programación (por ejemplo, Visual Basic.NET), puede llegar a recompilarse en otro lenguaje dentro del .NET Framework (por ejemplo, C#). De ésta forma se pueden llegar a crear aplicaciones y/o librerías que pueden ser reutilizadas de forma independiente al lenguaje en el que se programen.

Características:

- Conjunto muy extenso de librerías que aportan funcionalidades y clases definidas a los lenguajes de programación
- Ciclos de actualización moderados: Microsoft actualiza con la frecuencia justa para que los programadores se adapten a los cambios.
- Totalmente gratuito.

Ventajas:

- Permite crear librerías multilenguaje: las DLL creadas en un lenguaje servirán para otros.
- Las aplicaciones creadas utilizando .NET Framework serán siempre compatibles con sistemas operativos que puedan utilizar esa versión de .NET Framework.
- Documentación y ayuda altamente extensa, desde página web con búsqueda inteligente (buscar lo que quieres hacer, y te lleva al método y/o clase que lo realiza), hasta foros con programadores en línea que te ayudan.

Inconvenientes:

- Únicamente disponible (de forma oficial) para los sistemas operativos Microsoft Windows (XP SP3, Vista, 7).
- En la primera carga de cualquier programa que utilice .NET Framework, se cargarán todas las librerías utilizadas, lo que causa ralentización del sistema.

4.2.1.3 SQLite 3 (ADO.NET 4.0)

SQLite es un proyecto de dominio público (Open Source) realizado por *D. Richard Hipp* que consta de una simple librería (alrededor de 200 Kb) que nos permite utilizar un completo sistema gestor de bases de datos relacional. La versión utilizada es la 3 adaptada al .NET FRAMEWORK 4.0.

A diferencia de otros motores de gestión de bases de datos, éste simplemente utiliza lo básico y fundamental para ello: crear tablas, gestionarlas, y manejar datos. No incluye sistemas de control, y no tiene soporte para campos extremadamente grandes. Sin embargo, no necesita de instalación alguna en la máquina que ejecuta la aplicación.

Características:

- Sistema gestor de bases de datos relacional completo.
- Librería de poco tamaño
- Última versión compatible con bases de datos de hasta 2 Terabytes.

Ventajas:

- No requiere de instalación
- No requiere de configuración
- Gestión de datos liviano y rápido.

Inconvenientes:

- Pocos tipos de datos soportados: Texto, numérico, BLOB (Datos binarios) y INTEGER PRIMARY KEY, que es un tipo de datos numérico auto incremental que identifica las filas.
- Pocas actualizaciones.
- Sistema poco seguro. El cifrado de datos se realiza sobre el fichero que contiene la base de datos, y ese fichero puede ser fácilmente modificado.

4.2.1.4 Socket TCP/IP

TCP/IP es un conjunto de protocolos definidos en los que se basa Internet, y se utilizada para la intercomunicación de computadoras. Se basa en la unión de dos de los protocolos básicos para las comunicaciones en internet: el TCP y el IP. Se utiliza este protocolo en el diseño de aplicaciones cuando se necesita de seguridad y flexibilidad en la entrega de datos. A diferencia de otros protocolos, éste se asegurará siempre de que los datos lleguen de un extremo a otro (a excepción de problemas físicos de la línea, o de la aplicación).

La forma de utilizar el protocolo TCP/IP en el proyecto se hará mediante el uso de la definición *Socket.* Un socket es un concepto abstracto que implica la interconexión de dos programas, y el intercambio de flujos de datos de un extremo a otro, de forma bidireccional. Para ello se necesita un par de direcciones IP (origen y destino), un protocolo de transporte (en nuestro caso, TCP), y un par de puertos (origen y destino).

Características:

- Fiabilidad al enviar datos: el protocolo TCP /IP brinda una fiabilidad casi total de que los datos lleguen bien y en orden.
- Diferenciación de la capa de la lógica del programa con la de comunicaciones, gracias al Socket.

Ventajas:

- No se necesita hacer la implementación del protocolo entero en nuestro programa, simplemente se importa la librería necesaria
- Protocolo y librería creadas y mantenidas por un organismo que se encarga de las comunicaciones en internet.

Inconvenientes:

- Los mecanismos internos de control para asegurarse de la fiabilidad que nos aporta son pesados: los paquetes de datos enviados son más grandes y, por lo tanto, la conexión es menos rápida que si utilizáramos un protocolo sin esos controles.
- No codifica los datos. Hay que hacerlo en la aplicación.

4.2.1.5 Generadores de hash MD5 y SHA1

MD5 y SHA1 son funciones generadoras de hash. El hash es una cadena hexadecimal devuelta por el algoritmo de la función y en un principio (sobre todo en SHA1, puesto que en MD5 se han encontrado fallos), un hash de un archivo o de contraseña es único y diferente al de los demás. Para generar el hash ambos algoritmos utilizan el contenido de dicho archivo o contraseña, es por eso que si el archivo cambia, o la contraseña, el hash cambiará. La generación de hash no es reversible, no se pueden obtener los datos iniciales con el hash.

Ambos sistemas pueden utilizar una semilla. Una semilla es una palabra, o valor, que utilizará el algoritmo de codificación para utilizarlo de base a sus operaciones matemáticas internas. De ésta forma, si emisor y receptor conocen la semilla, pueden comparar la codificación. Comparándolo con otro término informático, la semilla es la contraseña de la generación de hash.

La utilización en éste proyecto es la comparación del hash, tanto de contraseñas como de archivos. Utilizando este método, podemos distinguir si un archivo con los mismos atributos y propiedades es diferente a otro, o si las contraseñas proporcionadas son las mismas. SHA1 es más seguro que MD5, pero es más pesado y costoso de calcular. Solamente se ha implementado como alternativa en caso de que falle el MD5 (más adelante explicado en un juego de pruebas)

Características:

- Creación de hash supuestamente único.
- Alta seguridad al cifrar datos y enviarlos.

Ventajas:

- Identificación de archivos rápida y eficaz
- Comparación entre hash de contraseñas rápida y eficaz.

Inconvenientes:

- En archivos grandes, la generación del hash puede ser muy costosa, de la orden de segundos.
- En archivos muy pequeños, se podría llegar a encontrar un hash MD5 idéntico.

4.2.2 Desarrollo y variaciones del diseño en fase de implementación

En este apartado explicaremos el desarrollo de la aplicación mediante el encuentro de problemas y obstáculos encontrados en la fase de implementación, así como la solución a ellos.

4.2.2.1. Threads en Visual Basic.NET 2010

Uno de los apartados más problemáticos en todo el proyecto han sido como utilizar los threads en el lenguaje Visual Basic.NET 2010, para la emisión y recepción de mensajes de Sky Net y pasarlos a la clase superior que controla las conexiones. A pesar de que el concepto es idéntico en la gran mayoría de lenguajes orientados a objetos, una serie de características han obligado a que el proyecto se ejecute con unos algoritmos diseñados para solventar algunos problemas que aparecen con la implementación de threads que nos ofrece el .NET Framework 4.0

El problema principal de un thread en Visual Basic.NET 2010 es que pierde la capacidad de comunicación con el thread principal desde el que se llamó. A diferencia de otros lenguajes, cuando creas un hilo de ejecución, siempre puedes referenciar al proceso que creó tal hilo. Sin embargo, en éste lenguaje saltan excepciones al intentar comunicarse. Para la utilización de threads hemos utilizado dos tipos de diseño de algoritmo distintos:

- **Semáforos y colas.**

  El primer paso para poder utilizar este sistema es crear una cola y un evento. La cola es una colección de objetos con tecnología FIFO (First In, First Out). De ésta forma, cada vez que un thread pase por la cola, dejará su mensaje de Sky net en la cola. Y se irán dejando según el orden de ejecución de dichos threads.

  Para controlar que únicamente un thread puede colocar a la vez el mensaje en la cola, se utilizará un semáforo que solamente dejará pasar un thread a la vez, dejará el mensaje en la cola, y saldrá del semáforo indicando que puede pasar el siguiente. De esta forma la cola únicamente podrá ser accedida y/o modificada por un solo thread a la vez. De ésta forma luego podemos recoger toda la cola de mensajes (utilizando el mismo semáforo), y se garantiza que ningún thread se pise entre sí, y

  los threads no interactúan directamente con el thread principal: interactúa la cola.

- **Threads de segundo plano**

  Independientemente a los threads implementados en Visual Basic .NET 2010, existe otra clase llamada “trabajador de segundo plano” (Background Worker). Ésta clase incorpora una serie de métodos y eventos para realizar un trabajo en segundo plano en un thread distinto, dentro de una misma clase.

  La implementación es muy sencilla: basta con programar el algoritmo dentro del método de dicha clase, y automáticamente lo realizará en segundo plano. A diferencia de los threads normales, ésta clase puede interactuar con los atributos y métodos del objeto en el que se ha incorporado. De ésta forma, podemos crear un hilo de ejecución que se queda observando el socket de entrada dentro del propio algoritmo, sin necesidad de separar código. Así conseguimos utilizar los atributos y métodos privados[9] de la clase.

  En nuestro proyecto, los threads en segundo plano han sido utilizados para el diseño y la implementación del thread principal de recepción de mensajes Sky Net. Por otro lado, los semáforos y las colas han sido utilizados mayormente para controlar el flujo de salida de mensajes Sky Net. A diferencia de los eventos normales, que siguen el flujo de ejecución de un programa normal, los eventos generados por el sistema operativo (sistema de archivos) son threads en paralelo. Esto quiere decir que si un programa externo hace muchas modificaciones a la vez en la carpeta sincronizada, el sistema operativo puede llegar a lanzar un conjunto de eventos, todos ellos threads, con todos los cambios. Se utiliza el semáforo para filtrar estos eventos, y la cola para guardarlos en el orden que va llegando.

4.2.2.2. Duplicidad de archivos

Un bug conocido del sistema es que el sistema operativo puede llegar a lanzar dos, o más threads de eventos a la vez, cuando se realiza un cambio, y puede llegar antes un evento que otro. En estos casos, Sky Net no será capaz de distinguir cuál de los eventos fue realmente creado al principio, e ignorará y no enviará al servidor los eventos destructivos (renombrar y borrar) en caso de que al intentar acceder a dicho archivo, ya no exista. Esto causará la posible duplicidad de archivos en el servidor, o que archivos anteriormente eliminados en el cliente aparezcan de nuevo.

[9] Atributos y métodos privados: Cualquier declaración de atributo o método en modo privado significará que únicamente se puede acceder a él desde la misma clase. Otra clase de fuera no podrá ver o utilizar dichos atributos o métodos.

Para solventar este problema, se diseñaron tres sistemas distintos de detección de archivos:

- El primer sistema se basaba en un sistema de repetición cada X tiempo (la prueba se realizó con 10 segundos de intervalo) de la tabla de situación de la carpeta sincronizada. Éste diseña es muy efectivo y evita totalmente la duplicidad de archivos, y mantiene una carpeta completamente sincronizada con el servidor. Sin embargo, conforme la carpeta sincronizada va creciendo, ésta tabla cada vez es más pesada, y el cálculo de hash de archivos grandes se hace tediosa para el procesador. Por lo tanto, para carpetas grandes éste sistema no es efectivo. Además perdemos la distinción del tipo de cambio que se ha generado (sobretodo renombrar), y por lo tanto, siempre estaremos mandando archivos que podrían ser el mismo.

- El segundo sistema es una modificación del primero. La tabla de situación de la carpeta sincronizada no incluye el cálculo de hash, y por cada subcarpeta dentro de la carpeta sincronizada se crea un thread que a su vez recorrerá las subcarpetas. Con éste sistema, en procesadores rápidos la creación de la tabla de situación se genera muy rápidamente, y en caso de que el servidor dude de si un archivo es el mismo o no, se pedirá la generación del hash. Éste sistema es efectivo, pero en el momento que hay muchos clientes conectados bajo un mismo sistema, el que se satura es el servidor: Intentar replicar la tabla de estado a todos los clientes y servirles todos los archivos y cambios.

- El tercer sistema, el actual, es la mezcla del sistema original y el primer sistema presentado. Los cambios se seguirán almacenando vía eventos del sistema operativo (crear, modificar, renombrar, borrar), y sirviendo los archivos sólo en caso de ser necesario. Sin embargo, cada vez que el servidor detecte alguna operación ilógica (renombrar un archivo inexistente, por ejemplo), activará una variable en la conexión para pedir la tabla de estado al cliente. Existirá un ciclo de actualización de cliente preestablecido en 60 segundos, y que solamente se activará si ambos sockets (lectura y escritura) están inactivos. Cada vez que pase el tiempo del ciclo de actualización se pedirá al cliente la tabla, y se estructurarán los archivos de nuevo.

  Los ciclos de actualización simplemente será un temporizador programado para saltar un evento cada X segundos. Solamente se creará un temporizador en el caso que tengamos la primera incoherencia en cuanto a eventos, y el temporizador se irá autoajustando dependiendo de la cantidad de incoherencias nos vayan llegando. Por cada incoherencia recibida, se reducirá en 5 segundos el siguiente ciclo de actualización, y cada vez que tengamos 5 mensajes coherentes de archivos, se ampliará en 5 segundos el ciclo de actualización.

En el caso que el ciclo de actualización llegue a 20 segundos, se informará al cliente que su sistema operativo está funcionando incorrectamente, y se le desconectará preventivamente para evitar colapsar la línea de comunicaciones. Si el temporizador llega a 300 segundos, se parará definitivamente, hasta que aparezca una nueva incoherencia, que iniciará todo el proceso de nuevo a 60 segundos.

En el caso de que haya más de un cliente conectado al mismo usuario, los ciclos de actualización se alternarán, en ningún caso se producirán al mismo tiempo. De ésta forma reducimos al mínimo la probabilidad de error de almacenar algún archivo. Actualmente éste sistema está implementado pero no activado, debido a que requiere que el servidor disponga de suficiente ancho de banda para el control de errores, y las pruebas han sido realizadas en una conexión básica de 1 Mbit/seg. (Más adelante explicado)

4.2.2.3. Clases auxiliares.

Estas clases son una ampliación del diseño inicial, y se han creado exclusivamente para facilitar la tarea de la programación y para no sobrecargar clases con métodos que no son propiamente de ellas. Por ejemplo, el cifrado de una contraseña o texto se encuentra fuera de las clases de control de conexiones, ya que no son propiamente de ellas. Éstas clases serán todas globales y estáticas: son clases que se pueden utilizar sin ser instanciadas en el código.

- **SkyNetAux**: Clase que incorpora los métodos de cifrado de datos (contraseñas), la transformación y recuperación de los mensajes Sky Net en cadenas de bytes, y el cálculo de hash de los archivos.
- **cFolderTable**: Clase que incorpora los métodos para crear una tabla de correspondencia de los archivos. Consiste en analizar la estructura de una carpeta que se pasa como parámetro. Cuando llega al otro extremo, se llama al método de sincronizar, pasándole la nueva carpeta como parámetro, y la misma clase generará dos listados: una con archivos que tiene un extremo, y la otra con los archivos que tiene el otro. De esta forma la misma clase se encarga de comprobar los cambios.
- **SkyNetFileConnection**: Clase que se encargará de leer del socket de lectura bloques de 1kb e ir construyendo el archivo en la ruta temporal establecida. De ésta forma sacamos el algoritmo de la clase principal de SkyNetConnection, y podemos controlarlo de forma separada. Se utilizará el mismo socket de lectura que la conexión.

- **cConexionSQL**: Clase que creará una conexión activa con una base de datos de SQLite y podrá modificar y guardar datos en la base de datos del programa. Contiene métodos para conectar, hacer selecciones masivas, selecciones unitarias (por ejemplo, recuperar una contraseña), introducir datos y eliminarlos.

4.2.3.Iteraciones (versiones)

Como ya se explicó en el punto 1.5 (metodología de desarrollo), se ha utilizado un sistema de desarrollo evolutivo para la realización del proyecto. Concretamente se han creado 3 versiones distintas a lo largo del desarrollo, para llegar al diseño y la implementación final. A continuación explicamos de qué constaba cada una de las versiones distintas, y el motivo de dichas versiones.

4.2.3.1. Iteración 1 (Versiones de 0.1-Alpha1 a 0.4-Alpha2)

Se nombró a la primera versión Alpha 1 ya que ésta versión no incluía más que un 20% de las funcionalidades diseñadas inicialmente. El objetivo de ésta versión inicial era el de conseguir la comunicación entre cliente y servidor, sin distinguir todavía los diferentes cambios de archivos. El protocolo de comunicaciones fue implementado con un sistema de confirmación de recepción de mensajes bidireccional: cada vez que una de las dos partes recibía un mensaje se comunicaba al otro que había recibido el mensaje. Esto no formaría parte del diseño final, ya que la definición de Socket del .NET FRAMEWORK 4.0 aseguraba la recepción íntegra del mensaje. Se hizo para la trazabilidad de los mensajes y poderlos visualizar y rastrear en caso de error.

Ésta primera versión carecía de interfaz gráfica cómoda para el usuario, y no disponía de base de datos alguna, todos los datos estaban guardados en la XML interna de las aplicaciones, y el sistema solamente detectaba cambios de creación y modificación de archivos. Además, la transferencia de archivos se realizaba por otro puerto distinto al de las conexiones de mensajes de control, con el fin de detectar mejor los fallos al recibir / enviar ficheros.

Por lo tanto, esta primera versión no incluía gran parte de las funcionalidades diseñadas con el objetivo de encontrar errores en la fase de comunicación entre cliente y servidor, y solventarlos antes de proceder a la implementación de las funciones más complejas.

4.2.3.2. Iteración 2 (Versiones de 0.5-Beta1 a 0.9-Beta2)

Inmediatamente después de la versión Alpha 1 se arregló la interfaz de usuario y se incorporaron las detecciones de los demás tipos de cambios. Además se mejoró el algoritmo de sincronización de archivos entre clientes, por lo que la aplicación servidor empezó a ir más fluida y sin tanta carga de recursos. Se quitó la confirmación de mensaje por parte de cliente y servidor, ya que el protocolo de comunicaciones funcionaba bien, los archivos se enviaban por el mismo puerto de comunicaciones, y se incorporó al servidor los grupos y los usuarios de grupos.

Sin embargo, ésta versión aun no tenía una detección de errores efectiva, y tampoco una corrección de errores en tiempo de ejecución. Se corrompían archivos en caso de desconexión por parte de cliente y servidor, y era difícil corregirlo. Se incorporó la solución de las repeticiones continuas de la tabla de situación de la carpeta sincronizada, y se solventó, pero la carga de tráfico de red perjudicaba a la parte del servidor.

La aplicación se instaló en dos empresas así como varios particulares que se prestaron a la depuración de errores, junto con un archivo de logs que enviaban diariamente, consiguiendo localizar los errores en la detección de cambios en archivos y corregirlos.

4.2.3.3. Iteración 3 (Versión 1.0)

Se incorpora el control de versiones a la parte cliente y servidor. Se realiza el diseño final de la interfaz de usuario por la parte del cliente, unificando todos los formularios en solo uno (el formulario de las carpetas sincronizadas), y desde ahí funciona como central para los demás formularios.

Se implementan los mensajes de control del control de versiones al protocolo de comunicaciones, y se implementan los ciclos de actualización y control en la parte servidor para corregir errores. Se mejora la transferencia de archivos. Se incorpora el uso de la base de datos SQLite para almacenar los datos, y se utiliza la codificación MD5 para almacenar datos críticos (contraseñas).

Ésta versión es la primera versión estable sin bugs críticos y que cumple con todos los requisitos y funciones diseñados.

CAPÍTULO 5

JUEGOS DE PRUEBAS

5. Juegos de pruebas

5.1 Introducción

En este apartado haremos una descripción de los distintos juegos de pruebas que hemos realizado a lo largo del proceso de implementación de Sky Net, así como de sus resultados y de cómo hemos resuelto los posibles errores.

5.2 Pruebas de la comunicación cliente-servidor

Uno de los principales pilares del proyecto es la perfecta comunicación entre el cliente y el servidor. Sin la seguridad de ésta comunicación no podemos garantizar el perfecto funcionamiento del resto de funciones que tenemos en Sky Net. Para ello, durante la primera iteración del proyecto (las primeras fases), se realizó una trazabilidad de todos y cada uno de los mensajes. Se programó un tipo de mensaje adicional (llamado ACK) que se enviaría continuamente de un extremo a otro indicando que un mensaje ha llegado correctamente y se ha podido interpretar bien. Cada vez que un extremo recibía el mensaje ACK, escribía en un archivo de registro el mensaje enviado y que se había recibido correctamente por parte del otro extremo.

El juego de pruebas para someterlo a una posible saturación de mensajes consistió en copiar y pegar 340 archivos pdf de 200 Kb cada uno de golpe en la carpeta sincronizada. En las primeras fases de la versión más de la mitad de mensajes se perdían, o el sincronismo entre cliente-servidor se detenía y no se podía recuperar. El motivo se descubrió gracias a la documentación MSDN acerca de los eventos generados en el sistema operativo. Cuando el sistema operativo genera un evento de lectura/escritura de archivos, estos eventos funcionan como threads aparte. Por lo tanto, en la primera implementación no se controlaba esto, y se perdían los eventos mientras el socket de escritura estaba ocupado.

Para solucionar este error, se diseñó la cola de eventos y los semáforos, explicado en el apartado 4.2.2.1. Una vez se solucionó éste error, los mensajes llegaban todos sin excepción (se realizaron pruebas durante varios días copiando archivos de forma masiva), y se eliminó el mensaje de tipo ACK del protocolo de comunicaciones.

5.3 Pruebas de estrés

Otro de los problemas que se encontró en el sistema Sky net es la alta demanda de procesamiento para calcular los hash de los archivos. Se detectó el problema en una ocasión al sincronizar una carpeta de más de 3000 archivos, todos de más de 1 Mb de tamaño. La creación de la tabla de estado tardó 13 segundos.

Ficheros carpeta	Tamaño ficheros	Tamaño total	Tiempo de creación de hash
1.000	1 Mb	1.000 Mb	5 segundos
3.000	1 Mb	3.000 Mb	13 segundos
10.000	5 Mb	50.000 Mb (*)	345 segundos
20	3 Gb	60 Gb	26 min.

Tabla 9. Resultados de las pruebas de stress

Como podemos observar en la taba, contra más grandes son los ficheros, más grandes son los tiempos de cálculo. Esto es debido a que en la primera versión de codificación de hash, se utilizaba totalmente el contenido del fichero. Este tiempo era el que tardaba la aplicación cliente en enviar la tabla de estado de la carpeta sincronizada al servidor. Una vez eso, se procedía a la sincronización. Evidentemente las pruebas se realizaron bajo un entorno hostil para el cual Sky Net no ha sido pensado: manejo de muchísima cantidad de archivos de gran tamaño. Aún así, una situación en la que una carpeta de usuario ocupe de 1 a 5 Gb podría darse con facilidad, así que no era viable esta solución.

Después de mucha investigación, se logró descubrir que el .NET FRAMEWORK 4.0 incorpora unas librerías específicas de codificación MD5 y SHA1 para ficheros grandes. Una vez readaptado el código utilizando las librerías, el tiempo de cálculo se redujo drásticamente, y la tabla quedó de la siguiente forma:

Ficheros carpeta	Tamaño ficheros	Tamaño total	Tiempo de creación de hash
1.000	1 Mb	1.000 Mb	0.02 segundos
3.000	1 Mb	3.000 Mb	0.15 segundos
10.000	5 Mb	50.000 Mb (*)	2.13 segundos
20	3 Gb	60 Gb	21.6 segundos

Tabla 10. Resultados de las pruebas de stress con nuevas librerías

A pesar de que aún tenemos un resultado alarmante (20 ficheros de 3 Gb calculados en 21.6 segundos), los resultados ya son muy optimizados en comparación a la primera versión. Para paliar esto, se implementó la opción de únicamente pedir el hash en caso de no poder identificar el archivo, reduciendo el cálculo al mínimo.

5.4 Pruebas de sincronización en banda estrecha

Una de las pruebas realizadas para poder orientar los requisitos de conexión a internet en la aplicación fue la de limitar a 64 Kbps la conexión entrante y saliente del servidor de pruebas, e ir aumentándola en potencia de 2 hasta averiguar qué conexión es la más óptima. Para las pruebas se utilizaron carpetas sincronizadas de 10 archivos de 1 Mb cada uno (en total 10 Mb). Los valores los recogemos en la siguiente tabla:

Usuarios conectados	Usuarios simultáneos(*)	Conexión internet	Tiempo de sincronización
1	1	64 Kbps	200 segundos
1	1	128 Kbps	105 segundos
1	1	256 Kbps	60 segundos
1	1	512 Kbps	40 segundos
1	1	1 Mbps	35 segundos
1	1	2 Mbps	12 segundos
1	1	4 Mbps	7 segundos
5	1	64 Kbps	4 minutos
5	1	128 Kbps	3 minutos
5	1	256 Kbps	3 minutos
5	1	512 Kbps	3 minutos
5	1	1 Mbps	1 minuto
5	1	2 Mbps	40 segundos
5	1	4 Mbps	15 segundos
5	5	64 Kbps	10 minutos
5	5	128 Kbps	8 minutos
5	5	256 Kbps	7-8 minutos
5	5	512 Kbps	5 minutos
5	5	1 Mbps	2 minutos
5	5	2 Mbps	40 segundos
5	5	4 Mbps	20 segundos
5	5	Conexión local (LAN) – 1Gbps	Instantáneo
10	10	Conexión local (LAN) – 1Gbps	5 segundos
20	20	Conexión local (LAN) – 1Gbps	5 segundos

(*) nº De usuarios conectados bajo una misma cuenta de usuario.

Tabla 11. Tiempo en tardar en sincronizar clientes, dependiendo de conexión.

De la tabla podemos deducir que los requisitos mínimos para funcionar fluidamente a través de internet son de una conexión mínima de 4 Mbps, y que en conexión local (por ejemplo, ámbito empresarial), la sincronización prácticamente se verá afectada solamente por la velocidad de escritura y la saturación de la red en sí.

5.5 Pruebas de CVS

Para realizar las pruebas de que el control de versiones funciona correctamente, se programó una petición de generar versión en un cliente que tenía una carpeta sincronizada de 100 Mb. Acto seguido se programaron en 4 máquinas distintas un script que durante la creación de la versión mandarían eventos de creación, modificación, renombrar y eliminar sobre esa misma carpeta sincronizada.

Las pruebas obtuvieron como resultado que el servidor bloqueaba todos los eventos enviados por los clientes, e iba almacenando los nuevos archivos en archivos temporales. Una vez terminaba de comprimir el archivo ZIP, atendía a todos los eventos conforme le iban llegando. El resultado fue que muchos archivos fueron eliminados y renombrados aleatoriamente, ya que esos eventos llegaban en orden aleatorio según el socket de lectura los iba enviando.

Para solucionar esto, se tuvieron en cuenta dos posibles soluciones: desconectar a todos los clientes mientras se produjera la generación de versión, o que el servidor siguiera atendiendo a peticiones y procesándolas mientras se generaba la versión. Se optó por la segunda opción con una puntualización. En el momento que el servidor recibe la petición de realizar versión, se genera él mismo una tabla de estado de carpeta sincronizada, y se basará en el listado de archivos de esa tabla para generar la versión. Si mientras se genera la versión un archivo es modificado después de haber sido comprimido, en el archivo comprimido constará el archivo. Si se intenta modificar mientras se está comprimiendo, dará error de escritura, y se ignorará el evento. Y si el archivo es modificado antes de ser comprimido, esa versión tendrá el archivo modificado.

No obstante, se enviará un mensaje a todos y cada uno de los clientes de que se ha realizado una petición de versión de la carpeta sincronizada, pidiéndoles que dejen de trabajar durante un momento. La realización de versión suele ser ligera y rápida (a excepción de carpetas sumamente grandes y pesadas).

CAPÍTULO 6

CONCLUSIONES

6. Conclusiones

6.1 Consecución de objetivos

El objetivo principal del proyecto, como se ha comentado al principio de la memoria, era la creación y el desarrollo de una aplicación de sincronización de archivos de forma on-line, sobre un servidor conectado a internet.

Se puede decir, en la finalización del proyecto, que el objetivo se ha cumplido completamente. Los usuarios son capaces de tener varias carpetas sincronizadas con varios servidores a la vez y recuperar los datos en todo momento, con una fluidez y transparencias muy optimizadas. Así pues, la aplicación es completamente funcional y, salvo algunos errores mínimos que se puedan encontrar, es completamente fiable.

¿Cómo se ha conseguido este objetivo?

Mediante la detección de los eventos del sistema operativo de creación, modificación, renombrar y eliminación de los archivos dentro de una carpeta predeterminada por el programa. Utilizando esos eventos somos capaces de comunicar al servidor qué cambios hemos realizado en la carpeta, y así se obtiene una réplica exacta en tiempo real en un servidor.

¿Qué funciones son necesarias para controlar estos eventos?

Es necesaria una corrección de errores en tiempo de ejecución, ya que los eventos del sistema operativo se pueden acumular y llegar de forma desordenada. Con este control de errores, seremos capaces de determinar, por ejemplo, si un comando de renombrar llega tardío e ignorarlo en vez de intentar renombrar un archivo ya eliminado. También podremos reiniciar la sincronización en cualquier momento en caso de detección de incoherencias en el sistema de archivo. En cualquier caso, si se duda de una sincronización, el sistema guardará todos los cambios, dejando la tarea de seleccionar en un futuro qué archivos deberían estar eliminados al usuario final.

Es necesario, también, un sistema para poder listar todos los archivos y poderlos comparar entre cliente y servidor. De esta forma podemos hacer un listado rápidamente de los archivos que debemos enviar de un extremo a otro para mantener la sincronización de forma exacta.

Asimismo, también es necesario un control de flujo de datos, para que no se pierdan los archivos en el momento del envío, y saber bloquear los hilos de ejecución en los puntos clave del código para que los eventos no se pisen entre ellos. Añadido a esto, necesitamos un sistema de almacenamiento de datos (base de datos) para poder guardar los usuarios y sus contraseñas, y un sistema de cifrado y codificación de datos críticos (contraseñas y hashes de archivo únicos).

¿Algún objetivo no se ha cumplido?

A pesar de que se han añadido muchísimos sistemas de control sobre los cambios en archivos, se han rediseñado los algoritmos de comprobación en la aplicación servidor, el sistema operativo seguirá lanzando los eventos de cambios de los archivos de forma caótica. Los eventos, al ser threads del propio sistema operativo, pueden llegar a superponerse entre ellos y a bloquearse, dejando cambios por avisar, o enviando información errónea al servidor. Es necesario un estudio en profundidad sobre éste fallo y un rediseño sobre cómo detectar los cambios en los archivos y avisarlos al servidor.

6.2 Desviaciones de la planificación

La planificación del proyecto ha sido respetada en casi toda su totalidad, salvo varias excepciones:

- El control de cambios de archivos (diseño de la aplicación cliente) tardó mas de lo esperado. En concreto se necesitaron dos semanas mas para poder comprender en su totalidad cómo funcionan los eventos del sistema operativo. La primera versión basada en el diseño original no coincidía con la forma en que el sistema operativo Windows lanza los eventos de cambios a la aplicación, y fue necesario volver a la fase de diseño, estudiando la documentación oficial de Microsoft, para poder desarrollar las funciones necesarias.
- Al 60% aproximadamente del desarrollo (cuando el proyecto se encontraba en la fase de implementación de la aplicación servidor), el .NET Framework sufrió una actualización mayor. Se pasó de la versión 3.5 (con la que se empezó el proyecto), a la versión 4.0. La repercusión más grave fue la de la redefinición del socket dentro del lenguaje de programación. Las comunicaciones extremo a extremo fueron mejoradas notoriamente por parte de Microsoft en sus librerías, y se paró durante una semana el desarrollo del proyecto para volver a la librería de comunicaciones (protocolo de comunicaciones) y aprovechar las nuevas funcionalidades para optimizar la comunicación.

Debido a estas desviaciones, se tuvo que trabajar más intensamente sobre las implementaciones de las aplicaciones cliente y servidor, para no retrasar el proyecto y salirse del plazo establecido.

6.3 Líneas de mejora

En este apartado explicaremos las ideas que fueron desechadas durante el transcurso de la implementación y el diseño del proyecto, ya sea por falta de tiempo en la realización o por alta complejidad.

6.3.1 Panel informativo

En una de las primeras versiones de la aplicación se creó un panel informativo emergente que se desplazaba desde la barra de tareas hacia el centro de la pantalla, como si de una notificación emergente se tratara. El problema de éste panel era que necesitaba de comunicación con threads hijos, ya que las notificaciones se iban generando conforme llegaban del servidor al cliente.

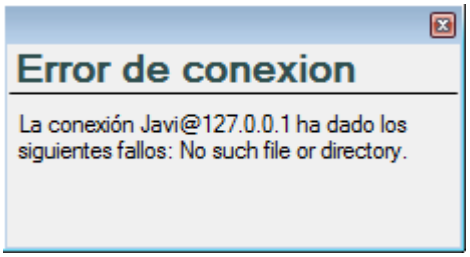


Fig 15. Ventana de notificaciones.

El formulario sigue diseñado y con todos los métodos para ir añadiendo mensajes de error conforme van llegando, pero está pendiente de la implementación de una cola y de un sistema de semáforos para que los threads puedan interactuar con él. Debido a que el sistema de colas y semáforos se implemento mucho mas tarde, se utilizó el sistema de notificaciones de Windows (mensajes emergentes desde el icono de la barra de tareas) para solventar el problema.

6.3.2 CVS ramificado

Una de las carencias de Sky Net es que no posee un gestor de versiones. Posee capacidad para generar versiones, y para recuperar esas versiones, pero no para modificarlas y crear nuevas versiones a partir de las antiguas. La aplicación permite visualizar una versión anterior, pero no volver a esa versión. Una futura mejora sería mejorar la gestión de CVS para detectar automáticamente cuándo se realiza un cambio en una versión antigua, y generar una nueva versión, totalmente de forma transparente. Y que el usuario pueda volver a una versión anterior en cualquier momento.

6.3.3 Almacenamiento externo

Otra de las carencias del sistema de Sky Net es que el almacenamiento por parte del servidor se realiza de forma local. Se puede llegar a configurar una ruta UNC, o utilizar una VPN para utilizar un disco duro de otro lugar (por ejemplo, un NAS, o un servidor de almacenamiento). Sin embargo, una posible mejora sería incorporar la compatibilidad con servidores FTP para poder almacenar los datos en otro servidor totalmente ajeno al sistema. De ésta forma se podría crear toda una infraestructura entre servidores Sky Net, y tener los archivos centralizados en un servidor con alta capacidad y alta seguridad de acceso a los datos.

CAPÍTULO 7

BIBLIOGRAFÍA

7. Bibliografía

7.1 Bibliografía On-Line

1. http://msdn.microsoft.com/es-es/: Website de la librería de ayudas de Microsoft sobre el .NET Framework 4.0 y sus respectivos lenguajes de programación. Gran cantidad de información y ejemplos sobre cómo utilizar clases y realizar algoritmos. **(último acceso 22/06/2011)**
2. http://www.canalvisualbasic.net/foro/visual-basic-net/: Website de comunidad de usuarios española sobre Visual Basic.NET 4.0. En ella se han realizado varias consultas sobre temas, así como recopilación de información sobre sockets y threads. **(último acceso 15/06/2011)**
3. http://www.syncrom.com/temarios/sfe1/demos/Demo%20Concurrencia.pdf.: Manual de threads y sincronización de threads por parte de la empresa Syncrom. En él se explican los principios básicos sobre la concurrencia de threads, la sincronización de threads, y métodos y ejemplos para poder realizarlo. **(último acceso 08/04/2011)**
4. http://sqlite.phxsoftware.com/.: Website de las librerías SQLite adaptadas para el .NET Framework 4.0. **(último acceso 12/05/2011)**
5. http://es.wikipedia.org/.: Enciclopedia libre sin ánimo de lucro que recoge gran cantidad de definiciones y artículos. Utilizada para la gran mayoría de definiciones de la documentación. **(último acceso 22/06/2011)**
6. http://www.codeproject.com.: Comunidad de programadores (en inglés) con amplios artículos, ejemplos, y códigos fuente sobre muchas consultas y problemas. Se han realizado varias consultas sobre sockets. **(último acceso 05/05/2011)**
7. http://www.modeliosoft.com/.: Website de donde se puede obtener el programa de diseño UML gratuito Modelio. **(último acceso 08/06/2011)**

7.2 Agradecimientos

Agradecimientos especiales a ***Nacor Teruel***, director de ECOM Telecomunicaciones, Valencia. ***Alex Grau Rodríguez,*** diseñador, ***Carme Corominas Bover,*** fotógrafa, ***Marc Pérez,*** programador y analista, por el apoyo, el interés, las sugerencias, las aportaciones y las observaciones proporcionadas durante la realización de todo el proyecto.

Agradecimiento especial a ***Gerisistem, S.L.,*** empresa donde trabajo actualmente, por prestarme todo el equipo informático y colaborar en la fase de pruebas, fuera de mi horario laboral.

Y agradecimientos generales, pero no menos importantes, a más de una veintena de personas que colaboraron en la detección de errores del software, utilizándolo y haciendo de beta-testers en las distintas fases del proyecto, a través del grupo de ***Facebook*** creado con el objetivo de conseguir colaboración en juegos de pruebas y logs completos del sistema.

Sabadell, *28 de Junio* de *2011*

Javier Carreño Izquierdo