Índice general

Capítulo 1

Resumen

El proyecto que he elegido y después desarrollado como proyecto de fin de carrera consiste en la creación de un aplicación distribuida de intercambio de audio por *streaming*[1].

La aplicación permite, a grandes rasgos, que los usuarios del sistema compartan canciones y escuchen aquellas que otros usuarios hayan compartido, la realización de todo tipo de operaciones(creación, eliminación, modificación, visualización, escucha) sobre listas de canciones o *playlists*, así como la consulta de diversos tipos de información sobre grupos y/o canciones, como imágenes y vídeos del grupo, letras de las canciones, conciertos...

La realización del proyecto se ha llevado a cabo siguiendo una metodología basada en la realización de prototipos, que actualmente cuenta con dos iteraciones o fases, con sus respectivos prototipos. El primero de estos prototipos simplemente incluye las funcionalidades centrales del sistema(control de reproducción, búsqueda de canciones, gestión de playlists, consulta de la información en la web), así como un primer prototipo de la interfaz gráfica de usuario del programa cliente de la aplicación y de la página web. Una vez finalizado y estudiado dicho prototipo, hemos establecido una serie de mejoras(que detallaremos en su momento) que se corresponderían con una segunda iteración en el desarrollo de la aplicación. Ambas fases del desarrollo de la aplicación se dividen a su vez en dos partes principales. La primera de ellas engloba todo lo referente al *análisis y al diseño* de la aplicación, desde la especificación o modificación de los requisitos de la aplicación hasta la implementación y las pruebas, además del diseño e implementación de una base de datos en la segunda fase. Por otro lado, la segunda parte está centrada en la *implementación* del proyecto, realizada en base al diseño de la fase anterior.

Al no tratarse de una aplicación de gestión específica para una organización, los requisitos han sido definidos por nosotros mismos, en lugar de recibirlos del

[1] Tipo de distribución remota de contenidos multimedia en red en la que no es necesario descargar el archivo completo.

cliente, especificando todas las funcionalidades deseadas para el sistema, dando pie a todo el resto del proceso de desarrollo.

De este modo, en el primer prototipo, el proyecto tiene como primer objetivo la realización de un aplicación distribuida de streaming de audio grande y compleja, como medio para llevar a la práctica conocimientos de programación distribuida y el proceso de diseño de software del paradigma orientado a objetos.

Por otra parte, el segundo prototipo tiene como objetivo la mejora tanto del modelo de datos(gracias a las ventajas de las bases de datos) como del modelo de la aplicación, llevando a cabo ampliaciones y modificaciones para mejorar el rendimiento del sistema obtenido como consecuencia del desarrollo del primer prototipo.

Capítulo 2

Introducción

Este capítulo tiene como objetivo dar una introducción al entorno que propició la idea del desarrollo de la aplicación que nos ocupa. Para ello, daré una breve introducción al concepto de las aplicaciones distribuidas, mostrando las características de algunas aplicaciones multimedia de este tipo que nos han servido de motivación e inspiración para la realización del proyecto. Por otro lado, también describiré los objetivos del proyecto, las motivaciones para su desarrollo y la estructura que seguirá esta memoria.

2.1. Descripción de la aplicación

Antes de nada, demos una breve definición de qué es un sistema distribuido. Según la wikipedia inglesa, la *computación distribuida* es un campo de las ciencias de la computación que estudia los sistemas distribuidos. Un *sistema distribuido* consiste en múltiples computadoras autónomas que se comunican a través de una red de ordenadores, de modo que cada una de ellas interactúa con las demás para llegar a un objetivo común . Un programa de ordenador que se ejecuta en un sistema distribuido se llama *programa distribuido*, mientras se llama *programación distribuida* al proceso de escribir programas distribuidos. Si bien existen numerosas tipos de arquitecturas dentro de la computación distribuida, no creo necesario exponer cada uno de estos tipos, sino más bien resaltar el concepto básico, es decir, la resolución de un problema dado por medio de una serie de ordenadores que colaboran para resolverlo, proporcionando las siguientes ventajas:

- Mayor *capacidad*: Mayor capacidad de almacenamiento de datos, más I/O, mayor capacidad de cómputo para poder atender más clientes, mayor precisión de procesamiento...
- Mayor *seguridad*: Mediante replicación datos, disponibilidad de procesamiento...

Cuando pensamos qué aplicación distribuida crear, pensamos en un sistema de intercambio de música en streaming distribuida, mezclando características de programas comerciales ya existentes como eMule o Spotify. A continuación, describiré la idea inicial sobre cómo iba a ser la aplicación que íbamos a desarrollar.

El funcionamiento básico sería similar a programas como Spotify, pero combinándolo con la idea de un programa P2P híbrido, en el que los clientes envían la música que tienen almacenada a otros clientes. Básicamente, una serie de clientes se conectan a un servidor, que a través de la interfaz cliente les muestra todas aquellas canciones y listas de canciones que están disponibles en sistema actualmente . Una vez el cliente haya decidido escuchar una de las canciones o haya creado una playlist con algunas de ellas, el servidor hace de intermediario a la hora de establecer las conexiones entre clientes y se encarga de las búsquedas. Un dato importante a tener en cuenta es que el servidor no almacena físicamente la música que se sirve, sino que lo hace cada cliente, localmente. Lo que hace el servidor es guardar un listado de las canciones disponibles de cada cliente asociado a su dirección IP.

Los usuarios van a tener un programa cliente con interfaz gráfica que usarán para reproducir la música y crear listas de reproducción. Cuando tengamos hecha una playlist personal, si el usuario que tiene una canción cierra la sesión, se mostrará esa canción como inaccesible, examinando para ello los usuarios conectados para marcarla como accesible en el momento adecuado. A través de esta interfaz gráfica, los usuarios serán también capaces de compartir sus propias listas de canciones.

Además, buscaremos información sobre los artistas que los clientes vayan solicitando escuchar: biografía, imágenes, vídeos de YouTube del grupo... Toda esa información se mostrará a través de una página web, de modo que todos los clientes(y cualquier otra persona que acceda a la web, aunque no tenga el programa cliente) tengan acceso a ella. Además, desde esa página web podremos ver qué música se está compartiendo actualmente en el sistema, pudiendo seleccionar cualquiera de los grupos que se nos muestren para mostrar la información ya descrita.

Todo lo anteriormente descrito se corresponde a la funcionalidad que en un principio ideamos para el sistema. La arquitectura real del sistema se describirá con todo detalle en el diagrama de despliegue.

2.2. Objetivos

El objetivo final de este proyecto consiste en la creación de una aplicación distribuida que cumpla las especificaciones expuestas en el apartado anterior, diseñando e implementando un sistema que cumpla todos los requisitos del sistema, que expondré con detalle posteriormente, dentro de este documento. La aplicación, simplificando al máximo y obviando, de momento, las subdivisiones en diferentes elementos(que tienen más que ver con la naturaleza distribuida de la aplicación y que podremos ver con todo detalle en el diagrama de despliegue), está formada por tres partes principales:

- Interfaces cliente que permitan al usuario del sistema llevar a cabo las funcionalidades básicas del sistema, como el control de reproducción, la creación de listas de canciones, su visualización... Además, aparte de aquellas funcionalidades básicas ya expuestas, la interfaz cliente contará con una ventana de configuración en la que el cliente podrá configurar la ruta donde quiere compartir las canciones, los puertos a utilizar por la aplicación...
- Un servidor que controle el acceso de los usuarios al sistema y se encargue de llevar a cabo el papel de mediador para que los clientes lleven a cabo el intercambio del audio por streaming(lo que se lleva a cabo de manera transparente al usuario, por supuesto). Al igual que he hecho antes, destacar que esto sólo se correspondería con un resumen básico de la funcionalidad del servidor, que será ampliada y expuesta al detalle posteriormente.
- Un servidor web que controla el registro de los usuarios en el sistema y que muestra a los usuarios del sistema diferentes tipos de información sobre los grupos musicales correspondientes a las canciones que están siendo compartidas en el sistema actualmente.

2.3. Motivación

La razón de mayor peso a la hora de impulsar la elección de este trabajo han sido principalmente tres:

En primer lugar, el desarrollo de esta aplicación brinda la posibilidad de llevar a cabo un diseño e implementación completo de una aplicación, haciendo frente a todas las etapas que componen el proceso de desarrollo del sistema : partiendo de la especificación de los requisitos y hasta hasta obtener el producto final. Así, un trabajo como éste permite llevar a la práctica los principios de Ingeniería del Software y de Planificación de Proyectos estudiados en mi titulación. Quizá se trate de la razón de más peso de todas las que me han motivado a llevar a cabo este proyecto y no otro.

En segundo lugar, la realización de este proyecto permite aplicar conocimientos adquiridos por separado de manera conjunta, desarrollando un programa

que implique conocimientos sobre bases de datos, ingeniería del software, planificación de proyectos, sistemas distribuidos, redes... Creo que es muy positivo afrontar el diseño de una aplicación de cierto tamaño que permita ensamblar todos estos conocimientos, ya que implica un mayor aprendizaje que el desarrollo de pequeñas prácticas sobre cada materia de un modo aislado.

En tercer y último lugar, pero no menos importante, este proyecto me ha permitido profundizar en el uso de herramientas de gran importancia dentro del desarrollo del software, tales como entornos de desarrollo integrados, herramientas CASE o LaTeXpara el desarrollo de la documentación.

2.4. Estructura de la memoria

La presente memoria está dividida en 11 apartados. En los dos primeros, se trata de dar una visión global de cómo es el sistema que queremos obtener como consecuencia del desarrollo de este proyecto.

El tercer apartado describe las herramientas utilizadas durante del desarrollo del proyecto. El enfoque que he elegido ha sido dar una escueta información sobre cada una de las herramientas, centrándome más en el porqué de la elección de cada herramienta y qué aporta al proyecto que he desarrollado.

El cuarto apartado abarca la planificación del proyecto, llevando a cabo la descomposición del proyecto en actividades y dividiendo éstas a su vez en tareas. Asignamos un tiempo a cada una de estas tareas, así como los recursos necesarios para llevarlas a cabo, ordenando las tareas en una sucesión coherente que permita llevar a cabo el proyecto. Llevar a cabo dicha planificación será de gran utilidad para estudiar la evolución del proyecto en relación a lo estudiado y para establecer el coste monetario, el tiempo de desabollo y los recursos necesarios para llevar a cabo el proyecto.

El quinto, sexto y séptimo apartado se corresponderán con el análisis, diseño y implementación del *primer prototipo* de la aplicación, respectivamente . Dentro del análisis, describiré todos los detalles que se han tenido en cuenta en el desarrollo de la aplicación. En el diseño, por su parte, mostraré tres tipos de diagramas(diagramas de casos de uso, diagramas de secuencia y diagramas de clase), lo cual proporcionará una excelente documentación para el diseño de la aplicación. Cada uno de los diagramas contará con una breve explicación para una mejor comprensión del diseño. El séptimo apartado, es decir, el apartado de implementación, analiza la estructura del sistema que se va a implantar,analizando sus características y dando diversos detalles acerca de la implementación de la aplicación. También se documentará el diseño del modelo físico de los datos, basado en una base de datos relacional.

El octavo apartado contiene una breve exposición del *segundo prototipo* de la aplicación, aquél que se obtiene como resultado de aplicar, de modo incremental, aquellas nuevas funcionalidades solicitadas así como mejoras a las funcionalidades ya existentes.

En el noveno apartado expondremos las conclusiones a las que hemos llegado tras la realización del proyecto. Dentro de dicho apartado, detallaré posibles mejoras en el sistema, además de posibles ampliaciones en la funcionalidad del sistema o adaptación a otro tipo de tecnologías alternativas.

Por último, los dos últimos apartados contienen los recursos Web, la bibliografía y el índice de las figuras contenidas en la memoria.

Capítulo 3

Consideraciones iniciales

Este capítulo contiene ciertas consideraciones que hemos realizado a la hora de plantear el proyecto, principalmente entorno a las tecnologías a utilizar en el proyecto y las herramientas utilizadas durante el desarrollo de este. Por último, exoonemos las razones por las que es de vital importancia utilizar un sistema gestor de bases de datos que se encargue de la gestión y almacenamiento de toda la información contenida en el sistema. El capítulo está dividido en dos secciones : una para las tecnologías y otra para las herramientas.

3.1. Tecnologías

Esta sección se encarga, como ya he dicho, de analizar las tecnologías seleccionadas para el desarrollo de la aplicación. El objetivo de la misma no es, ni mucho menos, dar una descripción minuciosa de dichas tecnologías sino más bien exponer aquellas características de las mismas que las hacen adecuadas para el proyecto, es decir, las características por las que las he elegido en lugar de otras tecnologías disponibles.

3.1.1. Java

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria, por medio del recolector de basura o *garbage collector*.

Java es un lenguaje interpretado : las aplicaciones Java están típicamente compiladas en un bytecode, aunque la compilación en código máquina nativo también es posible. En el tiempo de ejecución, el bytecode es normalmente interpretado o compilado a código nativo para la ejecución, aunque la ejecución directa por hardware del bytecode por un procesador Java también es posible.

Todo este proceso gira en torno al que quizás sea el componente más importante de la tecnología Java : la máquina virtual Java o *Java Virtual Machine,* JVM. Se trata de una máquina virtual de proceso nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial(el Java bytecode), el cual es generado por el compilador del lenguaje Java. Las características que más nos interesan son doslas siguientes:

1. Son un medio ideal para alcanzar la *portabilidad*, es decir, un código Java debería de poder ser ejecutado en cualquier plataforma eal 100% y hay que tomar ciertas precauciones para preservar siempre la portabilidad). Esto es de gran utilidad en nuestra aplicación, ya los usuarios que usan este tipo de aplicaciones pueden venir potencialmente de muchas plataformas distintas como Windows, Mac Os X, alguna de las muchas distribuciones GNU/Linux o *BSD... Debido a esa variedad de posibles plataformas, consideramos de gran utilidad utilizar un lenguaje multiplataforma, lo que nos permitirá ahorrarnos mantener una versión distinta del programa para cada plataforma, además de todas las compilaciones del código fuente que deberíamos realizar para cada arquitectura del procesador y para cada plataforma.

2. Introduce otro *nivel de abstracción* y de *protección*, entre la computadora y el software que ejecuta sobre ella. Esta característica también resulta de gran interés en una aplicación que si pasase a producción sería indudablemente un foco de ataques, al contener información sensible sobre direcciones ip de muchos usuarios. Otra característica que aumenta la seguridad del lenguaje es la gestión de memoria automática del garbage collector, ya que la gestión de memoria en lenguajes como C/C++ siempre ha sido un foco de *bugs* y posibles ataques. Como no necesitamos un nivel de control tan grande como en otro tipo de aplicaciones ni tampoco un rendimiento demasiado optimizado(que sí necesitaríamos para desarrollar un kernel de un sistema operativo, por ejemplo), Java me parece una gran opción para este proyecto en concreto.

3. Java proporciona las herramientas y librerías necesarias para la interconexión TCP/IP. Existen librerías de rutinas para acceder e interactuar con protocolos como http y ftp, entre muchos otros. Esto nos es de gran utilidad al desarrollar un *sistema distribuido* en el que varios ordenadores tienen que colaborar unos con otros para llegar a cumplir un objetivo. El hecho de que Java sea un lenguaje tan *orientado a la red* es una notable ventaja, por tanto.

4. Java es un lenguaje de programación *robusto*, ya que realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución(gracias a un gran sistema de excepciones). La comprobación de tipos en Java ayuda a detectar errores, lo antes posible, en el ciclo de desarrollo. Java obliga a la declaración explícita de métodos, reduciendo

así las posibilidades de error. Estas características reducen drásticamente el tiempo de desarrollo de aplicaciones en Java. Esto hace que sea un lenguaje muy adecuado para un desarrollo como el nuestro, que tiene un tiempo de desarrollo relativamente corto y en el que queremos crear un primer prototipo que muestre la funcionalidad del sistema con la mayor rapidez posible.

5. Java es un lenguaje *multihilo,* dicho de otro modo, permite muchas actividades simultáneas en un programa. El beneficio de ser *multihilo* consiste en un mejor rendimiento interactivo y mejor comportamiento en tiempo real. Esta característica es fundamental para el desarrollo de nuestro proyecto, para lograr un rendimiento y un tiempo de respuesta aceptable necesitamos un sistema multihilo, sobre todo a la hora de obtener información referente a los artistas. Más adelante, cuando describamos los componentes del sistema, veremos el porqué de esto.

6. Java cuenta con Java Database Connectivity, más conocida por sus siglas JDBC, una API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede, utilizando el dialecto SQL del modelo de base de datos que se utilice. Obviamos los diferentes tipos de drivers JDBC que existen, destacando que contamos con una serie de clases muy útiles para interactuar con una gran diversidad de sistemas gestores de bases de datos. Este útil sistema supone otra de las características por las que hemos elegido Java, ya que si bien muchos lenguajes de programación son capaces de interactuar con bases de datos, hemos utilizado con anterioridad estos sistemas en Java y consideramos que su uso es muy intuitivo y a la vez potente.

7. Por último, aunque no se trate estrictamente de una característica del lenguaje de programación Java, el ser un lenguaje muy utilizado y orientado a Internet hace que podamos encontrar una gran cantidad de APIs para todo tipo de servicios web como Picasa,Flickr,GoogleMaps, Wikipedia,YouTube,Last.fm... Esto nos es realmente útil en nuestra aplicación, ya que una buena parte de su atractivo resulta de la obtención de datos por medio de este tipo de servicios.

3.2. Herramientas

En esta sección se da una breve descripción de todas las herramientas utilizadas en la realización de este proyecto, así como las causas de la elección de esas herramientas.

3.2.1. Rational Rose

Rational Rose es una herramienta software para el modelado visual mediante UML de sistemas software. Es utilizada para especificar,analizar y diseñar

un sistema antes de pasar a su construcción y cubre todo el ciclo de vida de desarrollo de un proyecto : concepción y formalización del modelo, construcción de componentes, transición a los usuarios y certificación de las distintas fases. Por tanto, se trata de una aplicación ideal para analistas que utilizan el Proceso Unificado de Desarrollo con herramientas UML.

Las opciones que nos da esta herramienta son muchas, entre las que podemos destacar las siguientes:

- Rational Rose permite que los diseñadores utilicen el modelo de desarrollo iterativo(a veces llamado desarrollo evolutivo) donde el nuevo diseño se puede crear en etapas con la salida de una iteración que se convierte en la entrada al siguiente.
- Soporte de ingeniería Forward y/o reversa para algunos de los conceptos más comunes de Java 1.5.
- La generación de código Ada, ANSI C ++, C++, CORBA, Java y Visual Basic, con capacidad de sincronización modelo- código configurables .
- Capacidad de análisis de calidad de código.
- Modelado UML para trabajar en diseños de base de datos, con capacidad de representar la integración de los datos y los requerimientos de aplicación a través de diseños lógicos y físicos.
- Publicación web y generación de informes para optimizar la comunicación dentro del equipo.

Aplicado al proyecto que nos ocupa, resulta de gran utilidad el hecho de que la herramienta utilice un proceso de desarrollo iterativo controlado, donde el desarrollo se lleva a cabo en una secuencia de *iteraciones*. Cada una de ellas comienza con un primer análisis, diseño e implementación para identificar los riesgos del diseño y tratar de hallar estrategias para reducir su efectos o hacerlos desaparecer, en caso de ser posible. Cuando la implementación haya pasado todas aquellas pruebas que nosotros hemos definido, dicha implementación se revisa y se añaden los elementos modificados al modelo. Una vez se ha llevado a cabo la actualización del modelo, se pasa a la siguiente iteración.

3.2.2. MySQL

MySQL es un sistema de gestión de base de datos relacional, multihilo y multiusuario. El servidor MySQL está diseñado para entornos de producción críticos, con alta carga de trabajo así como para integrarse en software para ser distribuido.

El software MySQL tiene una doble licencia. Los usuarios pueden elegir entre usar el software MySQL como un producto Open Source bajo los términos de la licencia GNU General Public Licencia(`http://www.fsf.org/licenses/`) o pueden adquirir una licencia comercial estándar de MySQL AB.

A continuación, enumeraré unas pocas de las muchas características de MySQL que me han llevado a elegirlo como sistema gestor:

- Está escrito en C y C++, lo que hace que el sistema tenga un rendimiento notable.
- Gracias a GNU Automake, Autoconf, y Libtool, se trata de un sistema portable(básicamente esto hace que funcione en cualquier sistema que disponga de un compilador de C++ y una biblioteca de subprocesos o *threads* adecuada). Se puede ver una lista de los sistemas operativos que MySQL soporta en `http://dev.mysql.com/doc/refman/5.0/es/which-os.html`.
- Existen APIs para una gran variedad de lenguajes de programación, entre ellos C, C++, Eiffel, Java, Perl, PHP, Python, Ruby...
- Uso completo de multi-hilos mediante threads del kernel. Pueden usarse fácilmente multiples CPUs si están disponibles.
- Usa tablas en disco B-tree(MyISAM) muy rápidas con compresión de índice lo que hace de MySQL es una base de datos muy rápida en la lectura.
- Un sistema de reserva de memoria muy rápido basado en threads.
- Joins muy rápidos usando un multi-join de un paso optimizado.
- Un sistema de privilegios y contraseñas que es muy flexible y seguro, y que permite verificación basada en el host. Las contraseñas son seguras porque todo el tráfico de contraseñas está encriptado cuando se conecta con un servidor.

Así, las características de MySQL(donde no entraremos en más detalle, para ampliar información acudir a `http://dev.mysql.com/doc/refman/5.0/es/index.html`) hacen que sea un gestor de bases de datos relacionales muy utilizado en diferentes tipos de entorno como aplicaciones web(como Drupal o phpBB), aplicaciones de escritorio de todo tipo de sistemas operativos, la plataforma LAMP, el portal Wikipedia o aplicaciones de control de errores como Bugzilla. En nuestro caso, toda la parte de muestra de información tiene baja concurrencia en la

modificación de datos y en cambio el entorno es intensivo en lectura de datos, lo que hace a MySQL ideal para este tipo de aplicaciones y es lo que nos ha llevado a elegirlo como sistema gestor, además de por la gran variedad y calidad de herramientas para esta herramienta, disponibles para múltiples plataformas.

Para el desarrollo de este proyecto,hemos utilizado la versión 5.1.41 de MySQL. Además, hemos utilizado algunas herramientas asociadas a MySQL, las cuales detallamos a continuación.

3.2.2.1. MySQL Query Browser

MySQL Query Browser es una herramienta gráfica proporcionada por MySQL AB para crear, ejecutar, y optimizar consultas en un ambiente gráfico, donde el MySQL Administrator esta diseñado para administrar el servidor MySQL. MySQL Query Browser esta diseñado para ayudarle a consultar y analizar datos almacenados en una base de datos MySQL. Hemos utilizado esta herramienta por su simplicidad y porque, en conjunción con MySQL Administrator y MySQL Workbench, nos permite controlar totalmente nuestra base de datos en su conjunto, dentro de una interfaz gráfica amigable y sencilla.

3.2.2.2. MySQL Administrator

Es programa muy útil para administrar, visualmente y de manera sencilla, servidores de bases de datos MySQL. Se trata de un software multiplataforma, con un desarrollo activo, que dentro de su sencillez permite ciertas funcionalidades como:

- Configuración de las opciones de inicio de los servidores.
- Inicio y detención de servidores.
- Monitorización de conexiones al servidor.
- Administración de usuarios.
- Monitorización del estado del servidor, incluyendo estadísticas de uso.
- Visualización de los logs de servidor.
- Gestión de copias de seguridad y recuperaciones.
- Visualización de catálogos de datos.

Resumiendo, he elegido esta aplicación ya que permite realizar las tareas administrativas necesarias para la base de datos de nuestro proyecto por medio de un programa intuitivo y que se adapta perfectamente al tamaño del proyecto, que al no tratarse más que de un prototipo en desarrollo, no necesita de una labor de administración demasiado profunda por el momento . En caso de que nuestras necesidades cambiaran, habría que sopesar un cambio hacia otras aplicaciones con una mayor funcionalidad.

3.2.2.3. MySQL Workbench

MySQL Workbench es una herramienta de modelado de bases de datos visual multiplataforma. Es una herramienta poderosa, con numerosas funcionalidades detalladas en `http://www.mysql.com/products/workbench/features.html` . Para este proyecto, lo hemos utilizado para la creación y documentación de la base de datos, ya que permite crear un modelo de base de datos y todos sus componentes(tablas, relacionas, disparadores, procedimientos almacenados,vistas...) de manera sencilla y semi-automática. Esto nos permite centrarnos más en los aspectos de diseño de la base de datos que en su implementación.

3.2.3. NetBeans

NetBeans es un entorno de desarrollo integrado(*Integrated Development Tool*) apto para Java y con una enorme variedad de plugins que permiten desarrollo en otros lenguajes(C,C++,Python,Perl, JavaScript y un largo etcétera), modelado UML básico, diseño web, integración con Tomcat y con sistemas gestores de bases de datos, control de versiones... En resumen, estamos ante un entorno integrao de desarrollo gratuito que proporciona todas las características que necesitamos para nuestro proyecto y muchas más en caso de un crecimiento en la entidad del mismo o en los lenguajes de programación y plataformas utilizadas. Se trata de un IDE en constante evolución, del cual he utilizado la versión 6.8, en conjunto con la máquina virtual OpenJDK Client VM 1.6.0_18.

3.2.4. Apache Tomcat

Tomcat es un servidor web con soporte de servlets y JSPs desarrollado por la Apache Foundation. Tomcat implementa las especificaciones de los servlets y de JavaServer Pages(JSP) de Sun Microsystems. Aunque Tomcat puede funcionar como servidor web por sí mismo, hoy en día el uso principal es el de servidor web autónomo para entornos con alto nivel de tráfico y alta disponibilidad. Sin ahondar demasiado en aspectos técnicos internos de la herramienta ni pretender realizar un estudio riguroso, veamos algunas de las ventajas y desventajas del mismo:

- Ventajas
  - Es gratis, es fácil de instalar, se ejecuta en máquinas más pequeñas y es compatible con las API más recientes de Java. Puede descargarse, instalarse y probarse en menos de una hora. Tomcat ocupa muy poco espacio, teniendo su código binario(todo clases de Java) un tamaño total de apenas un megabyte, con lo cual proporciona un buen rendimiento.
  - Es muy fiable al tratarse de un proyecto de software libre, lo cual implica que miles de desarrolladores contribuyen con código y ayudan a solucionar *bugs,* hecho de especial importancia para un sistema que se encuentra en producción en innumerables empresas.

  - Su integración con Java y el IDE NetBeans es total, lo que hace bastante sencillo crear un esbozo del servidor web que necesitamos en un tiempo relativamente corto. Además, cuenta con una excelente documentación, plagada de ejemplos y didácticas explicaciones en el sitio web oficial del proyecto `http://tomcat.apache.org/tomcat-6.0-doc/`.

- Desventajas

  - Compatibilidad con J2EE. Aunque Tomcat no es directamente compatible con todas las API de J2EE que soportan otras alternativas como WebSphere(como JDBC, JNDI, JavaMail, RMI, JMS, XML y EJB), todas esas API, con la notable excepción de EJB, están disponibles agregando archivos Java(JAR) disponibles gratuitamente.
  - El proyecto Jakarta carece de un departamento de servicio técnico, si bien existe la opción de plantear dudas en foros específicos. De todos modos, el sitio web oficial cuenta con numerosos tutoriales y FAQs.
  - La escalabilidad de las aplicaciones y el soporte de Java Enterprise Beans es peor que en otras alternativas.

  Por lo tanto, hemos optado por un servidor web basado en Tomcat debido a las necesidades actuales de la aplicación: si bien una aplicación totalmente profesional y con gran carga de trabajo necesitaría de un servidor Glassfish o WebSphere(que pueden dar un mejor rendimiento en conjunto pero que también suponen un coste y una dificultad nada despreciables), se ha optado por Tomcat porque nos proporciona un servidor ligero, bien integrado con Java y con un proceso de desarrollo más sencillo para proyectos como el que nos ocupa. En el caso de un gran crecimiento de las necesidades del servidor web, habría que estudiar el migrado a otro servidor web, si bien, como ya he dicho, Tomcat se adapta perfectamente a las necesidades actuales.

3.2.5. Texworks

Texworks es un entorno de escritura de documentos TEX(en este caso,LATEX) Unicode, que tiene un visualizador PDF integrado, y una interfaz simple y accesible. La razón de la utilización de este programa es obtener una documentación elegante y aprovechar todas las ventajas que LATEX aporta a la edición de documentos científicos, además de aplicar y asentar mis conocimientos básicos sobre este complicado sistema de composición de textos.

Capítulo 4

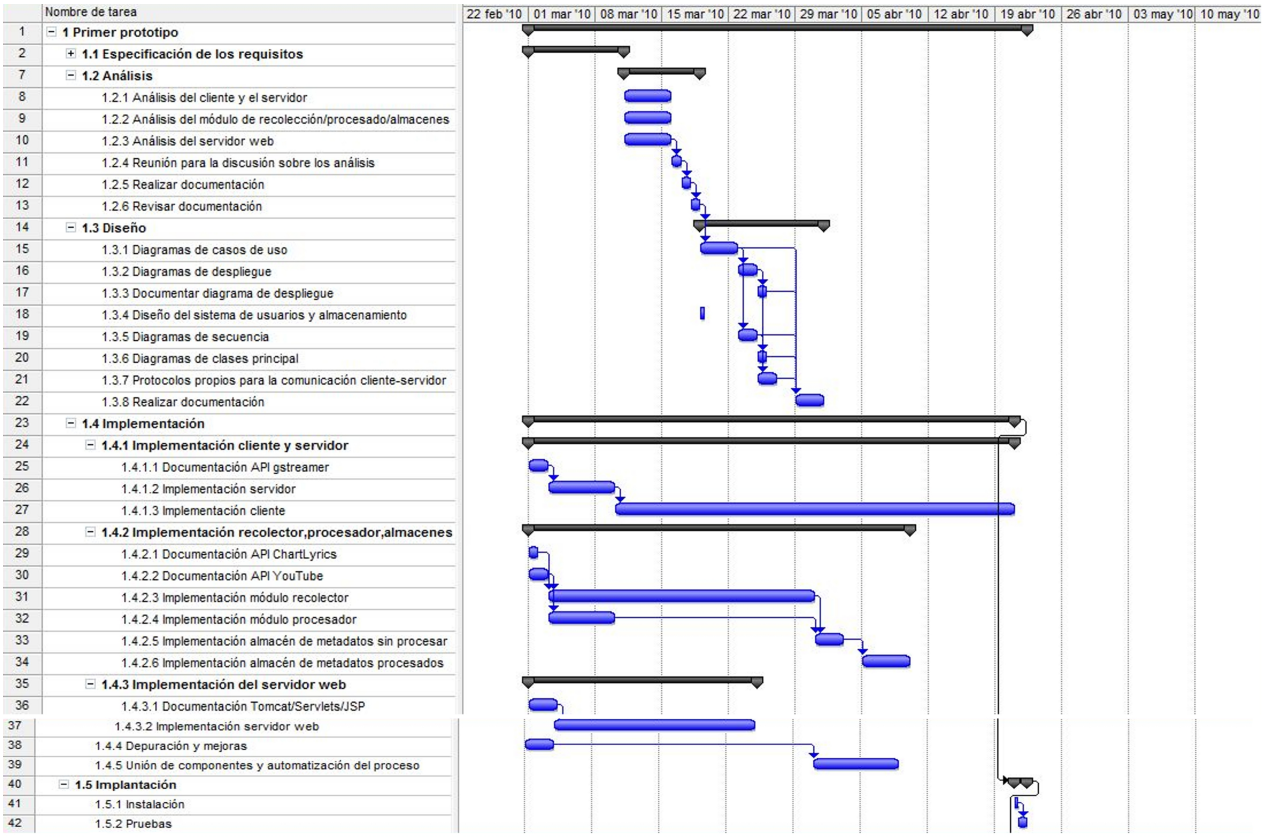
Gestión del proyecto

Este capítulo se centra en los aspectos de gestión de proyectos. Por ello, dividiremos el proyecto en una serie de tareas, estableciendo la duración de cada tarea y las dependencias que tengan entre ellas, así como los recursos asignados, por medio de un diagrama de Gantt.

4.1. Diagrama de Gantt

El siguiente diagrama de Gantt muestra las fechas de inicio y de fin de cada tarea. Las tareas son representas en el eje vertical, mientras que el eje horizontal se corresponde al tiempo. Las tareas que forman el proyecto están agrupadas en función de la fase del desarrollo del proyecto en que han sido realizadas. Además, podemos ver las dependencias entre las diferentes tareas y los recursos asignados a cada una de ellas.

Notar que las tareas están situadas en paralelo en el eje correspondiente a tiempo, aunque unas tareas comiencen unas después de otras. Esto es debido a la anchura de la imagen.



	Nombre de tarea
1	**1 Primer prototipo**
2	**1.1 Especificación de los requisitos**
7	**1.2 Análisis**
8	1.2.1 Análisis del cliente y el servidor
9	1.2.2 Análisis del módulo de recolección/procesado/almacenes
10	1.2.3 Análisis del servidor web
11	1.2.4 Reunión para la discusión sobre los análisis
12	1.2.5 Realizar documentación
13	1.2.6 Revisar documentación
14	**1.3 Diseño**
15	1.3.1 Diagramas de casos de uso
16	1.3.2 Diagramas de despliegue
17	1.3.3 Documentar diagrama de despliegue
18	1.3.4 Diseño del sistema de usuarios y almacenamiento
19	1.3.5 Diagramas de secuencia
20	1.3.6 Diagramas de clases principal
21	1.3.7 Protocolos propios para la comunicación cliente-servidor
22	1.3.8 Realizar documentación
23	**1.4 Implementación**
24	**1.4.1 Implementación cliente y servidor**
25	1.4.1.1 Documentación API gstreamer
26	1.4.1.2 Implementación servidor
27	1.4.1.3 Implementación cliente
28	**1.4.2 Implementación recolector,procesador,almacenes**
29	1.4.2.1 Documentación API ChartLyrics
30	1.4.2.2 Documentación API YouTube
31	1.4.2.3 Implementación módulo recolector
32	1.4.2.4 Implementación módulo procesador
33	1.4.2.5 Implementación almacén de metadatos sin procesar
34	1.4.2.6 Implementación almacén de metadatos procesados
35	**1.4.3 Implementación del servidor web**
36	1.4.3.1 Documentación Tomcat/Servlets/JSP
37	1.4.3.2 Implementación servidor web
38	1.4.4 Depuración y mejoras
39	1.4.5 Unión de componentes y automatización del proceso
40	**1.5 Implantación**
41	1.5.1 Instalación
42	1.5.2 Pruebas

Figura 4.1: Diagrama de Gantt del proyecto(1)

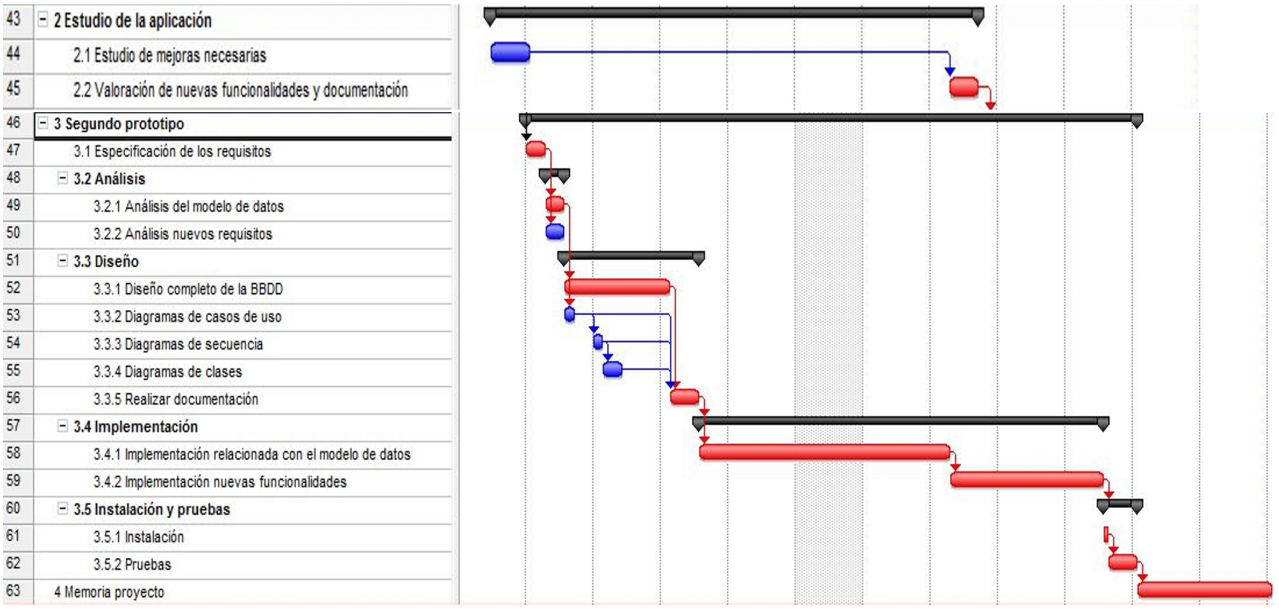


Figura 4.2: Diagrama de Gantt del proyecto(2)

Capítulo 5

Análisis

En esta sección se expone el análisis llevado a cabo a partir de las especificaciones establecidas para el proyecto. El objetivo es concretar todos los aspectos concernientes a qué se espera del sistema en esta fase del desarrollo del mismo, con el fin de servir de base para el diseño posterior. El análisis está dividido en tres partes principales. Las partes son: la sección cliente-servidor, la sección de recolección, procesado y almacenamiento de información y por último, la sección web.

5.1. Arquitectura cliente-servidor

Esta sección engloba toda la interacción entre los clientes y el servidor principal de la aplicación. Veamos las diferentes funcionalidades que se esperan de la aplicación cliente y del servidor principal en diferentes apartados.

5.1.1. Cliente

Los usuarios del sistema se conectan al sistema por medio de una aplicación cliente, facilitando para ello un nombre de usuario y una contraseña(el proceso de registro se lleva a cabo en la interfaz web) y la dirección IP y el puerto del servidor al que se quiere conectar. La necesidad de detallar la dirección y el puerto al que se quiere conectar el usuario viene motivada por una posible ampliación del programa: en las sucesivas iteraciones, si el sistema debe soportar un gran volumen de peticiones por parte de numerosos usuarios, será necesario un cluster de servidores, por lo que es conveniente que el usuario elija uno de esos servidores. En posteriores iteraciones del proceso de desarrollo, podríamos incluir algún sistema de configuración de aquellos servidores que más nos interesen. Además, lógicamente, el usuario tiene la capacidad de desconectarse del sistema en cualquier momento. Como ayuda a todo el proceso de conexión y desconexión la aplicación debe mostrar visualmente el estado de la conexión al servidor y si éste está en funcionamiento actualmente. Una vez verificados los

datos aportados por el usuario, el usuario tiene acceso a todas las funcionalidades básicas de la aplicación cliente: control de reproducción, compartición de canciones, búsqueda de canciones locales y remotas y gestión de playlists.

El usuario tiene la opción de realizar las operaciones usuales de control de reproducción en aplicaciones de este tipo: reanudar la reproducción, pausarla, detenerla y ajustar el volumen del audio.

Los usuarios del sistema comparten las canciones simplemente colocando los archivos de audio correspondientes en una carpeta de su disco duro. Por el momento, en esta fase de desarrollo, optamos por un emplazamiento fijo en lugar de uno configurable, simplemente por simplicidad.

En cuanto a la búsqueda de canciones, el usuario debe poder listar todas las canciones locales(aquellas residentes en su disco duro) y las remotas, es decir, aquellas canciones que estén compartiendo otros usuarios. Los usuarios también pueden buscar canciones compartidas utilizando el nombre del grupo cuyas canciones desea escuchar. Además, como característica adicional, los usuarios pueden ver qué otros usuarios están escuchando canciones que estén compartiendo, mostrando información sobre el usuario, su dirección IP, la canción que esta escuchando y el porcentaje del buffer que ha cargado.

Todas las canciones compartidas en el sistema pueden ser añadidas a listas de canciones o playlists. Los usuarios pueden realizar las siguientes operaciones sobre dichas listas de canciones: creación de una nueva playlist, listado de todas las playlist disponibles en el sistema, visualización del contenido de una playlist, modificación de una playlist ya existente y, por último, eliminación de una playlist. Una vez seleccionada una lista de canciones, el usuario puede controlar la reproducción a su antojo y además puede avanzar o retroceder en las canciones que la forman.

5.1.2. Servidor

El servidor se encarga básicamente de gestionar las conexiones de los usuarios al sistema, lo cual implica las siguientes funcionalidades:

- Controlar la conexión y desconexión de los usuarios al sistema.
- Mediar en el intercambio de audio entre los usuarios del sistema. Este proceso, que debe ser transparente al usuario, tiene como objetivo establecer una conexión entre los clientes para el intercambio.
- Gestión de la música actualmente compartida por los usuarios. El servidor se encarga de comprobar periódicamente el contenido de las carpetas en las que los usuarios colocan las canciones que quieren compartir, a fin de mostrar las canciones compartidas a los usuarios y que la sección de recolección, procesado y almacenado de la información busque información sobre los grupos a los que pertenecen esas canciones.

- Procesar las búsquedas de canciones que lleven a cabo los usuarios llevar a cabo la gestión de playlists.

Además de estas funcionalidades básicas, buscamos una serie de requisitos adicionales para el buen desempeño del servidor. Potencialmente, el servidor puede tener que enfrentarse con una gran cantidad de clientes, hecho que debe repercutir lo menos posible en el tiempo de respuesta en la aplicación cliente. Para ello, debemos buscar paralelizar todo lo posible el control de los usuarios por parte del servidor y que el control de reproducción se lleve a cabo de manera independiente al resto de funcionalidades. Posteriormente, en el capítulo correspondiente a la implementación de este primer prototipo, señalaré cómo se ha llevado a la práctica esta paralelización de tareas.

5.2. Protocolos de comunicación

El proceso de análisis de los requisitos de nuestra aplicación nos ha llevado a la decisión de desarrollar un pequeño protocolo de comunicación entre los clientes y el servidor o servidores de la aplicación. Las razones principales por las que hemos tomado esta decisión son dos:

1. El RMI es asíncrono y necesitamos que la comunicación entre los dos extremos sea síncrona(que haya un canal de comunicación entre ellos, en base a las funciones clásicas *send* y *receive*).

2. El uso de RMI aumentaría la complejidad de funcionamiento y añadiría problemas de rendimiento entre los clientes y el servidor. Esto es un grave problema ya que buscamos sencillez en nuestra aplicación y un buen rendimiento en las conexiones de los clientes al servidor.

Dicho esto, pasemos a detallar el funcionamiento de los protocolos que hemos desarrollado para esta aplicación.

5.2.1. Protocolo de comunicación entre clientes

Como ya hemos expuesto, la música es compartida entre los clientes. Para ello, cada cliente tendrá ejecutando un hilo preparado para atender y procesar las peticiones de otros clientes. La comunicación de control entre clientes se realiza mediante TCP(Transmission Control Protocol), para garantizar que el proceso de negociación previo a la transmisión de los datos se completa satisfactoriamente. La definición del protocolo será mediante texto, encapsulado en unos objetos con el único propósito de facilitar la comunicación entre los clientes.

El envío del stream del servidor al cliente se realizará a través de UDP(User Datagram Protocol) ya que el protocolo de transporte que mejor se adapta a la funcionalidad esperada para esta aplicación. Esto se debe a que podemos tolerar la pérdida de algunos paquetes a favor de una mayor velocidad de transferencia,

básica en todo tipo de servicios basados en streaming. Como el puerto UDP puede ser configurable, este debe ser enviado junto con la petición al servidor.

Debido a que todos los clientes pueden ser servidores en potencia, vamos a definir ambos roles. Un cliente que actúa como servidor es aquel que aceptará las peticiones para enviar la música, mientras que el que actuará como cliente es aquel que pedirá la música.

En esta fase del proceso de desarrollo, las únicas opciones que se barajarán en el protocolo serán:

- Solicitar el inicio de la reproducción de una canción.
- Parar reproducción de una canción.

Una vez que conectado el cliente al servidor(del cliente), este último esperará a recibir un objeto por parte del cliente, este procesará dicho objeto y enviará otro objeto de vuelta indicando el resultado de la operación. A continuación mostramos cuáles serán los mensajes encapsulados en esos objetos:

- Mensaje del cliente al servidor:

  - ***Play xxxxx.mp3 yyyy***. Donde yyyy es puerto UDP al que le deberá enviar el stream.

  - **Stop**

- Mensaje de respuesta del servidor al cliente:

  - En caso de que haya éxito: **OK xxxxxxx**, donde xxxxxxx es el tamaño en bytes del archivo mp3 que le va a enviar. Si ocurre algún tipo de error, debido a que el fichero no exista, se devolverá un mensaje **FAIL**.
  - Para el comando STOP, en caso de que haya éxito, **OK.** En caso de error, también se devolverá un mensaje FAIL.

5.2.2. Protocolo de comunicación entre cliente y servidor

Todos los clientes deben estar conectados al servidor durante todo el tiempo que dure su sesión. Como ya sabemos, el servidor será el encargado de realizar las búsquedas sobre las lista de ficheros compartidos y enviar dichas búsquedas a los clientes, además se encargará de gestionar el login de los usuarios y sus como sus playlists.
Operaciones entre el cliente y el servidor:

- Conectar

  El cliente se conecta al servidor. Acto seguido el cliente le envía el usuario y contraseña(se envía el hash MD5) al servidor. Si el login es exitoso, el servidor le envía un OK, en caso contrario le envía un FAIL. Una vez conectado el cliente le envía un objeto que contiene toda la música que va

a compartir con el resto de usuarios. El servidor procesa dicha información y cuando termina de procesarla, le envía otro OK si todo ha salido bien o un FAIL si ha habido algún error.

- Definición de los mensajes:
  - Cliente: Establece la conexión.
  - Cliente: Envía Usuario + Password.
  - Servidor: Envía OK o FAIL
  - Cliente: Envía un objeto con la lista de ficheros compartidos.
  - Servidor: Procesa la lista. Envía OK o FAIL.

- SEARCH

  El cliente envía un objeto con las palabra a buscar. El servidor realiza una búsqueda sobre la lista de canciones compartidas y envía un objeto con la lista de canciones que coinciden con el texto a buscar.

  - Definición de los mensajes:
    - Cliente: *SEARCH "xxxxx"*
    - Servidor: Procesa la búsqueda y envía un objeto con la información requerida.

- PLAY

  La función de este comando es indicar al cliente que usuario tiene la canción que ha solicitado. Para ello el servidor implementará una búsqueda en su lista. Una vez encontrada la canción, el servidor le enviará al cliente que realizó la petición la dirección IP y el puerto del cliente que tiene dicha canción. Si no encuentra la canción, el servidor devuelve FAIL.

  - Definición de los mensajes:
    - Cliente: *PLAY xxxx.mp3*.
    - Servidor: *SEARCH xxxx.mp3*.
    - Servidor: Envía ip + Puerto del cliente o FAIL.

- NUEVA PLAYLIST

  La función de este comando es crear una playlist en el almacén de perfiles para el usuario que lo ha solicitado. Si ha habido éxito se devuelve OK y si no FAIL.

  - Definición de los mensajes:
    - Cliente: NEWPLAYLIST nombre_usuario playlist.
    - Servidor: Envía la petición al almacén de perfiles por RMI .
    - Servidor: Envía OK o FAIL.

- Cargar Playlist

  La función de este comando es cargar una playlist en el almacén de perfiles para el usuario que lo ha solicitado. Si ha habido éxito se devuelve la playlist y en caso contrario FAIL.

  - Definición de los mensajes:
    - Cliente: LOADPLAYLIST nombre_usuario nombre_playlist.
    - Servidor: Envía la petición al almacén de perfiles por RMI.
    - Servidor: Envía la playlist o bien FAIL.

- Listar playlists

  La función de este comando es listar las playlists de un usuario . Si ha habido éxito ,etc. devuelve una lista con las playlists y en caso contrario FAIL.

  - Definición de los mensajes:
    - Cliente: *LISTPLAYLIST* nombre_usuario.
    - Servidor: Envía la petición al almacén de perfiles por RMI .
    - Servidor: envía: Lista de playlists o FAIL.

- Borrar playlist

  La función de este comando es borrar una playlist del almacén de perfiles para el usuario que lo ha solicitado. Si ha habido éxito se devuelve OK y si no FAIL.

  - Definición de los mensajes
    - Cliente: DELETEPLAYLIST nombre_usuario nombre_playlist.
    - Servidor: Envía la petición al almacén de perfiles por RMI.
    - Servidor: Envía OK o FAIL.

5.3. Adquisición, procesamiento y almacenamiento de la información

Esta parte de la aplicación es la que tiene como objetivo obtener información(imágenes, vídeos y letras) sobre los grupos musicales a los que pertenecen las canciones compartidas actualmente en el sistema, de modo que el usuario sea capaz de visualizar esa información a través de la interfaz web. La obtención de esa información conlleva un proceso que se lleva a cabo en tres fases:

1. Recolectar toda la información y almacenarla temporalmente.
2. Procesar esa información encontrada en base a una serie de criterios.
3. Almacenar la información procesada físicamente.

A fin de obtener un rendimiento aceptable en el sistema como conjunto, la recolección y el procesado de la información de cada grupo ha de ser realizada en paralelo.

5.4. Página Web

La sección web tiene dos funcionalidades básicas: el registro de nuevos usuarios al sistema y mostrar la información disponible sobre los grupos que están siendo actualmente compartidos en el sistema.

Para ello, el servidor web ha de comunicarse, por un lado, con el almacén de metadatos para extraer los datos solicitados por los usuarios de la página web. Por otro, también ha de comunicarse con el almacén de perfiles, que contiene los datos correspondientes a los usuarios registrados en el sistema.

Capítulo 6

Diseño

Este capítulo contiene el diseño realizado a partir del análisis del capítulo anterior. El objetivo de este diseño es describir todos los aspectos a conocer del sistema que queremos construir, tratando de implementar tanto los requisitos funcionales contenidos en el análisis como aquellos requisitos no funcionales que están implícitos en el mismo. Una vez finalizado el diseño, se habrá proporcionado una idea completa del sistema a desarrollar, enfocándolo desde el modelo de datos, el modelo funcional y el comportamiento.

6.1. Prototipos de la interfaz gráfica

En la fase de diseño, diseñamos unos pequeños prototipos de la interfaz del programa cliente de la aplicación, que mostraamos a continuación:

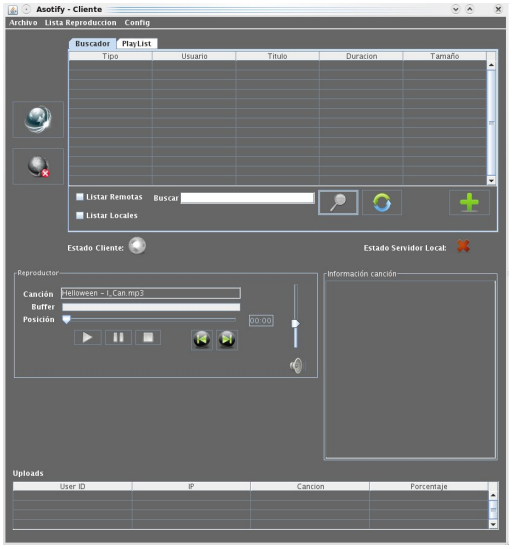


Figura 6.1: Primera versión de la interfaz gráfica del cliente(1)

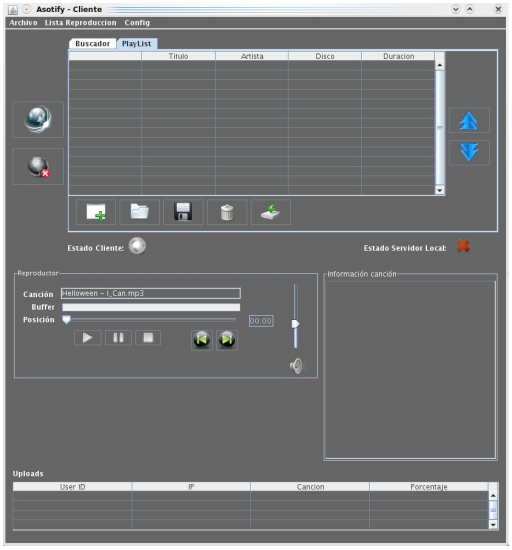


Figura 6.2: Primera versión de la interfaz gráfica del cliente(2)

Durante el diseño preliminar de esta interfaz, que se puede ver en 6.1y 6.1, hemos buscado que las interfaces gráficas sean sencillas, intuitivas y agradables, de modo que el uso de esta aplicación no suponga demasiado esfuerzo para el usuario. En un futuro, en caso de una gran ampliación en la funcionalidad de la aplicación, sería muy interesante añadir un menú de ayuda a la aplicación, con el objetivo de mejorar la satisfacción de un usuario hipotético.

Por último, también diseñamos una primera interfaz gráfica para el servidor web de nuestra aplicación:

Figura 6.3: Versión preliminar de la interfaz Web

6.2. La base de datos

6.2.1. Implantación de un SGBDR

La mejora que obtendríamos al implementar un sistema gestor de bases de datos para la aplicación que nos ocupa es enorme, tanto en el rendimiento como en la calidad de la organización de la información. En un principio, si no optamos por un modelo de datos basado en un sistema gestor de bases de datos fue por la escasa cantidad de información, manejable fácilmente en aquel momento mediante estructuras de ficheros y carpetas adecuadas, con una serie de criterios para su almacenamiento. Esto estaba más orientado a obtener un primer prototipo de la aplicación lo más rápido posible, principalmente. Sin embargo, con el crecimiento de los tipos de datos a almacenar en el sistema y el conjunto de funcionalidades del sistema en concreto, se hace absolutamente

necesaria la implementación de un sistema gestor de bases de datos, pues de otro modo el modelo de datos llegaría a ser absolutamente ingobernable y su rendimiento sería pésimo. Así, en el apartado siguiente veremos cuáles son los problemas de los sistemas de ficheros frente a los sistemas basados en bases de datos.

6.2.1.1. Situación actual. Sistema de archivos

El modo en el que estamos trabajando actualmente con la información en el sistema es con un sistema de archivos, es decir, guardando la información en una serie de archivos sitos en el sistema de archivos del sistema operativo. En este contexto, para permitir a los usuarios manipular la información, el sistema tiene un número de programas de aplicación que manipulan los archivos. Si las necesidades se incrementan, añadimos nuevos programas de aplicación al sistema. Procediendo de este modo, se van añadiendo más archivos y programas de aplicación al sistema, sobre la marcha. Mantener información de la organización en un sistema de procesamiento de archivos tiene una serie de inconvenientes importantes, que se enumeran a continuación:

- **Redundancia e inconsistencia de los datos**. Debido a que, en un proyecto de un cierto tamaño y duración en el tiempo, los archivos y programas de aplicación son creados por diferentes programadores en un largo período de tiempo, los diversos archivos tienen probablemente diferentes formatos y los programas pueden estar escritos en diferentes lenguajes. Más aún, la misma información puede estar duplicada en diferentes lugares(duplicidad de la información en diferentes archivos). Esta redundancia conduce a un almacenamiento y coste de acceso más altos. Además, puede conducir a inconsistencia de datos; es decir, las diversas copias de los mismos datos pueden no coincidir.

- **Dificultad en el acceso a los datos**. La utilización de un sistema de archivos para el modelado de los datos implica una mayor dificultad a los datos, ya que el acceso a dichos datos se hace mediante programas que accedan a los ficheros. De este modo, pequeños cambios en las necesidades de acceso a los datos o nuevas necesidades implican la creación o la modificación de dichos programas, que dependiendo de la complejidad de los ficheros de datos pueden llegar a ser realmente complejos. Un ejemplo muy ilustrativo de este problema puede encontrarse en la siguiente cita:

  > Supóngase que uno de los empleados del banco necesita averiguar los nombres de todos los clientes que viven en el distrito postal 28733 de la ciudad. El empleado pide al departamento de procesamiento de datos que genere dicha lista. Debido a que esta petición no fue prevista cuando el sistema original fue diseñado, no hay un programa de aplicación a mano para satisfacerla. Hay, sin embargo, un programa de aplicación que genera la lista de

> todos los clientes. El empleado del banco tiene ahora dos opciones: bien obtener la lista de todos los clientes y obtener la información que necesita manualmente, o bien pedir al departamento de procesamiento de datos que haga que un programador de sistemas escriba el programa de aplicación necesario. Ambas alternativas son obviamente insatisfactorias. Supóngase que se escribe tal programa y que, varios días más tarde, el mismo empleado necesita arreglar esa lista para incluir sólo aquellos clientes que tienen una cuenta con saldo de 10.000 ď o más. Como se puede esperar, un programa para generar tal lista no existe. De nuevo, el empleado tiene que elegir entre dos opciones, ninguna de las cuales es satisfactoria. La cuestión aquí es que el entorno de procesamiento de archivos convencional no permite que los datos necesarios sean obtenidos de una forma práctica y eficiente.

Veamos cuáles son las ventajas que nos aporta la utilización de un SGBDR frente a una organización basada en un sistema de archivos:

- **Mejora en la integridad de los datos**. La integridad de los datos de una base de datos se refiere a la validez y la consistencia de los mismos. Cuando los datos se modifican como consecuencia de una sentencia de inserción, actualización o eliminación, la integridad de los datos almacenados puede perderse de diversas maneras. No enumeraremos todas ellas, sino que simplemente diremos que dicha integridad se expresa mediante una serie de reglas que no se pueden violar. Estas reglas pueden ser aplicadas a los tados como a sus relaciones. Como ya hemos visto, no es fácil conseguir implementar esas restricciones con el modelo basado en ficheros, mientras que el sistema gestor de bases de datos, por naturaleza, debe de encargarse de garantizar las restricciones que el diseñador especifique, lo cual es una gran ventaja.

- **Mejora en la seguridad**. La información almacenada en una base de datos puede ser de gran valor, por lo que los SGBD deben encargarse de garantizar que esa información está segura frente a usuarios malintencionados(que traten de acceder a información privilegiada); frente a ataques que tengan como objetivo destruir o manipular la información; o bien simplemente ante los errores de los usuarios que sí estén autorizados. Normalmente, los SGBD disponen de un complej sistema de permisos a usuarios y grupos de usuarios mediante la otorgación de diferentes *roles*. Esto supone una enorme ventaja frente a los sistemas basados en ficheros, ya que permite, de una manera relativamente sencilla, otorgar diferentes categorías de permisos, lo cual reduce los problemas de seguridad de manera significativa.

- **Independencia de los datos**. Consiste en poder modificar el esquema de una base de datos(ya se trate del esquema lógico o el esquema físico) sin

necesidad de realizar modificaciones en las aplicaciones que acceden a los datos. Esto supone una mejora enorme con respecto al modelo basado en ficheros, donde es imposible garantizar un cierto grado de independencia entre los datos y las aplicaciones que acceden a ellos.

- **Concurrencia**. Se garantiza que más de un usuario puede acceder a la información al mismo tiempo manteniendo la integridad de los datos.
- **Servicio de copias de seguridad y recuperación ante fallos**. Los SGBD actuales proporcionan una forma eficiente de realizar copias de respaldo de la información almacenada en ellos, y de restaurar a partir de estas copias los datos que se hayan podido perder.

6.2.2. Estudio del dominio de información : Supuesto

Queremos modelar una base de datos para una aplicación de streaming de audio distribuida: los usuarios que se conectan al sistema, las canciones que escuchan y comparten, las playlist que construyen con éstas, así como todos los metadatos que el sistema recolecta de los grupos que están siendo compartidos por los usuarios del sistema en este momento: imágenes, vídeos, letras, biografía del artista, carátula del disco,etc.

De los usuarios que se conectan al sistema, es nesesario conocer el nombre de usuario(que será único y identificará a un usuario de manera unívoca) como dato obligatorio, su contraseña(encriptada con *md5*[1]), así como otra serie de datos correspondientes al perfil de usuario. Estos datos podrán ser consultados por otros usuarios en la web del programa: edad, sexo y país, además del identificador de la sesión actual . Por último, como consecuencia de la ampliación de la funcionalidad del sistema, se va a querer almacenar el correo electrónico del usuario, así como una imagen a modo de avatar del usuario , siendo éste de carácter no obligatorio.

Los usuarios pueden compartir canciones y/o escucharlas. De estas canciones que los usuarios comparten, necesitamos almacenar el grupo al que se corresponde, el título de la canción, el álbum al que pertenece,su duración,etc. Además, como resultado del proceso de recolección de información que se lleva a cabo para toda canción compartida en el sistema, contamos con la letra de la canción. Las canciones se identificarán de manera única con la combinación del nombre del grupo , el título del álbum y el título de la canción, considerando que existen múltiples canciones de distintos grupos con el mismo nombre pero no grupos con un mismo nombre.

Cada canción pertenece a un álbum, de los que queremos almacenar el título del album, la fecha de lanzamiento, su carátula y los tags que del álbum.

[1]En criptografía, MD5(abreviatura de Message-Digest Algorithm 5, Algoritmo de Resumen del Mensaje 5) es un algoritmo de reducción criptográfico de 128 bits ampliamente usado.

Además, cada álbum puede pertenecer a un grupo o a varios(por ejemplo si se trata de una colaboración entre varios grupos o un tributo a otro artista), pudiendo tener cada grupo varios álbumes, lógicamente.

Los usuarios también cuentan con la opción de crear/abrir/modificar/borrar listas de canciones o playlists. Cada una de estas playlist tendrá un identificador y un nombre, así como la lista de canciones que la componen. Cada una de estas playlist se identificará por medio de su código, que ha de ser único.

Sobre los grupos musicales, se desea conocer su nombre y si ya ha sido procesado dentro del sistema . Además, para cada grupo, se mostrará el artículo de la wikipedia correspondiente, para que el usuario disponga de más información sobre dicho grupo . Cada grupo se identificará por su nombre.

Para cada uno de estos grupos, se va a almacenar aquellos grupos de un estilo similar, de modo que los usuarios que escuchen una canción puedan acceder también a otros grupos similares, que quizás sean de su interés.

Como ya se ha dicho antes, para cada grupo del que los usuarios hayan compartido canciones, el sistema inicia un proceso de búsqueda de información asociada a dicho grupo: vídeos, imágenes, letras, carátulas, biografía y conciertos en fechas próximas.

En cuanto a los vídeos, el sistema se encarga de buscar y mostrar vídeos normales y vídeos relacionados con éstos, para que los usuarios puedan verlos a través de la interfaz web de la aplicación. Estos dos tipos de vídeos se van a almacenar de manera conjunta, ya que sus características son fundamentalmente idénticas: queremos almacenar el título del vídeo, la descripción , el embed o código HTML que permite incluir el vídeo en cualquier página web y el identificador del mismo(una cadena alfanumérica que identifica cada vídeo de manera única).

Las imágenes tienen un contenido análogo, es decir, para cada imagen recolectada sobre un grupo se almacenará un identificador(que es único e identifica cada imagen) y el link a la imagen(no vamos a almacenar físicamente el archivo de la imagen sino un enlace al mismo, por motivos de espacio).

Como complemento a toda la información anterior, el sistema se encarga de buscar y mostrar al usuario los conciertos de aquellos grupos cuyas canciones están siendo compartidas en el sistema, de los que nos interesa la fecha, ciudad, país, código postal, latitud y longitud. Para simplificar la distinción entre conciertos, usaremos un identificador(un código numérico) único para cada concierto.

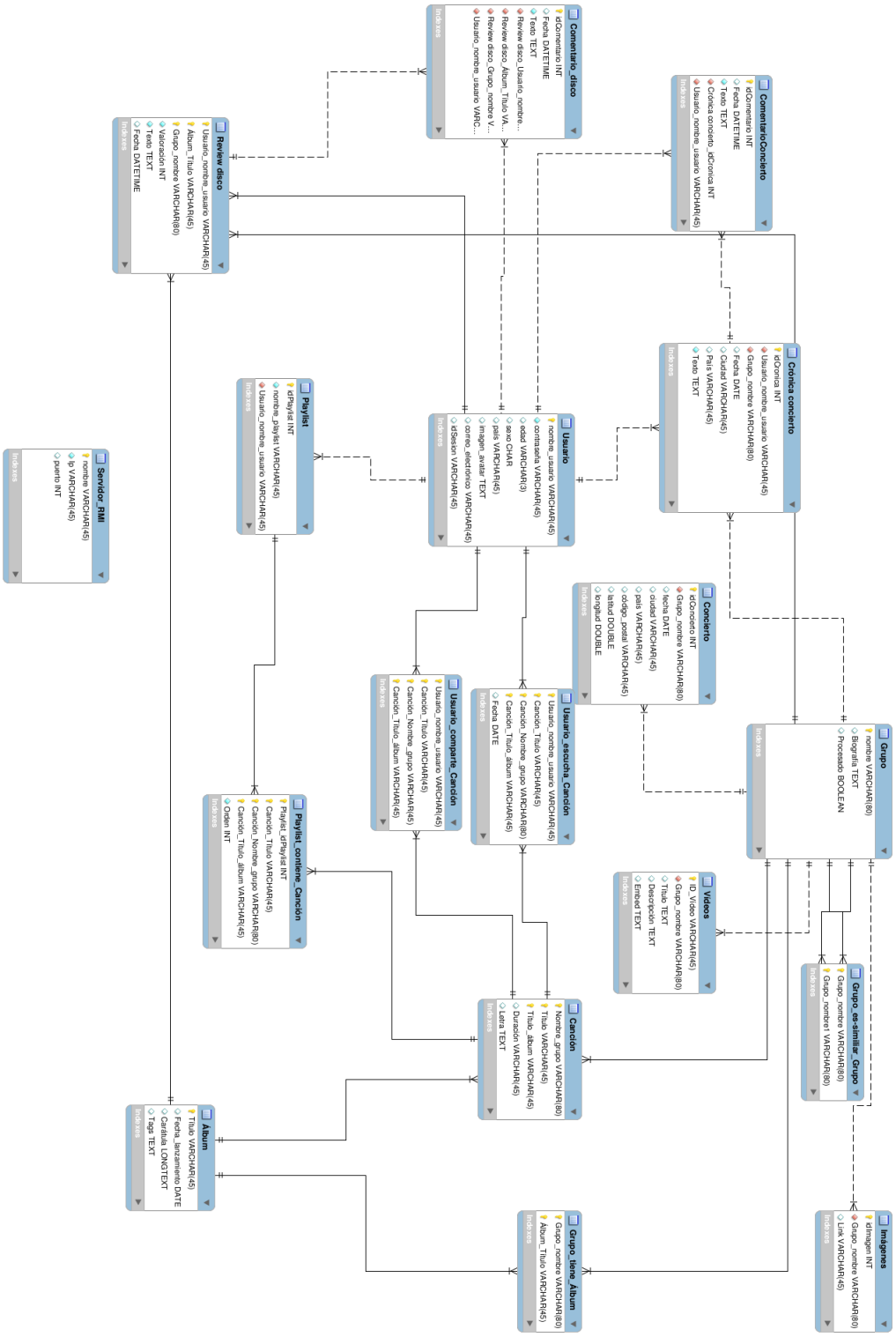
Además, como resultado de la ampliación en la funcionalidad de la página web solicitada por el cliente, vamos a añadir dos funcionalidades nuevas: escribir

y compartir reseñas de discos y crónicas de conciertos con otros usuarios del sistema y escribir comentarios en ambas. Una reseña de un disco se caracterizará por el nombre del grupo, el título del disco, la fecha en que fue escrita, una valoración numérica y el texto de la reseña propiamente dicha. Cada reseña se identificará gracias al nombre del grupo, el título del álbum y el autor de la misma, permitiendo de este modo varias reseñas por parte de distintos usuarios para un mismo disco. En cuanto a las reseñas de los conciertos, se deseará almacenar el nombre del grupo, la fecha, la ciudad, el país, una valoración numérica y imágenes que ilustren el mismo. Como ya hemos dicho, en ambas secciones, un usuario tendrá la opción de escribir comentarios, que aparte de los atributos que obtendrán como resultado de las relaciones con las reviews de discos y de conciertos, se caracterizarán por una fecha y un texto, este último de carácter obligatorio.

Por último, para simplificar la implementación y el acceso a los diversos servidores RMI de objetos, vamos a crear una tabla para los distintos servidores de RMI del sistema. Para cada servidor RMI, almacenamos su nombre(que identificará al mismo de manera única) , y la dirección IP y el puerto donde está ubicado(ambos datos serán de carácter obligatorio, ya que son imprescindibles para acceder a cada uno de los servidores).

6.2.3. Diseño de la base de datos

Una vez analizado el dominio de información o semántica, debemos realizar el análisis, diseño e implementación de un modelo de datos que lo refleje, basándonos para ello en el modelo relacional de Codd.



6.2.4. Estudio de las tablas

A continuación, realizamos un pequeño estudio de las tablas que componen el modelo, describiendo su funcionalidad, estructura, así como la interconexión entre ellas, centrándonos en las políticas que se llevarán a cabo en cada una de las tablas ante actualizaciones y borrados de sus tuplas, con objeto de mantener la integridad referencial.

- Usuario
  - Función : Tabla que almacena los usuarios conectados al sistema actualmente.
  - Descripción de la tabla

Atributo	Tipo	Descripción
Nombre_usuario(PK)	VARCHAR	Nombre de usuario o nick con el que se conecta el usuario
Contraseña(OBL.)	VARCHAR	Contraseña del usuario, encriptada con md5(Obligatorio)
Edad	INT	Edad del usuario
Sexo	CHAR	Sexo del usuario
País	VARCHAR	País de nacimiento del usuario
Imagen_avatar	TEXT	Link a la imagen que servirá de avatar al usuario
Correo	VARCHAR	Dirección de correo electrónico del usuario
idSesion	VARCHAR	Identificador de la sesión del usuario en la página web

  - Actualizaciones y borrados.
    - La tabla no contiene ninguna clave foránea.
- Canción
  - Función : Tabla que contiene las canciones compartidas por los usuarios actualmente.
  - Descripción de la tabla

Atributo	Tipo	Descripción
Título(PK)	VARCHAR	Título de la canción
Título_álbum(PK)	VARCHAR	Título del álbum al que pertenece la canción
Nombre_grupo(PK)	VARCHAR	Nombre del grupo autor del álbum
Duración	VARCHAR	Duración de la canción
Letra	TEXT	Letra de la canción, en caso de ser encontrada

  - Actualizaciones y borrados
    - FK 'Título_album' REFERENCIA 'Álbum'
      - ◇ ON UPDATE CASCADE : Todo álbum actualizado en 'Álbum' hará que se actualicen las tuplas que lo referencian en 'Canción'.
      - ◇ ON DELETE CASCADE : Todo grupo eliminado en 'Álbum' hará que se elimine toda tupla de 'Canción' que referencie esa tupla.

- FK 'Nombre_grupo' REFERENCIA 'Grupo'
  - ON UPDATE CASCADE : Todo grupo actualizado en 'Grupo' hará que se actualicen las tuplas que lo referencian en 'Canción'.
  - ON DELETE CASCADE : Todo grupo eliminado en 'Grupo' hará que se elimine toda tupla de 'Canción' que referencie esa tupla.

- Usuario_comparte_canción
  - Función : Tabla que contiene información sobre qué canciones ha compartido cada usuario.
  - Descripción de la tabla

Atributo	Tipo	Descripción
Usuario_nombre_usuario(PK)	VARCHAR	Nombre del usuario
Canción_*Título(PK)*	VARCHAR	Título de la canción
Canción_Nombre_grupo(PK)	VARCHAR	Nombre del grupo
Canción_álbum(PK)	VARCHAR	Nombre del álbum

  - Actualizaciones y borrados :
    - FK 'Usuario_nombre_usuario' REFERENCIA 'Usuario'
      - ON UPDATE CASCADE : Todo usuario actualizado en 'Usuario' tendrá como efecto la actualización de las tuplas que referencian dicho usuario en 'Usuario_comparte_canción'.
      - ON DELETE CASCADE : La eliminación de un usuario en 'Usuario' tendrá como efecto la eliminación de todas las tuplas que referencien a dicho usuario en 'Usuario_comparte_canción'.
    - FK 'Canción_grupo,Canción_álbum,Canción_título' REFERENCIA 'Canción'
      - ON UPDATE CASCADE : Todo nombre de grupo actualizado en 'Grupo' implicará la actualización de todas las tuplas de 'Usuario_comparte_canción' que referencien a ese grupo.
      - ON DELETE CASCADE : La eliminación de un grupo en 'Grupo' hará que sea eliminada de 'Usuario_comparte_canción' toda tupla que referencia a ese grupo.

- Usuario_escucha_canción
  - Función : Tabla que contiene información sobre las canciones escuchadas por cada usuario en la sesión.
  - Descripción de la tabla

Atributo	Tipo	Descripción
Usuario_nombre_usuario(PK)	VARCHAR	Nombre del usuario
Canción_*Título(PK)*	VARCHAR	Título de la canción
Canción_Nombre_grupo(PK)	VARCHAR	Nombre del grupo
Canción_álbum(PK)	VARCHAR	Nombre del álbum
Fecha	DATETIME	Fecha de la escucha

- Actualizaciones y borrados :
  - FK 'Usuario_nombre_usuario' REFERENCIA 'Usuario'
    - ON UPDATE CASCADE : Todo usuario actualizado en 'Usuario' tendrá como efecto la actualización de las tuplas que referencian dicho usuario en 'Usuario_escucha_canción'.
    - ON DELETE CASCADE : La eliminación de un usuario en 'Usuario' tendrá como efecto la eliminación de todas las tuplas que referencien a dicho usuario en 'Usuario_escucha_canción'.
  - FK 'Canción_grupo,Canción_álbum,Canción_título' REFERENCIA 'Canción'
    - ON UPDATE CASCADE : Todo nombre de grupo actualizado en 'Grupo' implicará la actualización de todas las tuplas de 'Usuario_escucha_canción' que referencien a ese grupo.
    - ON DELETE CASCADE : La eliminación de un grupo en 'Grupo' hará que sea eliminada de 'Usuario_escucha_canción' toda tupla que referencia a ese grupo.

- Playlist
  - Función : Tabla que contiene información sobre las listas de canciones que crean los usuarios del sistema.
  - Descripción de la tabla

Atributo	Tipo	Descripción
idPlayist(PK)	INT	Identificador numérico de la playlist
Nombre_playlist(OBL.)	VARCHAR	Nombre de la playlist(obligatorio)
Usuario_nombre_usuario	VARCHAR	Nombre del usuario autor de la playlist

  - Actualizaciones y borrados
    - FK 'Usuario_nombre_usuario' REFERENCIA 'Usuario'
      - ON UPDATE CASCADE : Todo usuario actualizado en 'Usuario' tendrá como efecto la actualización de las tuplas que referencian dicho usuario en 'Playlist'.
      - ON DELETE CASCADE : La eliminación de un usuario en 'Usuario' tendrá como efecto la eliminación de todas las tuplas que referencien a dicho usuario en 'Playlist'.

- Relación Playlist_contiene_canción

- Función : Tabla que contiene la relación entre playlists y las canciones que la componen.
- Descripción de la tabla

Atributo	Tipo	Descripción
Playlist_id(PK)	INT	Identificador numérico de la playlist
Canción_*Título(PK)*	VARCHAR	Título de la canción
Canción_Nombre_grupo(PK)	VARCHAR	Nombre del grupo
Canción_álbum(PK)	VARCHAR	Nombre del álbum
Orden(OBL.)	INT	Orden de cada canción dentro de la playlist

- Actualizaciones y borrados
  - FK 'Playlist_id' REFERENCES 'Playlist'
    - ◇ ON DELETE CASCADE : La eliminación de una playlist implicará la eliminación de todas aquellas tuplas de 'Playlist_contiene_canción' que referencien dicha lista de canciones.
  - FK 'Canción_grupo,Canción_álbum,Canción_título' REFERENCIA 'Canción'
    - ◇ ON UPDATE CASCADE : Todo nombre de grupo actualizado en 'Grupo' implicará la actualización de todas las tuplas de 'Playlist_contiene_canción' que referencien a ese grupo.
    - ◇ ON DELETE CASCADE : La eliminación de un grupo en 'Grupo' hará que sea eliminada de 'Playlist_contiene_canción' toda tupla que referencia a ese grupo.

- Grupo
  - Función : Tabla que contiene todos los grupos musicales correspondientes a las canciones que están siendo compartidas en el sistema en un momento dado.
  - Descripción de la tabla

Atributo	Tipo	Descripción
Nombre(PK)	VARCHAR	Nombre del grupo
Biografía	VARCHAR	Biografía de la banda
Procesado	BOOLEAN	Indica si la información está procesada

  - Actualizaciones y borrados.
    - La tabla no contiene ninguna clave foránea.

- Relación grupo_similar_grupo
  - Función : Tabla que contiene la relación entre aquellos grupos parecidos en estilo.

- Descripción de la tabla

Atributo	Tipo	Descripción
Grupo_nombre1(PK)	VARCHAR	Nombre del primer grupo
Grupo_nombre2(PK)	VARCHAR	Nombre del segundo grupo

- Actualizaciones y borrados:
  - FK 'Grupo_nombre1' REFERENCIA 'Grupo'
    - ON UPDATE CASCADE : Todo grupo que sea actualizado en 'Grupo' tendrá como efecto la actualización de todas las tuplas de la relación que referencien a dicho grupo.
    - ON DELETE CASCADE : Todo grupo eliminado en grupo tendrá como efecto la eliminación de aquellas tuplas que referencien a ese grupo en esta relación.

- Álbum

  - Función : Tabla que contiene información sobre los álbumes de los artistas que están siendo compartidos en el sistema.
  - Descripción de la tabla

Atributo	Tipo	Descripción
Título(PK)	VARCHAR	Tïtulo del álbum
Fecha_lanzamiento	DATE	Fecha de lanzamiento del álbum
Carátula	LONGTEXT	Imagen de la carátula del álbum
Tags	TEXT	Tags correspondientes al álbum

  - Actualizaciones y borrados
    - La tabla no contiene ninguna clave foránea.

- Relación Grupo_tiene_álbum

  - Función : Tabla que contiene la relación entre grupos y álbumes.
  - Descripción de la tabla

Atributo	Tipo	Descripción
Grupo_nombre(PK)	VARCHAR	Nombre del grupo
Álbum_titulo(PK)	VARCHAR	Título del álbum

  - Actualizaciones y borrados
    - FK 'Nombre_grupo' REFERENCIA 'Grupo'
      - ON UPDATE CASCADE : Todo grupo actualizado en 'Grupo' hará que se actualicen las tuplas que lo referencian en 'Grupo_tiene_álbum'.
      - ON DELETE CASCADE : Todo grupo eliminado en 'Grupo' hará que se elimine toda tupla de 'Grupo_tiene_álbum' que referencie esa tupla.
    - FK 'Álbum_titulo' REFERENCIA 'Álbum'

◇ ON UPDATE CASCADE : Todo álbum actualizado en 'Álbum' hará que se actualicen las tuplas que lo referencian en 'Grupo_tiene_álbum'.

◇ ON DELETE CASCADE : Todo grupo eliminado en 'Álbum' hará que se elimine toda tupla de 'Grupo_tiene_álbum' que referencie esa tupla.

- Imágenes
  - Función : Tabla que contiene las imágenes que ha recolectado el sistema para todos aquellos grupos que están siendo compartidos en el sistema actualmente.
  - Descripción de la tabla

Atributo	Tipo	Descripción
Idimagen(PK)	INT	Identificador único de la imagen
Grupo_nombre	VARCHAR	Grupo al que se corresponde la imagen
Link	VARCHAR	Link a la imagen como tal

  - Actualizaciones y borrados
    - FK 'Grupo_nombre' REFERENCIA 'Grupo'
      - ◇ ON UPDATE CASCADE : La actualización de la clave primaria de la tabla origen tendrá como efecto la actualización de la clave foránea de aquellas tuplas de esta relación que referencien a ese grupo.
      - ◇ ON DELETE CASCADE : La eliminación de las tuplas correspondientes a un grupo al que referencie esta relación tendra como efecto la eliminación de dichas tuplas.

- Vídeos
  - Función : Tabla que contiene los vídeos(normales y relacionados) de YouTube para los grupos de música que se estén compartiendo en el sistema actualmente.
  - Descripción de la tabla

Atributo	Tipo	Descripción
ID_Vídeo(PK)	VARCHAR	Identificador del vídeos de YouTube
Grupo_nombre	VARCHAR	Grupo al que se corresponde el vídeo
Título	TEXT	Título del vídeo
Descripción	TEXT	Descripción del contenido del vídeo
Embed	TEXT	Código HTML necesario para mostrar el vídeo

  - Actualizaciones y borrados
    - FK 'Grupo_nombre' REFERENCIA 'Grupo'
      - ◇ ON UPDATE CASCADE : La actualización de la clave primaria de la tabla origen tendrá como efecto la actualización de la clave foránea de aquellas tuplas de esta relación que referencien a ese grupo.

◇ ON DELETE CASCADE : La eliminación de las tuplas correspondientes a un grupo al que referencie esta relación tendra como efecto la eliminación de dichas tuplas.

- Crónica concierto
  - Función : Tabla que contiene las crónicas de conciertos que escriben los usuarios.
  - Descripción de tabla

Atributo	Tipo	Descripción
Id_crónica(PK)	INT	Identificador de la crónica del concierto
Usuario_nombre_usuario	VARCHAR	Nombre del usuario que escribe la crónica
Grupo_nombre	VARCHAR	Grupo correspondiente al concierto
Fecha	DATE	Fecha de publicación de la crónica
Ciudad	VARCHAR	Ciudad en la que se celebró el concierto
País	VARCHAR	País donde se celebró el concierto
Texto(OBL.)	TEXT	Texto de la crónica

  - Actualizaciones y borrados
    - FK 'Usuario_nombre_usuario' REFERENCIA 'Usuario'
      - ◇ ON UPDATE CASCADE : La actualización de la clave primaria de un usuario tendrá como efecto la actualización de todas las tuplas de esta tabla que lo referencien.
      - ◇ ON DELETE CASCADE : El borrado tendrá un efecto análogo, eliminándose todas las tuplas de esta tabla que referencien a usuarios que han sido borrados de la tabla origen.
    - FK 'Grupo_nombre_grupo' REFERENCIA 'Grupo'
      - ◇ ON UPDATE CASCADE : La actualización de la clave primaria de un grupo tendrá como efecto la actualización de todas las tuplas de esta tabla que lo referencien.
      - ◇ ON DELETE CASCADE : El borrado tendrá un efecto análogo, eliminándose todas las tuplas de esta tabla que referencien a grupos que han sido borrados de la tabla origen.

- Concierto
  - Función : Tabla que contiene información sobre los conciertos que realizan los grupos musicales.
  - Descripción de la tabla

Atributo	Tipo	Descripción
IdConcierto(PK)	INT	Identificador del concierto
Fecha	DATE	Fecha de realización del concierto
Ciudad	VARCHAR	Ciudad donde se realiza el concierto
País	VARCHAR	País donde se realiza el concierto
Código_postal	VARCHAR	Código Postal de la localidad
Latitud	DOUBLE	Latitud geográfica del emplazamiento
Longitud	DOUBLE	Longitud geográfica del emplazamiento
Grupo_nombre_grupo	VARCHAR	Grupo al que se corresponde el concierto

- Actualizaciones y borrados
  - FK '*Grupo_nombre_grupo* ' REFERENCIA 'Grupo'
    - ON UPDATE CASCADE : La actualización de la clave primaria de un grupo tendrá como efecto la actualización de todas las tuplas de esta tabla que lo referencien.
    - ON DELETE CASCADE : El borrado tendrá un efecto análogo, eliminándose todas las tuplas de esta tabla que referencien a grupos que han sido borrados de la tabla origen.

- Review disco
  - Función : Tabla que contiene las reviews o críticas que los usuarios realizan en torno a un álbum de un grupo.
  - Descripción de la tabla

Atributo	Tipo	Descripción
Grupo_nombre(PK)	VARCHAR	Grupo autor del álbum
Album_título(PK)	VARCHAR	Titulo del álbum
Usuario_nombre_usuario(PK)	VARCHAR	Usuario autor de la crónica
Valoración(OBL.)	INT	Valoración numérica
Texto(OBL.)	TEXT	Texto propiamente dicho de la review
Fecha	DATETIME	Fecha en la que se escribió la reseña

  - Actualizaciones y borrados
    - FK 'Grupo_nombre ' REFERENCIA 'Grupo'
      - ON UPDATE CASCADE : La actualización de la clave primaria de un grupo tendrá como efecto la actualización de todas las tuplas de esta tabla que lo referencien.
      - ON DELETE CASCADE : El borrado tendrá un efecto análogo, eliminándose todas las tuplas de esta tabla que referencien a grupos que han sido borrados de la tabla origen.
    - FK 'Album_título' REFERENCIA 'Album'
      - ON UPDATE CASCADE : La actualización en una clave primaria referenciada por las tuplas de esta tabla tendrá como efecto la actualización en cascada de las mismas.

◇ ON DELETE CASCADE : Cuando se elimine una tupla de la tabla referenciada se eliminarán aquellas tuplas que la referencien.

- FK 'Usuario_nombre_usuario' REFERENCIA 'Usuario'
  - ◇ ON UPDATE CASCADE : La actualización de la clave primaria de un usuario tendrá como efecto la actualización de todas las tuplas de esta tabla que lo referencien.
  - ◇ ON DELETE CASCADE : El borrado tendrá un efecto análogo, eliminándose todas las tuplas de esta tabla que referencien a usuarios que han sido borrados de la tabla origen.

- Comentario Concierto
  - Función : Tabla que contiene los comentarios que los usuarios escriben para los conciertos.
  - Descripción de la tabla

Atributo	Tipo	Descripción
Id_comentario	INT	Identificador único del comentario
Fecha	DATETIME	Fecha en que fue escrito el comentario
Texto(OBL.)	TEXT	Texto del comentario
Id_cronica	INT	Identificador de la crónica
Usuario_nombre_usuario	VARCHAR	Nombre del usuario autor del comentario

  - Actualizaciones y borrados
    - FK 'Usuario_nombre_usuario' REFERENCIA 'Usuario'
      - ◇ ON UPDATE CASCADE : La actualización de la clave primaria de un usuario tendrá como efecto la actualización de todas las tuplas de esta tabla que lo referencien.
      - ◇ ON DELETE CASCADE : El borrado tendrá un efecto análogo, eliminándose todas las tuplas de esta tabla que referencien a usuarios que han sido borrados de la tabla origen.
    - FK 'Id_cronica' REFERENCIA 'Cronica Concierto'
      - ◇ ON UPDATE CASCADE : La actualización de la clave primaria de una crónica implicará la actualización de todas las tuplas de esta tabla que lo referencien.
      - ◇ ON DELETE CASCADE : El borrado tendrá un efecto análogo, eliminándose todas las tuplas de esta tabla que referencien a crónicas de conciertos que hayan sido borradas.

- Comentario Review
  - Función : Tabla que contiene los comentarios que los usuarios escriben en las reviews de discos.

- Descripción de la tabla

Atributo	Tipo	Descripción
idComentario(PK)	INT	Identificador del comentario
Fecha	DATETIME	Fecha de escritura del comentario
Texto(OBL)	TEXT	Texto del comentario
ReviewDisco_Grupo	VARCHAR	Grupo al que se corresponde la review
ReviewDisco_Usuario	VARCHAR	Usuario autor de la review referenciada
ReviewDisco_Titulo_Album	VARCHAR	Título del álbum de la review referenciada
Usuario_nombre_usuario	VARCHAR	Autor del comentario

- Actualizaciones y borrados
  - FK '*ReviewDisco_Grupo*,*ReviewDisco_Usuario*,*ReviewDisco_Titulo_Album*' REFERENCIA Review Disco
  - FK '*Usuario_nombre_usuario*' REFERENCIA Usuario
    - ON UPDATE CASCADE
    - ON DELETE CASCADE

- Servidor RMI

  - Función : Tabla que contiene información sobre los Servidores RMI del sistema.
  - Descripción de la tabla

Atributo	Tipo	Descripción
Nombre	VARCHAR	Nombre del servidor RMI
IP	VARCHAR	Dirección IP del servidor RMI
Puerto	INT	Puerto en el que está escuchando el servidor RMI

  - Actualizaciones y borrados
    - La tabla no contiene claves foráneas.

6.3. Diagramas de modelado

El modelado del sistema se ha realizado con UML(Unified Modeling Language) que es el lenguaje de modelado de sistemas software más utilizado en la actualidad. Se trata de un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema basándonos en el paradigma orientado a objetos. De los 13 tipos de diagramas definidos en el estándar 2.0 de UML, hemos creído conveniente la creación de los siguientes tipos de diagrama:

- Diagrama de despliegue
- Diagramas de casos de uso
- Diagramas de secuencia

- Diagramas de clase

La creación de estos diagramas nos dará una representación gráfica de *cómo* ha de ser el sistema desde el punto de vista de cada uno de los actores participantes.

6.3.1. Diagrama de despliegue

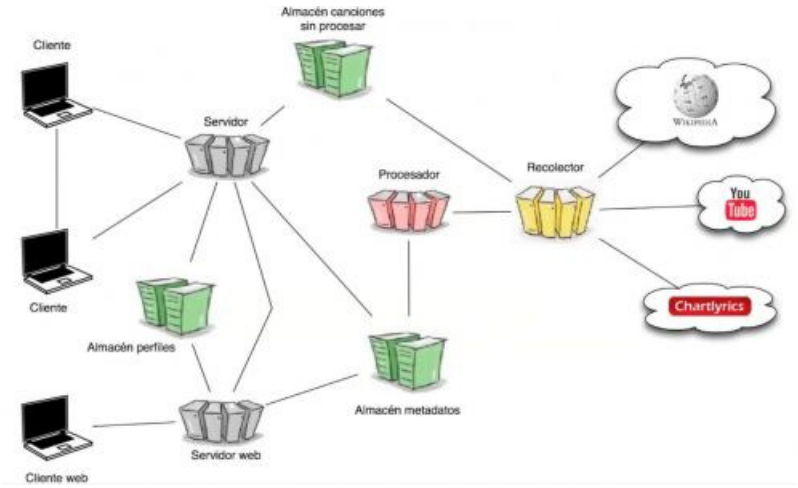


Figure 6.4: Diagrama de despliegue del sistema

A continuación mostramos el diagrama de despliegue del sistema. Un diagrama de despliegue es un tipo de diagrama del Lenguaje Unificado de Modelado que se utiliza para modelar el hardware utilizado en las implementaciones de sistemas y las relaciones entre sus componentes. Dicho de otro modo, nos permite ver cuáles son los componentes principales en los que se subdivide un sistema, así como las conexiones entre esos componentes.

Creemos conveniente llevar a cabo una pequeña exposición de la función de cada componente y con qué elementos del diagrama de despliegue se comunica. Finalmente, con la intención de aclarar definitivamente el funcionamiento del sistema, daré una descripción de cómo sería una secuencia de ejecución típica del sistema. Posteriormente, en el apartado correspondiente a la implementación, describiremos detalladamente cuáles han sido las técnicas de implementación empleadas para cada uno de los componentes del sistema.

- Cliente
    - El componente cliente es la aplicación cliente principal, desde la que podemos controlar la mayoría de los aspectos del programa como conexión al sistema, playlists, desconexión, control de reproducción, búsqueda de canciones,etc.Este módulo es el encargado de compartir

la música con el resto de clientes, para ello este módulo hará las veces de cliente y servidor pues cada cliente puede escuchar música de otro cliente y a la vez puede enviar música a uno o más clientes. Además, el cliente se conecta con un servidor central con quien mantendrá una conexión durante todo el tiempo de vida del usuario. El protocolo de comunicación establecido entre el cliente y el servidor se especifica después.

- Comunicación
  - Con servidor: Vía TCP.
  - Con cliente: Vía TCP(control) y UDP(streaming de audio).

- Servidor principal

  - Este programa gestiona las conexiones de los clientes al sistema. Hace las veces de mediador entre los clientes para que intercambien el audio por streaming(indicando qué cliente tiene la canción que pide para establecer una conexión con él). Además gestiona la música actualmente compartida por los usuarios permitiendo la búsqueda de canciones de otros usuarios. Una vez conectados los clientes, pueden realizar operaciones con el servidor(búsqueda de canciones y usuarios y gestión de playlists, entre las que cabe destacar la creación, carga y borrado de playlists). También se encarga de la comunicación con el almacén de perfiles, cada vez que se trabaje con playlists.
  - Comunicación
    - Con cliente: Vía TCP.
    - Con Almacenes: Vía RMI.

- Cliente web

  - El cliente web simplemente representa al usuario de la página web del sistema, que puede tanto registrarse como visualizar la información correspondiente a todos los grupos que se están compartiendo en ese momento en el sistema.
  - Comunicación
    - Con el servidor web: Vía RMI.

- Servidor web

  - Este componente en un principio permite el registro de los usuarios para tener acceso a la aplicación. Además nos muestra la información actual del almacén de metadatos. Para ello recibe información de las canciones compartidas actualmente para poder realizar una búsqueda en tiempo real sobre dicho almacén y mostrarlas. Para cada petición sobre la página web mostraremos los vídeos, letras e imágenes. Para ello, hemos empleado RMI para obtener los datos de los almacenes, Tomcat para la página web y Servlets para las peticiones.

- Comunicación
  - Con servidor principal: Vía RMI.
  - Con almacén de metados y de perfiles: Vía RMI.
  - Con cliente web: Vía HTTP.

- Recolector
  - El recolector se encarga de buscar información referente a canciones compartidas por los usuarios del sistema. Concretamente, buscamos imágenes, vídeos normales y relacionados del artista en YouTube y, por último, las letras de las canciones.
  - Comunicación
    - Con procesador: Vía RMI.
    - Con almacén de datos sin procesar: Vía RMI.
    - Red global: Vía HTTP.

- Procesador
  - Necesitamos filtrar toda la información obtenida por los recolectores con el fin de obviar la información que hayamos obtenido y no nos interese. Para ello he utilizado un procesador para los vídeos y otro para las imágenes, siendo tratado cada tipo de información de manera independiente. Para ello, el criterio común para filtrar los contenidos ha sido el nombre del grupo compartido por alguno de los clientes del sistema, mientras que en el procesador de imágenes hay además un criterio adicional : el tamaño de la imagen obtenida. Una vez filtrados todos los contenidos, éstos se almacenarán en el almacén de metadatos procesados.
  - Comunicación
    - Con Recolector: Vía MI.
    - Con almacén metadatos procesados: Vía MI.

- Almacén de perfiles
  - El almacén de perfiles se encarga de guardar la información referente a los usuarios registrados en el sistema(userID,password) así como sus playlists o listas de canciones. En un principio, dada la necesidad de agilidad de desarrollo y la relativamente escasa diversidad de datos, optamos por implementar todos los almacenes como sistemas de archivos. Para simplificar el acceso a los datos por medio de clases especializadas, elegimos, en el caso del almacén de perfiles, crear una estructura de carpetas y archivos rígida, que se describe a continuación:

    - Id_usuario
      -password.pwd
      -playlists/
      playlist1.bin
      playlist2.bin
  - Comunicación
    - Con el servidor principal y el servidor web: Vía RMI.[2]

- Almacén de canciones sin procesar
  - Este almacén guardará, de manera no permanente, las canciones(nombre y grupo) que se están compartiendo en ese momento para que el recolector pueda cogerlas e iniciar el proceso de recolección de información.
  - Comunicación:
    - Con recolector: Vía RMI.
    - Con servidor principal: Vía RMI.

- Almacén de metadatos procesados
  - Este almacén guardará la información procesada referente a las canciones buscadas. Esta información se podrá servir vía web, adecuadamente presentada, a todos los usuarios del sistema.
  - La estructura que se utilizará en este almacén será la siguiente:
    - Id_usuario
      - ◇ Imágenes: Grupo_img.txt
      - ◇ /Letras
        1. Grupo_Cancion1.txt
        2. Grupo_Cancion2.txt
      - ◇ /Vídeos
        1. Grupo_vídeos.txt
        2. Grupo_vídeos_rel.txt
  - Comunicación:
    - Con procesador: Vía RMI.
    - Con servidor principal: Vía RMI.
    - Con servidor web: Vía RMI.

[2]RMI(Java Remote Method Invocation) es un mecanismo ofrecido por Java para invocar un método de manera remota. Daremos una descripción más profunda de su funcionamiento y utilidad en la sección de implementación.

6.3.2. Secuencia de ejecución típica del sistema

Este apartado contiene un caso de ejecución típico del sistema, lo cual hemos considerado muy útil para comprender mejor el funcionamiento del sistema y la comunicación entre los diferentes elementos del sistema por medio del siguiente ejemplo:

1. El cliente web se conecta a la página web para registrarse en el sistema, lo que hace que se guarde un nuevo usuario en el almacén de perfiles.
2. Un cliente de la aplicación se conecta al servidor principal, compartiendo su música y viendo la música compartida por otros usuarios y las playlists disponibles en el sistema.
3. El servidor principal le indica a dónde debe conectarse para escuchar la canción seleccionada.
4. Las canciones que comparte cada usuario son transmitidas al almacén de canciones sin procesar, para que el recolector pueda buscar información sobre ellas.
5. El recolector verifica periódicamente si se han compartido nuevas canciones. En caso afirmativo, por cada nuevo grupo, se lanza una instancia del mismo que pasa a buscar imágenes, vídeos y letras correspondientes a esas canciones.
6. Una vez se haya reunido toda la información, lanzamos una instancia del procesador que la filtra. Una vez filtrada toda la información, ésta pasa al almacén de metadatos procesados.
7. Por su parte, el cliente web puede acceder a esa información recolectada, que está situada físicamente en el almacén de metadatos.

6.3.3. Casos de uso

Los diagramas de casos de uso permiten mostrar la manera en que un usuario final va a interactuar con el sistema a desarrollar, sin prestar atención a cómo realizaremos su desarrollo. En este caso, he optado por llevar a cabo el proceso en dos fases principales: primero, la identificación de los actores que intervienen en el sistema y después la identificación y descripción de los casos de uso identificados a partir del análisis.

6.3.3.1. Identificación de actores

A continuación enumeramos los diferentes actores que interactúan con el sistema:

- Cliente: Representa al cliente que se conecta al sistema tanto para compartir canciones, pedir canciones al servidor(junto con su información asociada) como construir sus propias playlist.

- Servidor: Representa al servidor del sistema, encargado de mediar en la comunicación entre clientes para el intercambio de canciones por streaming como de comunicarse con los demás componentes del sistema(procesador, recolector y almacenes).
- Almacén de perfiles: Representa al Almacén encargado de guardar las canciones de las playlists que quiera guardar cada usuario del sistema, así como los datos de dichos usuarios.
- Almacén de metadatos procesados: Representa al Almacén que guardar la información asociada a cada grupo, previamente filtrada por el procesador del sistema. Dicho de otro modo, representa el almacén que contiene la información final, aquella que podrá ser accedida a través de la página web.
- Almacén de metadatos sin procesar : Representa al almacén que contiene las canciones compartidas en este momento en el sistema.
- Recolector: Representa el sistema encargado de recoger todo tipo de información de la red referente a las canciones que se están compartiendo en el sistema actualmente.
- Procesador: Representa el sistema encargado de procesar la información obtenida por el recolector, a fin de retener sólo aquella información que nos sea de utilidad.
- Cliente web : Representa al cliente web de la aplicación.
- Servidor web : Representa al servidor web de la aplicación.

6.3.3.2. Identificación y breve descripción de los casos de uso

El siguiente diagrama 6.3.3.1 muestra la primera iteración de los casos de uso del sistema. Estos diagramas son los más generales, de modo que se irá definiendo el sistema a desarrollar con más precisión conforme vayamos avanzando en las iteraciones. Para este sistema y en el primer prototipo del mismo, un diseño en tres iteraciones será suficiente para capturar adecuadamente la funcionalidad esperada.

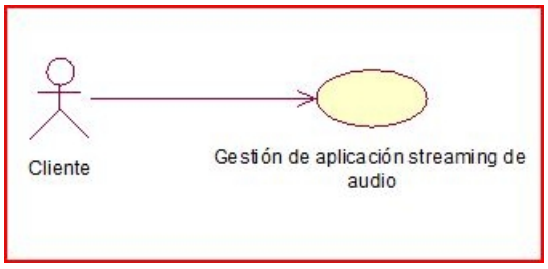


Figure 6.5: Caso de uso “Gestión aplicación streaming de audio” en Borrador

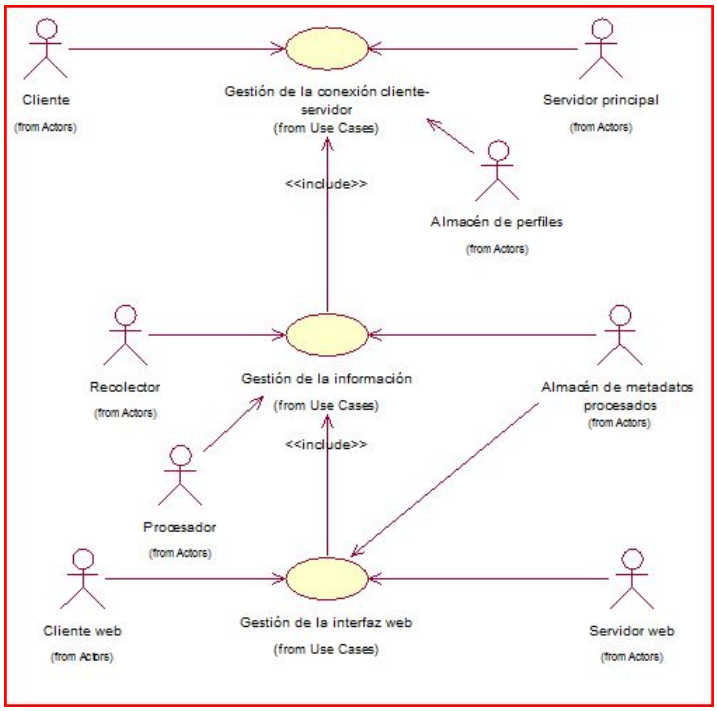


Figure 6.6: Caso de uso "Gestión aplicación streaming de audio" en Iteración 1

Esta primera iteración ya muestra cuáles son los casos de uso que debemos definir a continuación: Gestión de la conexión cliente-servidor, gestión de la información y gestión de la interfaz web. Para una mayor claridad, estudiaremos cada uno de estos casos de uso por separado, en una sección distinta.

6.3.3.3. Gestión de la conexión cliente-servidor

Dentro da gestión de la conexión cliente-servidor, podemos encontrar a su vez cuatro casos de uso, en los que participan dos actores: el cliente y el servidor principal.

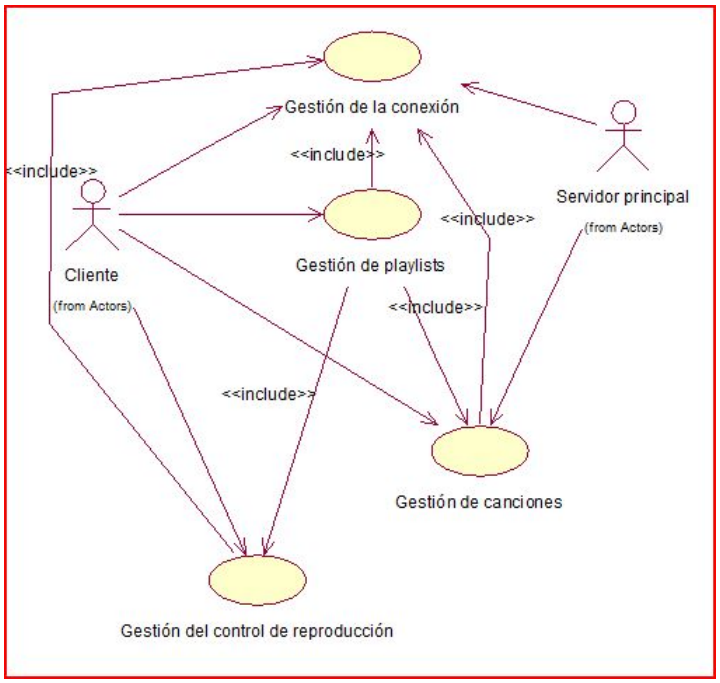


Figure 6.7: Caso de uso "Gestión de la conexión cliente-servidor" en Iteración 2

- Gestión del control de reproducción
  - Actores participantes: Cliente.
  - Explicación : Este caso de uso consistirá en llevar el control de la reproducción del stream.

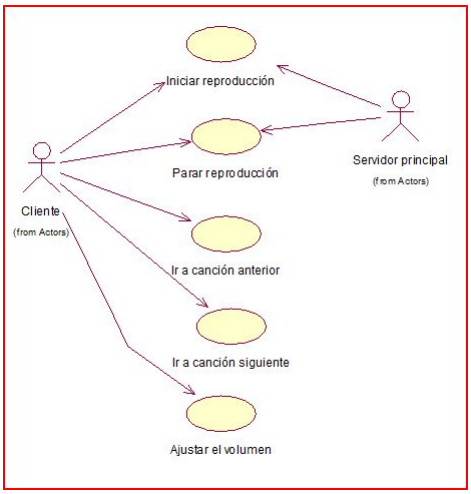


Figure 6.8: Caso de uso "Gestión del control de reproducción" en Iteración 3

En la siguiente iteración, este caso de uso se ha dividido en los siguientes casos de uso:

- Iniciar/reanudar reproducción
  - Actores participantes: Cliente, Servidor principal.
  - Explicación : Este caso de uso consistirá en iniciar la reproducción de una canción o bien en reanudarla si ha sido previamente pausada.
- Parar reproducción
  - Actores participantes: Cliente, Servidor principal.
  - Explicación : Este caso de uso consistirá en detener la reproducción de una canción.
- Ir a canción anterior
  - Actores participantes: Cliente.
  - Explicación : Desplazarse hacia la canción anterior en una lista de reproducción.
- Ir a canción siguiente
  - Actores participantes: Cliente.
  - Explicación : Desplazarse hacia la canción siguiente en una lista de reproducción.
- Ajustar el volumen
  - Actores participantes: Cliente.
  - Explicación: Subir o bajar el volumen de reproducción de las canciones.

- Gestión de la conexión
  - Actores participantes: Cliente, Servidor principal.
  - Explicación: Caso de uso que se encarga de llevar el control de la conexión entre los diferentes clientes y el servidor del sistema.

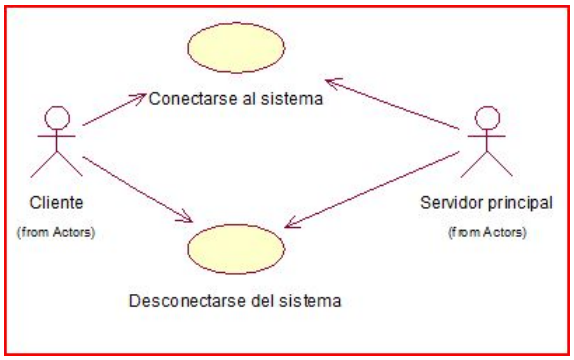


Figure 6.9: Caso de uso "Gestión de la conexión" en Iteración 3

Este caso de uso se divide en los siguientes casos de uso, en la siguiente iteración:

- Conectarse al sistema
  - Actores participantes: Cliente, Servidor principal.
  - Explicación : Este caso de uso consistirá en realizar la conexión TCP con el servidor del sistema. El servidor se encargará de asignar los parámetros de la conexión.
- Desconectarse del sistema
  - Actores participantes: Cliente, Servidor principal.
  - Explicación : Este caso de uso, simétrico al anterior, consistirá en cerrar la conexión TCP con el servidor del sistema.

- Gestión de playlists

  - Actores participantes: Cliente, Almacén de perfiles.
  - Explicación : Este caso de uso consistirá en llevar el control de los playlists de los usuarios. El cliente subirá las canciones de esas playlists y el almacén guardará todas esas playlists.

El caso de uso de gestión de playlists se dividirá a su vez en los siguientes casos de uso, tal y como podemos ver en la figura 7:

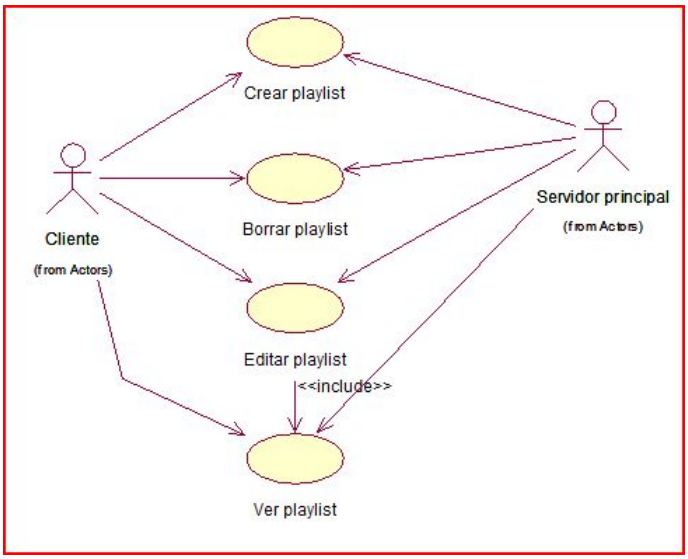


Figure 6.10: Caso de uso "Gestión de playlists" en Iteración 3

- Crear playlist
  - Actores participantes: Cliente, Servidor principal.
  - Explicación: Creación de una nueva lista de canciones o playlists a partir de las canciones que seleccione el usuario dentro de todas las que están siendo compartidas en el sistema.
- Borrar playlist
  - Actores participantes: Cliente, Servidor principal.
  - Explicación: Eliminar una lista de canciones(sin que esto suponga una eliminación de las canciones que la componen).
- Editar playlist
  - Actores participantes: Cliente, Servidor principal.
  - Explicación: Modificar el contenido de una playlist ya existente en el sistema.

- Ver playlist
  - Actores participantes: Cliente, Servidor principal.
  - Explicación: Visualizar el contenido de una playlist ya existente en el sistema.

- Gestión de canciones

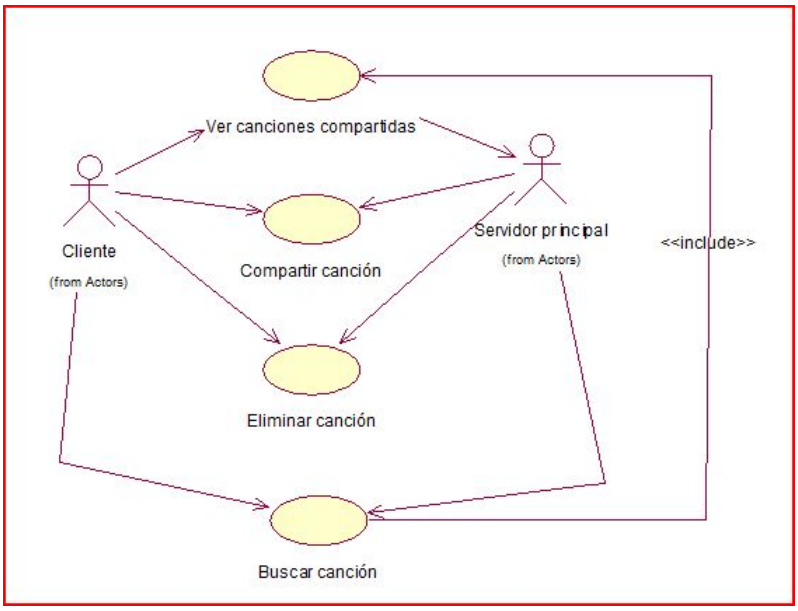


Figure 6.11: Caso de uso "Gestión de canciones" en Iteración 3

- Compartir canción
  - Actores participantes: Cliente, Servidor principal.
    - Explicación : Este caso de uso consistirá en procesar los ficheros que comparte cada cliente, de modo que el cliente subirá dichos archivos y el servidor se encargará de guardar en el almacén las canciones que se han subido.
  - Eliminar canción
    - Actores participantes: Cliente, Servidor principal.
    - Explicación : Este caso de uso consistirá en eliminar del almacén de metadatos sin procesar las canciones que haya dejado de compartir un usuario.
  - Buscar canción
    - Actores participantes: Cliente, Servidor principal.

- Explicación : Se trata de procesar las búsquedas de canciones que hagan los usuarios. El servidor se conecta con este caso de uso para, una vez vistas las búsquedas que se producen, buscar qué cliente ha subido esa canción para la comunicación posterior entre clientes.

6.3.3.4. Gestión de la información

La sección de la gestión de información cuenta, en su segunda iteración, con el siguiente diagrama de casos de uso:

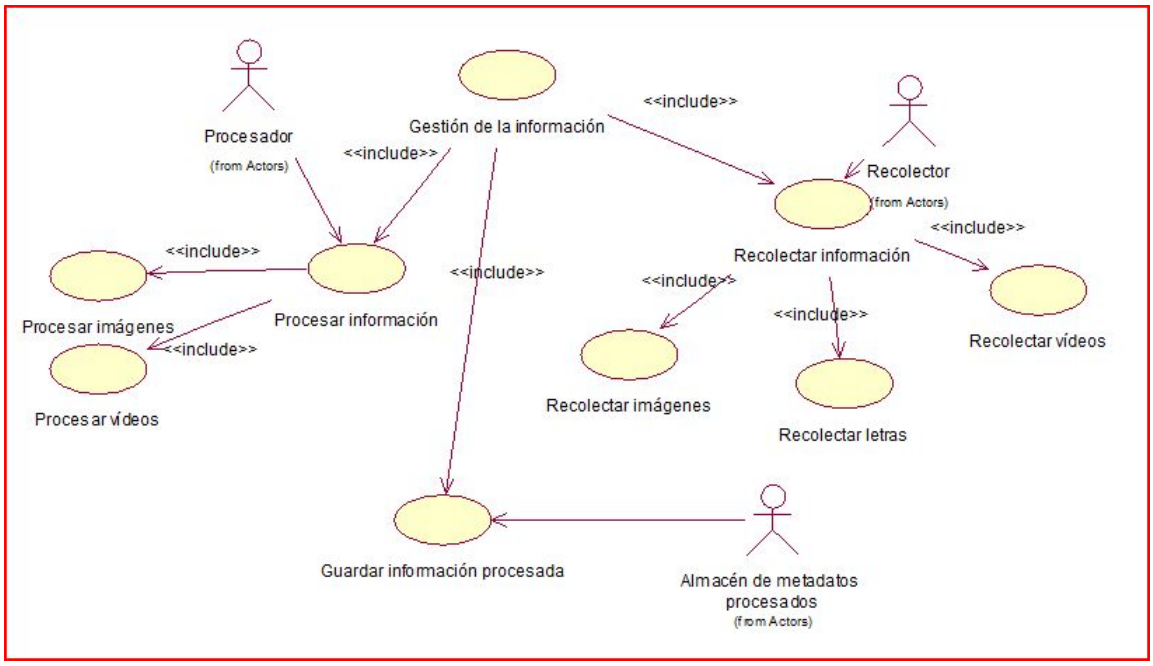


Figure 6.12: Caso de uso "Gestión de la información" en Iteración 2

- Comprobar grupos compartidos
  - Actores participantes: Almacén de metadatos sin procesar.
  - Explicación: Comprobación de los grupos actualmente compartidos en el sistema, consultando en el almacén de metadatos sin procesar.
- Recolectar información
  - Actores participantes: Recolector.

- Explicación : Caso de uso encargado de recolectar toda la información referente al grupo y canción que se hayan subido por parte de clientes al servidor.

- Procesar información
  - Actores participantes: Procesador.
  - Explicación: Procesado de toda la información recolectada previamente.

- Guardar información procesada
  - Actores participantes: Almacén de metadatos procesados.
  - Explicación: Almacenamiento físico, siguiendo la estructura ya comentada, de toda la información recolectada y procesada, dejando esa información lista para la consulta por parte de los usuarios de la aplicación web.

- Recolectar imágenes
  - Actores participantes: Recolector.
  - Explicación: Caso de uso encargado de la recolección de imágenes de los grupos que están siendo compartidos en el sistema actualmente.

- Recolectar vídeos
  - Actores participantes: Recolector.
  - Explicación: Caso de uso encargado de la recolección de vídeos de los grupos que están siendo compartidos en el sistema actualmente.

- Recolectar letras
  - Actores participantes: Recolector.
  - Explicación: Caso de uso encargado de la recolección de las letras de las canciones que se estén compartiendo en el sistema en un momento dado.

- Procesar imágenes:
  - Actores participantes: Procesador.
  - Explicación : Consiste en llevar a cabo un filtrado para todas las imágenes recolectadas, según una serie de criterios que serán explicados con más detalle en la sección de implementación.

- Procesar vídeos :
  - Actores participantes: Procesador.
  - Explicación : Consiste en llevar a cabo un filtrado para todos los vídeos recolectados, según una serie de criterios que serán explicados con más detalle en la sección de implementación.

6.3.3.5. Gestión de la interfaz web

La siguiente figura ilustra el diagrama de casos de uso de la gestión de la interfaz web, en su segunda iteración.

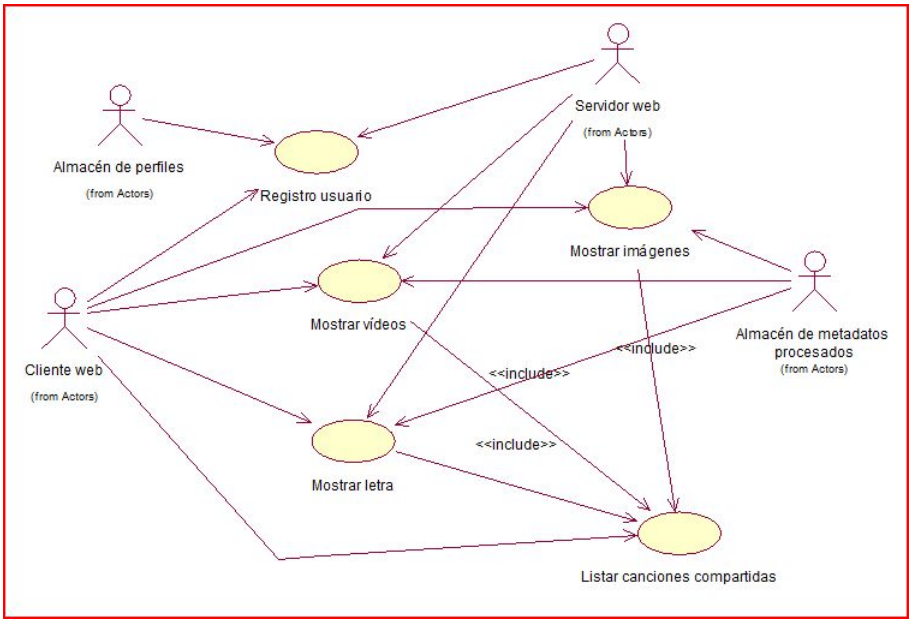


Figure 6.13: Caso de uso "Gestión de la interfaz web" en Iteración 2

- Registro usuario
  - Actores participantes:Cliente web, Servidor web, Almacén de perfiles.
  - Explicación: Caso de uso que controla el registro de un usuario en el sistema a través de la página web.
- Mostrar imágenes
  - Actores participantes: Cliente web, Almacén de metadatos procesados.

  - Explicación: Caso de uso que se encarga de acceder a las imágenes del grupo seleccionado por el usuario en el almacén de metadatos procesados y generar el código html adecuado para mostrar claramente dichas imágenes.

- Mostrar vídeos

  - Actores participantes: Cliente web, Almacén de metadatos procesados.
  - Explicación: Caso de uso que accede a los vídeos almacenados en el almacén de metadatos procesados y genera el código html adecuado para mostrar los vídeos del grupo seleccionado por el cliente web.

- Mostrar letras

  - Actores participantes: Cliente web, Almacén de metadatos procesados.
  - Explicación: Caso de uso que accede a las letras de canciones almacenados en el almacén de metadatos procesados para un canción de un grupo seleccionada por el usuario y genera el código html adecuado para una adecuada visualización de esa letra.

- Listar canciones compartidas

  - Actores participantes: Almacén de metadatos sin procesar.
  - Explicación : Caso de uso que se encarga de crear y mostrar vía web un listado con todas las canciones actualmente compartidas en el sistema, de modo que el usuario pueda seleccionar ampliar la información de cualquiera de ellas.

6.3.4. Diagramas de clases de análisis

Las clases de análisis representan una abstracción de una o varias clases y/o subsistemas en el diseño del sistema. Son diagramas de tipo estático que describen a alto nivel la arquitectura del sistema, centrándose en los requisitos funcionales y especificando el comportamiento del sistema mediante artefactos con un nivel de abstracción muy alto.

Esta sección contiene diagramas de clases de análisis para la mayoría de los casos de uso del sistema(no incluimos algunos por tener un comportamiento semejante), lo cual nos permite dar una descripción a muy alto nivel del comportamiento del sistema, en conjunto con los diagramas de secuencia de la sección siguiente. Para ello, dividiremos esta sección en tres subsecciones, correspondientes a la gestión de la conexión cliente-servidor, la gestión de la información y la gestión de la interfaz web.

6.3.4.1. Gestión de la conexión cliente-servidor

En el diagrama de clase que podemos ver abajo mostramos cuáles son las clases principales que participan en la conexión al sistema. En ella participan tanto el cliente(a través de la interfaz gráfica del programa cliente), así como el servidor principal, que trabaja en la coordinación de esa conexión, guardando los clientes conectados al sistema actualmente.

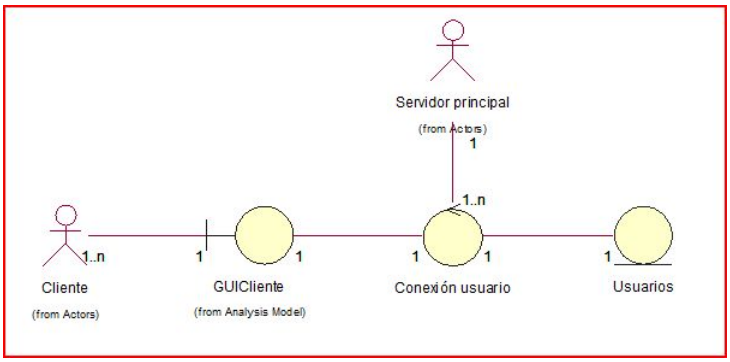


Figure 6.14: Diagrama de clases de análisis Conectarse al sistema, iteración 3

Por otro lado, el proceso de la desconexión de los usuarios del sistema es análoga en comportamiento, tal y como podemos ver en la siguiente figura:

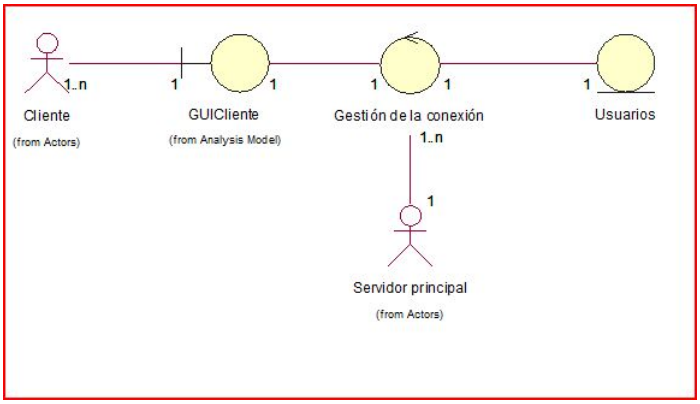


Figure 6.15: Diagrama de clases de análisis Desconectarse del sistema, iteración 3

A continuación podremos ver los diagramas de clases relacionados con la gestión de las playlists. El funcionamiento es similar para todos ellos. Podemos ver que el encargado de la gestión de las playlist es el cliente, que a través de la

interfaz gráfica tiene la opción de realizar todas las operaciones definidas en los requerimientos del sistema:

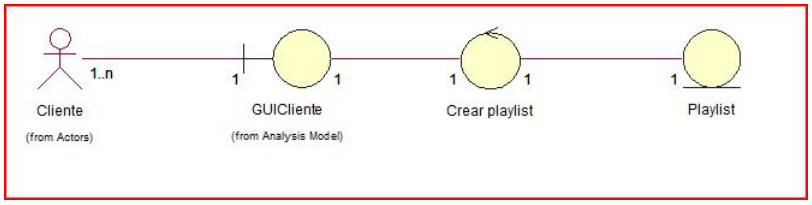


Figure 6.16: Diagrama de clases de análisis Crear playlist, iteración 3

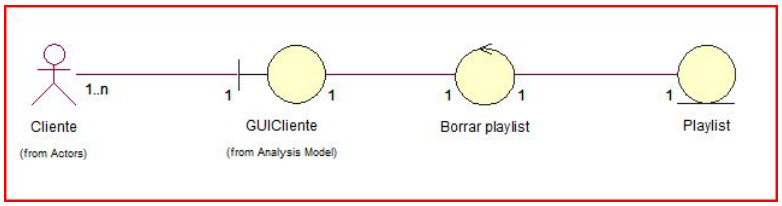


Figure 6.17: Diagrama de clases de análisis Borrar playlist, iteración 3

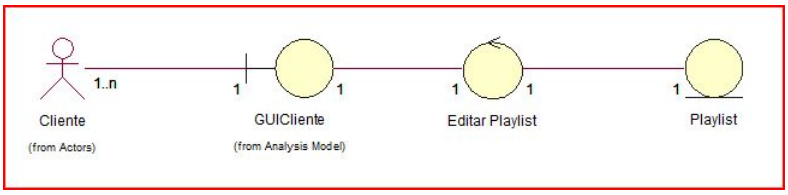


Figure 6.18: Diagrama de clases de análisis Editar playlist, iteración 3

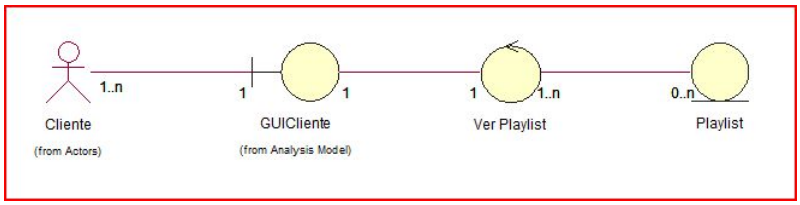


Figure 6.19: Diagrama de clases de análisis Ver playlist, iteración 3

Veamos ahora el diseño de clases de análisis en torno a la gestión de las canciones. Todos ellos cuentan con una estructura semejante, al no haber de momento diferentes roles o privilegios entre los usuarios, todos acceden con una

misma interfaz gráfica e interactúan con el servidor, de modo que se actualiza la información referente a las canciones compartidas en el sistema, para cualquiera de los cuatro casos.

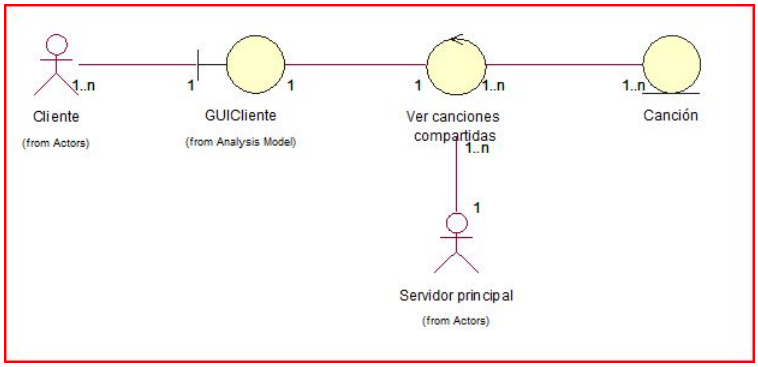


Figure 6.20: Diagrama de clases de análisis Ver canciones compartidas, iteración 3

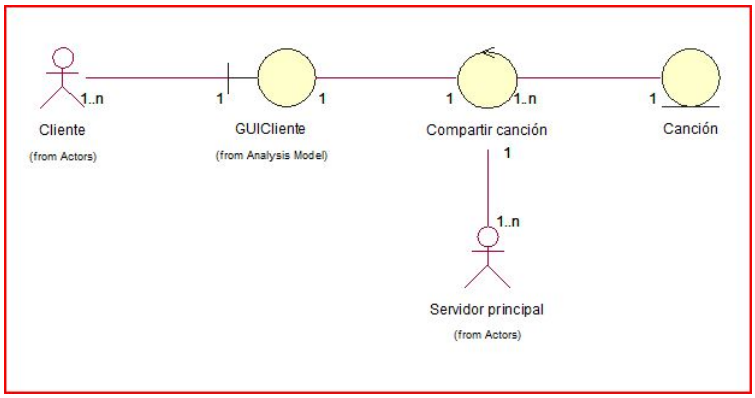


Figure 6.21: Diagrama de clases de análisis Compartir canción, iteración 3

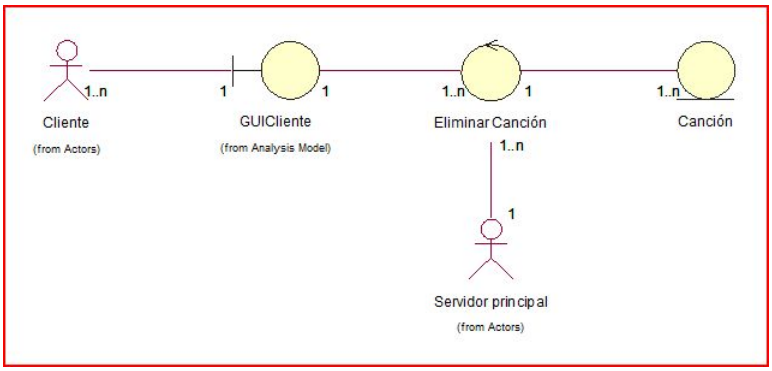


Figure 6.22: Diagrama de clases de análisis Eliminar canción, iteración 3

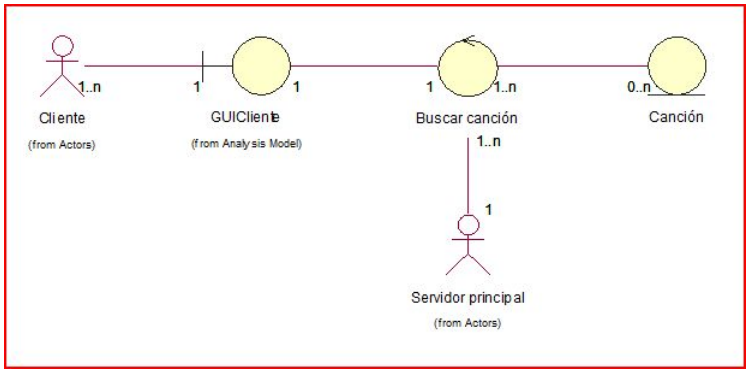


Figure 6.23: Diagrama de clases de análisis Buscar canción, iteración 3

Pasamos a estudiar la gestión del control de reproducción. El control de estos casos de uso se hace en todos los casos a través de la interfaz gráfica. En el caso de los casos de uso correspondientes a iniciar y detener la reproducción, podemos ver que cuentan con la mediación del servidor principal. Esto se debe a que éste debe iniciar y detener, respectivamente, el flujo de datos por UDP, abriendo y cerrando una conexión entre el cliente que solicita la canción y aquel que la esté compartiendo. El resto de los casos son más simples, una vez negociado el intercambio de audio, el control del volumen y el avance y retroceso a través de listas de reproducción se lleva a cabo simplemente a partir de la interfaz cliente, con una serie de clases de control que trabajan con los diferentes eventos posibles.

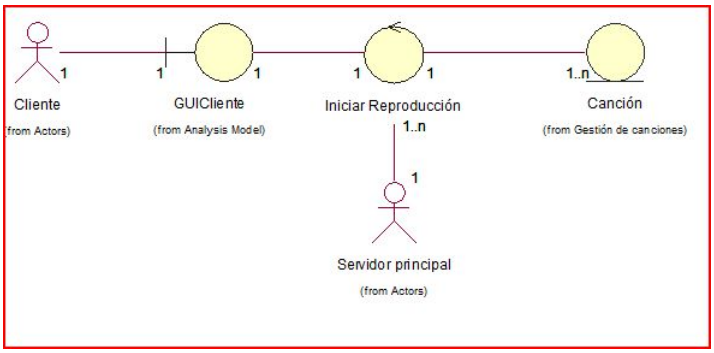


Figure 6.24: Diagrama de clases de análisis Iniciar reproducción, iteración 3

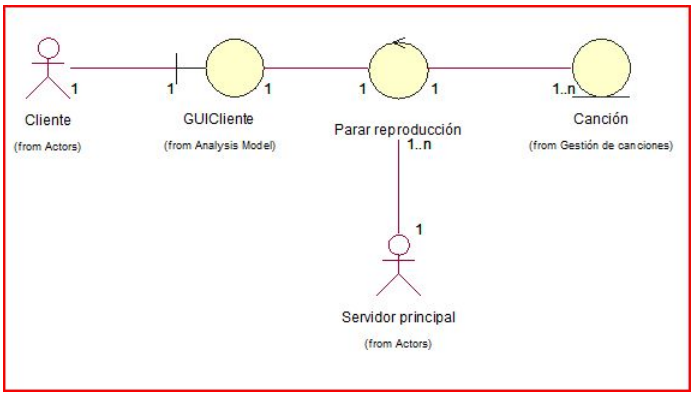


Figure 6.25: Diagrama de clases de análisis Parar reproducción, iteración 3

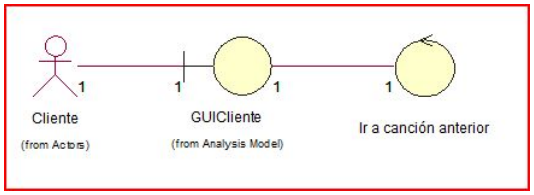


Figure 6.26: Diagrama de clases de análisis Ir a canción anterior, iteración 3

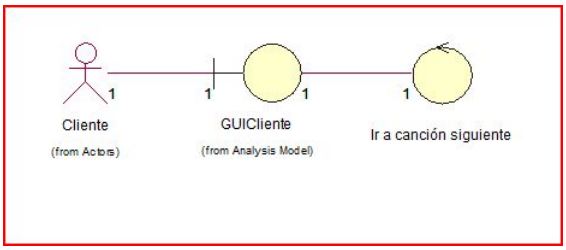


Figure 6.27: Diagrama de clases de análisis Ir a canción siguiente, iteración 3

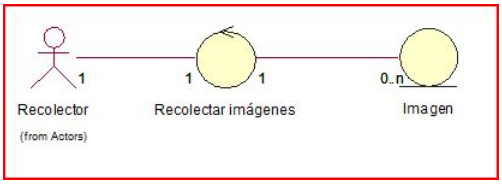


Figure 6.28: Diagrama de clases de análisis Ajustar el volumen, iteración 3

6.3.4.2. Gestión de la información

Esta sección contiene los diagramas de clases de análisis correspondientes a la sección que gestiona la información que se recolecta en el sistema. Para ello, hemos creado un diagrama por cada caso de uso de la gestión de la información 6.3.3.4.

En primer lugar, analicemos los casos de uso correspondientes a la recolección de la información. En todos ellos, el recolector es el que se encarga de iniciar el proceso de búsqueda de esa información, ya se trate de imágenes, vídeos o letras, guardando dicha información localmente, lo cual reflejamos con las clases de entidad imagen, vídeos o letras.

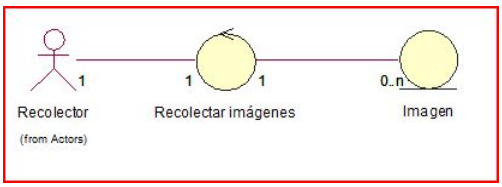


Figure 6.29: Diagrama de clases de análisis Recolectar imágenes, iteración 2

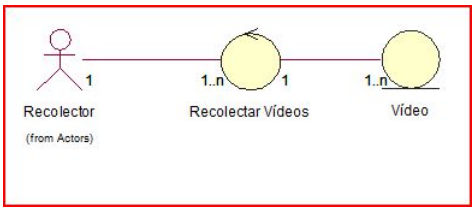


Figure 6.30: Diagrama de clases de análisis Recolectar vídeos, iteración 2

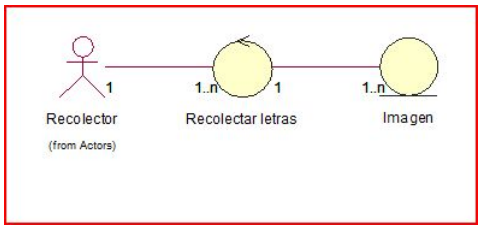


Figure 6.31: Diagrama de clases de análisis Recolectar letras, iteración 2

El procesamiento de la información reunida es un proceso bastante semejante, abstrayéndonos de momento del comportamiento interno de la clase de control: el procesador lanza el procesamiento de la información, lo cual tiene como efecto, una vez finalizado ese tratamiento de la información, el almacenamiento físico de la misma. Un detalle a tener en cuenta es que las letras no necesitan ningún filtrado.

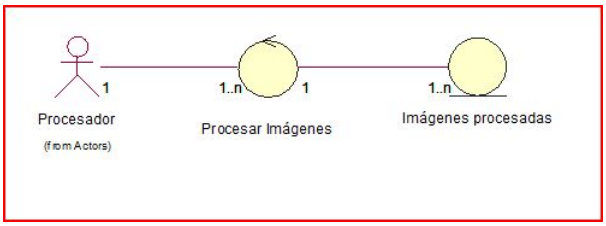


Figure 6.32: Diagrama de clases de análisis Procesar imágenes, iteración 2

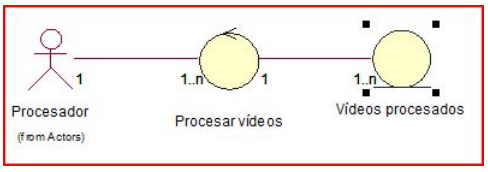


Figure 6.33: Diagrama de clases de análisis Procesar vídeos, iteración 2

Por último, tenemos el caso de uso encargado de guardar la información procesada. La función de este caso de uso es guardar en el almacén de metadatos procesados aquella información definitiva, ya procesada satisfactoriamente. Para ello, hay que reunir toda la información que hayan guardado localmente los procesadores y desplazarla al almacén, eliminando después toda la información local de los procesadores y los recolectores, que no tiene ya ninguna utilidad.

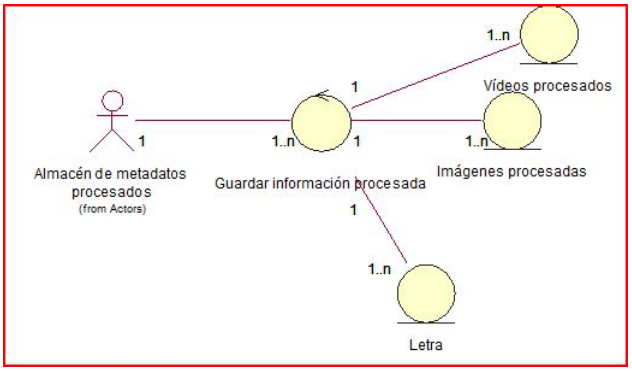


Figure 6.34: Diagrama de clases de análisis Procesar vídeos, iteración 2

6.3.4.3. Gestión de la interfaz web

De acuerdo con el diagrama de casos de uso 6.3.3.5, hemos creado diagramas de clases de análisis para cada uno de esos casos de uso.

El registro de usuario la realiza el Cliente web, a través de la interfaz web que le proporcionamos. En el proceso de control interno, el servidor web se encarga de tratar cada proceso de registro por separado, para después actualizar los datos correspondientes a los usuarios registrados, sito en el almacén de perfiles.

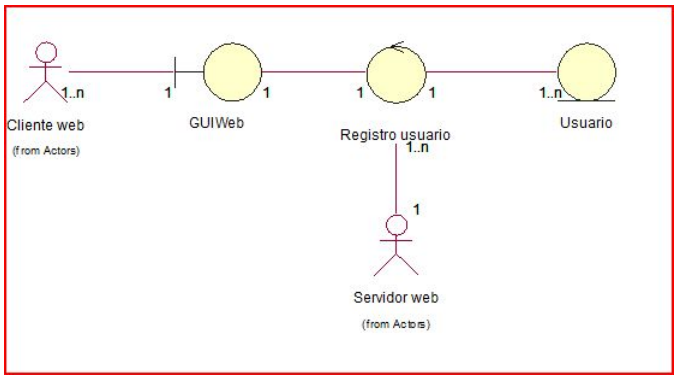


Figure 6.35: Diagrama de clases de análisis Registro usuario, iteración 2

Por otro lado, tenemos los casos de uso centrados en mostrar la información que ofrecemos a los usuarios del sistema. Se trata de crear el código necesario para la generación dinámica de estos contenidos, según la voluntad del cliente web. El proceso, si nos abstraemos lo suficiente, es muy semejante para todos los tipos de información, independientemente de cómo se realice internamente: el cliente web solicita que se le muestre la información sobre un grupo o canción, lo que hace que el servidor web acceda al almacén de metadatos procesados para acceder a esa información y construir la página web con esa información, debidamente presentada.

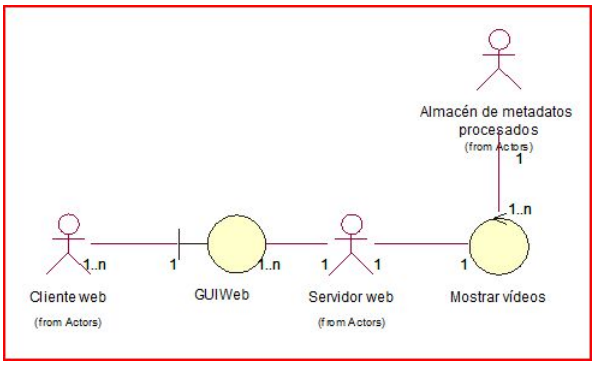


Figure 6.36: Diagrama de clases de análisis Mostrar imágenes, iteración 2

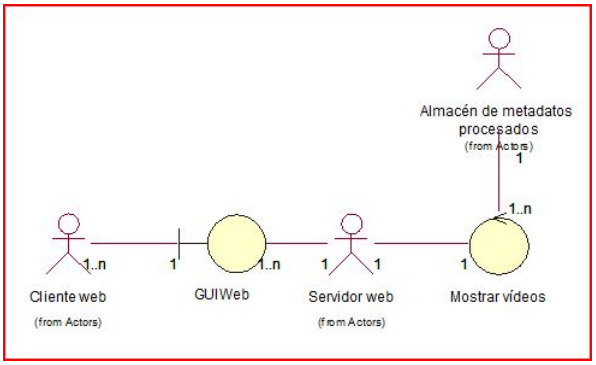


Figure 6.37: Diagrama de clases de análisis Mostrar vídeos, iteración 2

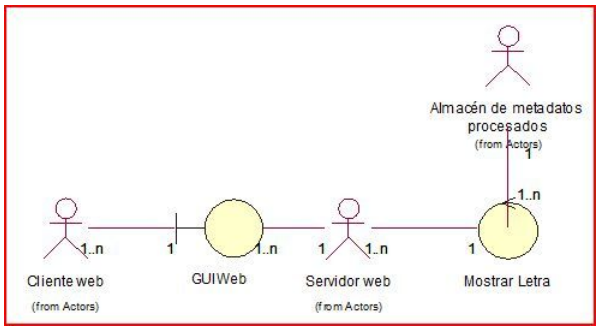


Figure 6.38: Diagrama de clases de análisis Mostrar letras, iteración 2

6.3.5. Diagramas de secuencia

Los diagramas de secuencia son un tipo de diagrama usado para modelar la interacción entre los objetos en un sistema. Pone especial énfasis en el orden y el momento en el que se envían los mensajes. Al contrario de lo que ocurre con los casos de uso, los diagramas de secuencia sí contienen detalles sobre la implementación del sistema, incluyendo los objetos que componen a éste y los mensajes pasados entre ellos. Dentro del proceso unificado, se modela un diagrama de secuencia para cada caso de uso.

6.3.5.1. Gestión de la conexión cliente-servidor

GESTIÓN DE LA CONEXIÓN

La conexión y desconexión de clientes al sistema funciona en base a un pequeño protocolo de implementación propia que describiremos en la sección de implementación. En este punto, resulta conveniente explicar que todos los casos de uso de esta sección los llevamos a cabo mediante el intercambio de objetos que contienen texto de diferentes comandos y objetos que contienen respuestas con el resultado de la operación realizada.

En el caso de la conexión de clientes al sistema, la clase de control recibe los datos y los valida, para después establecer una conexión con el servidor principal.

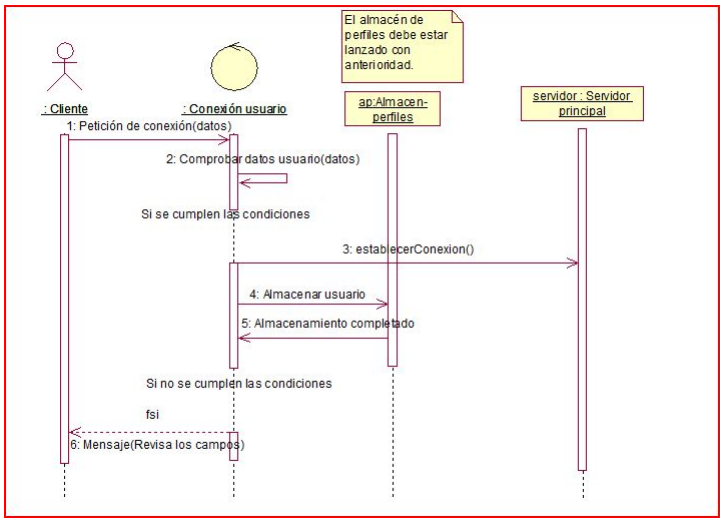


Figure 6.39: Diagrama de secuencia Conectarse al sistema, iteración 3

El proceso para la desconexión de un usuario es semejante. Una vez que el cliente solicita desconectarse del sistema, creamos un objeto para la operación del protocolo correspondiente a la desconexión. Ese objeto es enviado al servidor, que verifica el mensaje de ese objeto y pasa a cerrar la conexión con ese cliente, marcando después a ese usuario como desconectado dentro del almacén de perfiles.

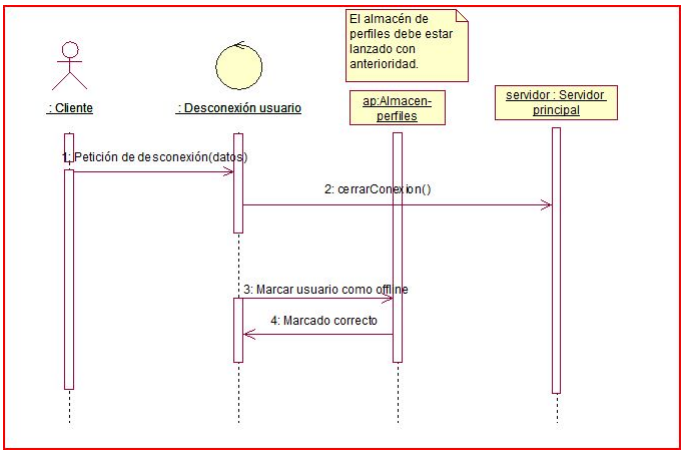


Figure 6.40: Diagrama de secuencia Desconectarse del sistema, iteración 3

GESTIÓN DE PLAYLISTS

A continuación vemos cuáles son las interacciones entre objetos para la gestión

de las playlists. Esta gestión también la llevamos a cabo con el protocolo citado, que contiene un tipo de mensaje para cada operación que nos interesa para trabajar con las playlist.

La creación de playlists se inicia cuando un usuario selecciona una serie de canciones y solicita crear una playlist con ellas, asignándole un nombre. La clase de control se encarga de construir el objeto de la petición de creación para el servidor, que envía a su vez un mensaje al almacén de perfiles para que verifique esa lista y la almacene. En caso de éxito, devolvemos un objeto con la confirmación de que todo ha ido bien. Por contra, si hay algún problema, devolvemos un objeto que refleje esa situación.

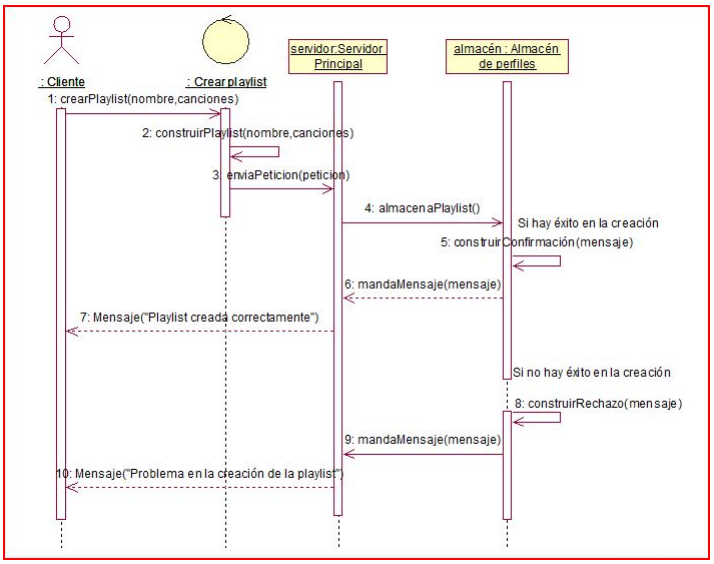


Figure 6.41: Diagrama de secuencia Crear playlist, iteración 3

Por otro lado, el borrado de playlists es semejante, con la diferencia de los objetos construidos, que trabajarán con la operación de borrado.

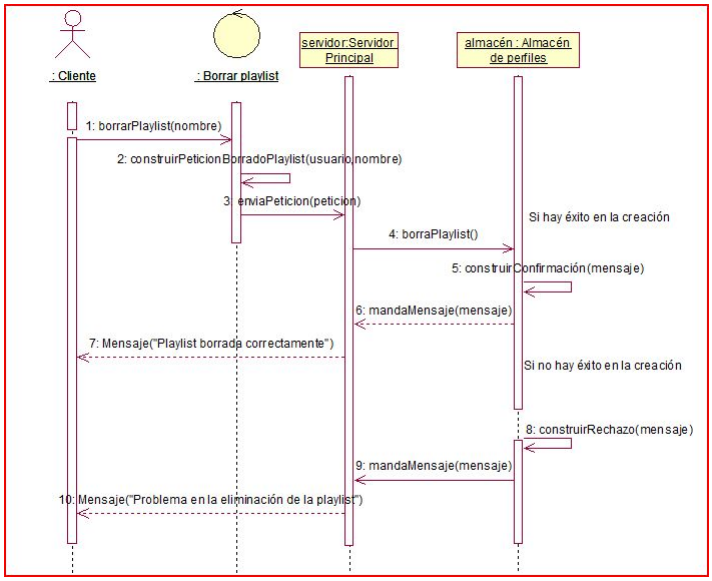


Figure 6.42: Diagrama de secuencia Borrar playlist, iteración 3

La edición de playlists se inicia cuando el cliente solicita que se le muestren todas las listas de canciones disponibles, para después modificar la lista de canciones que la componen y guardarla con otro nombre dentro del almacén de perfiles. Este proceso también lo realizaremos mediante objetos que describen las diferentes acciones y mensajes sobre el éxito o el fracaso de las operaciones.

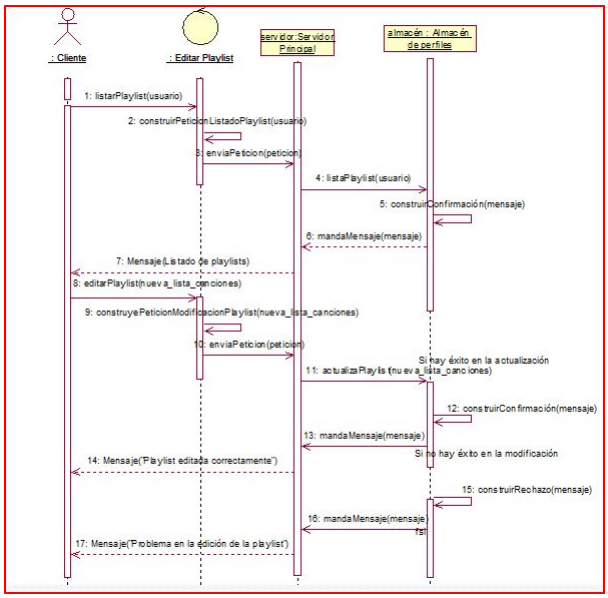


Figure 6.43: Diagrama de secuencia Editar playlist, iteración 3

Por último, el caso de uso de visualizar playlists, semejante a los anteriores y que utiliza un comando del protocolo para ver una lista de canciones.

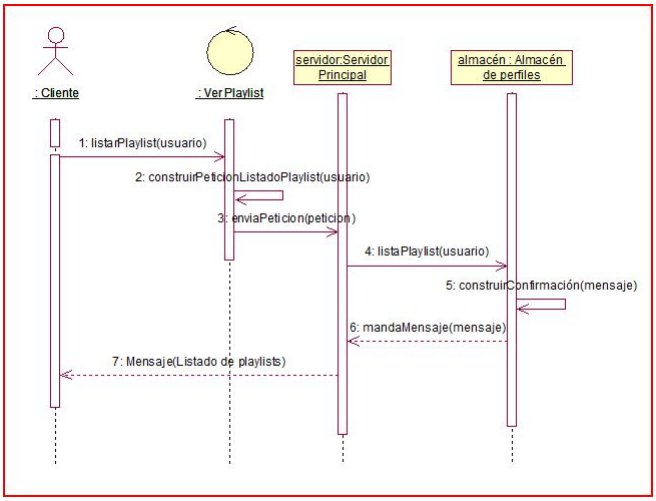


Figure 6.44: Diagrama de secuencia Ver playlist, iteración 3

GESTIÓN DE CANCIONES

Dentro de la gestión de las canciones, encontramos el caso de uso de visual-

ización de las canciones que están siendo compartidas. Cada vez que un cliente solicite ese listado de canciones, se construye un objeto que solicite al servidor esa lista. El servidor construye dicha lista y la devuelve al usuario a través de la interfaz gráfica.

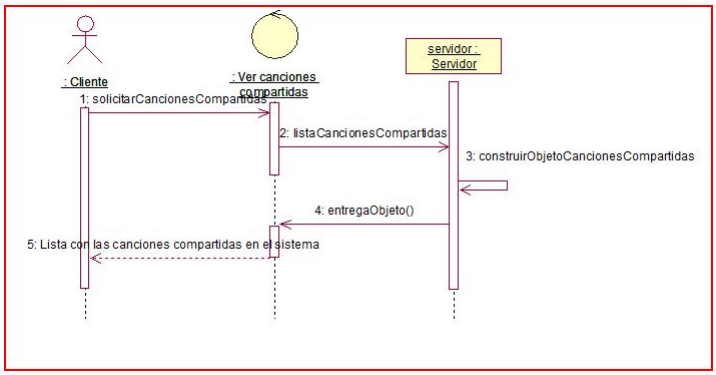


Figure 6.45: Diagrama de secuencia Ver canciones compartidas, iteración 3

A la hora de compartir canciones, cuando un usuario añada nuevos ficheros en su carpeta compartida, se creará un objeto con la nueva lista de canciones compartidas y se mandará un mensaje al servidor para que registre dicho objeto. Por último, el servidor construirá un objeto con la respuesta.

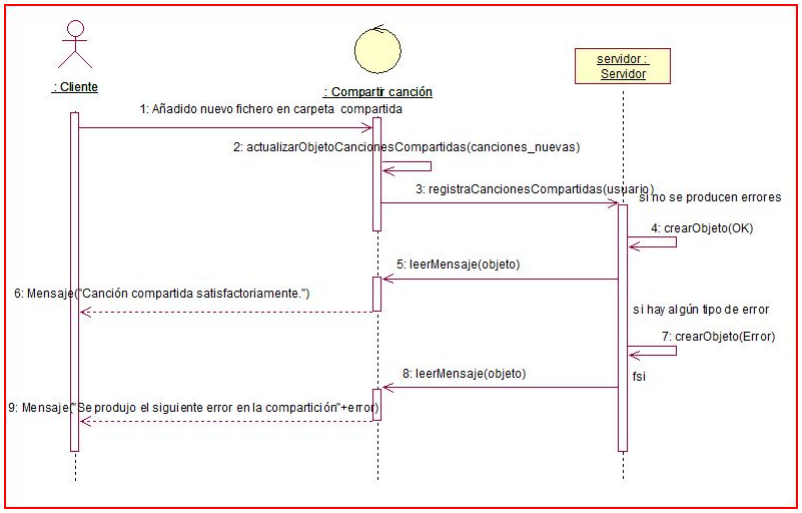


Figure 6.46: Diagrama de secuencia Compartir canción, iteración 3

El caso de uso de borrar una canción que se esté compartiendo es semejante al anterior, con la única diferencia de los comandos internos de los objetos.

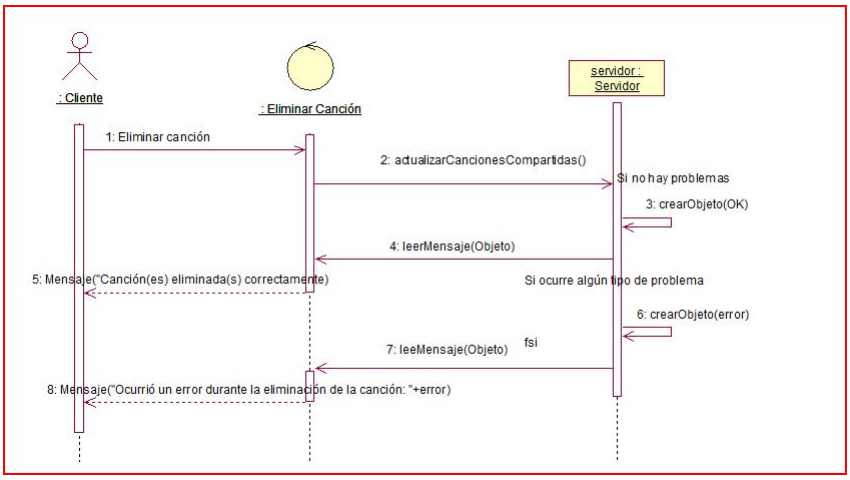


Figure 6.47: Diagrama de secuencia Borrar canción, iteración 3

La búsqueda de canciones es semejante a los casos anteriores, con la diferencia de que el objeto que se envía al servidor contiene el comando de búsqueda de canciones.

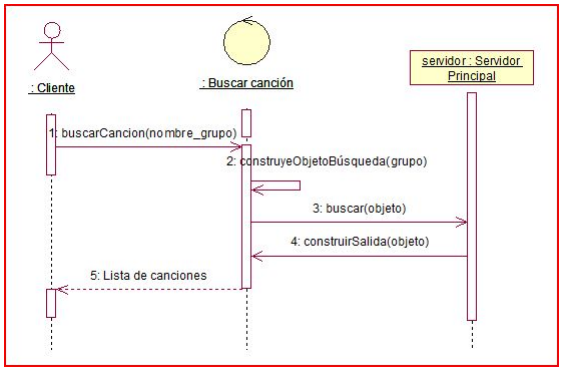


Figure 6.48: Diagrama de secuencia Buscar canción, iteración 3

GESTIÓN DEL CONTROL DE REPRODUCCIÓN

Cuando un cliente desea reproducir una canción, creamos un objeto con el comando correspondiente a la reproducción y se la enviamos al servidor, que se encarga de buscar el cliente que comparte la canción solicitada. Una vez hecho esto, creamos un objeto de respuesta con el resultado de la operación y si no ha habido ningún problema, comienza el flujo UDP de la música entre el usuario que solicita la canción y el que la comparte.

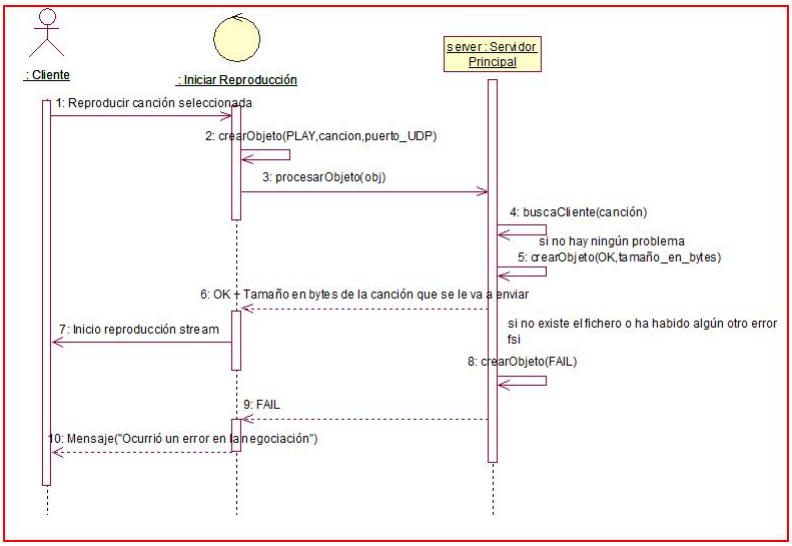


Figure 6.49: Diagrama de secuencia Iniciar reproducción, iteración 3

Para parar la reproducción, una vez el usuario pulse el botón correspondiente, construimos y enviamos al servidor un objeto que contiene la petición de detener el flujo de datos. Una vez más, construimos un objeto de respuesta que informa a la clase de control del resultado de la operación.

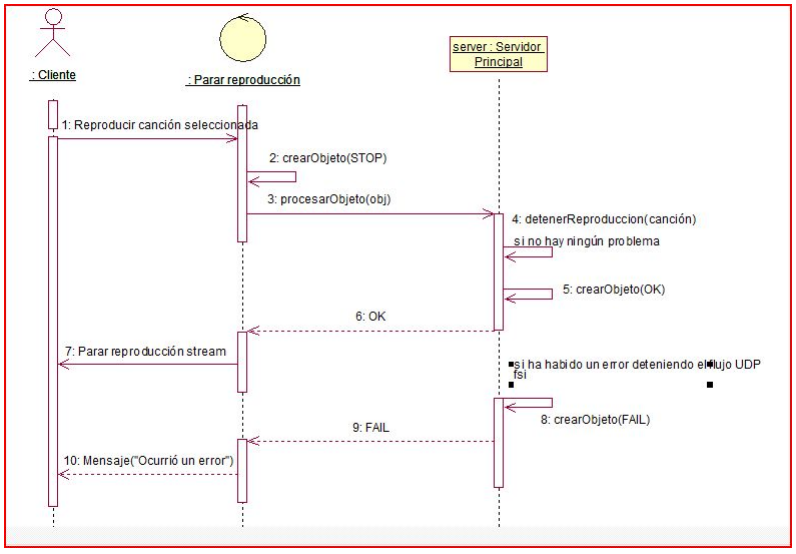


Figure 6.50: Diagrama de secuencia Parar reproducción, iteración 3

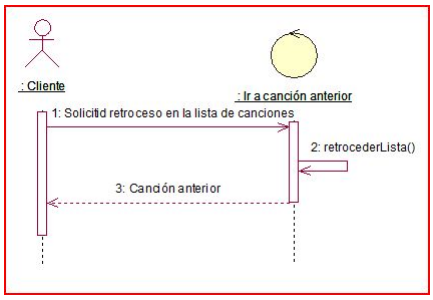


Figure 6.51: Diagrama de secuencia Ir a canción anterior, iteración 3

La secuencia de mensajes para ir a la canción anterior y la canción siguiente se realiza mediante un método de la clase control que se encarga de avanzar o retroceder en la lista de reproducción que esté escuchando el cliente.

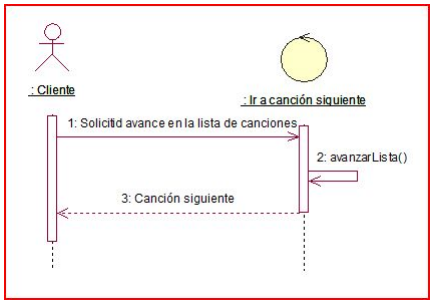


Figure 6.52: Diagrama de secuencia Ir a canción siguiente, iteración 3

Por último, el ajuste de volumen se realiza gracias a un método de la clase de control que se encargará de aumentar o disminuir el volumen al que se reproduce la música en base al scroll que controla el usuario.

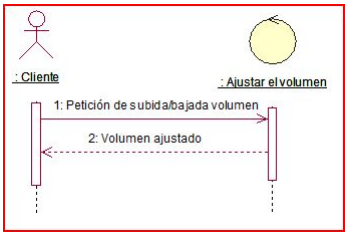


Figure 6.53: Diagrama de secuencia Ajustar el volumen, iteración 3

6.3.5.2. Gestión de la información

RECOLECTAR INFORMACIÓN

La recolección de imágenes se realiza mediante un hilo principal que lanzamos cuando el recolector detecta que han sido compartidas canciones de algún grupo cuya información no tenemos. Con los vídeos y las letras ocurre algo parecido. Posteriormente, en la sección de la implementación, describiremos más detalladamente cómo se lleva a cabo este proceso.

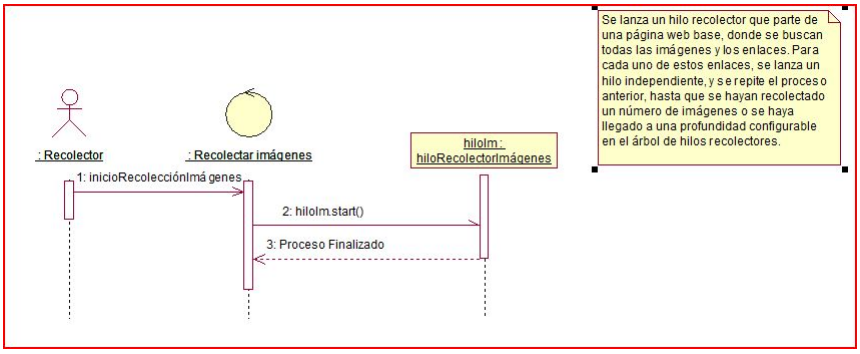


Figure 6.54: Diagrama de secuencia Recolectar imágenes, iteración 2

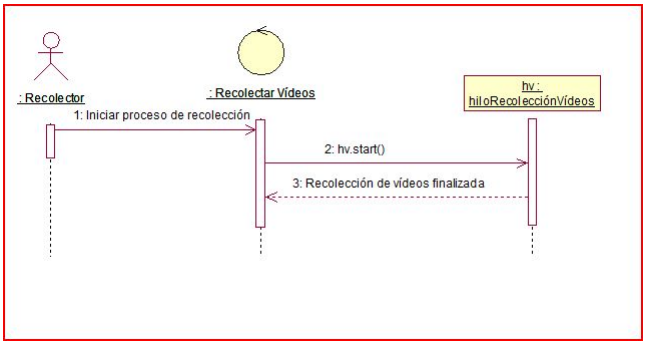


Figure 6.55: Diagrama de secuencia Recolectar vídeos, iteración 2

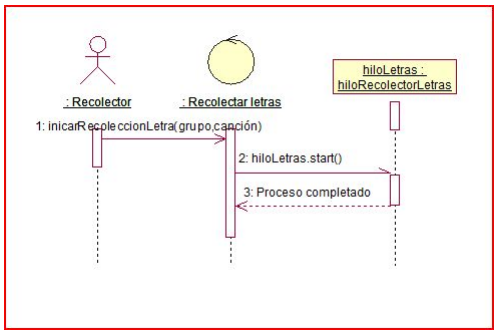


Figure 6.56: Diagrama de secuencia Recolectar letras, iteración 2

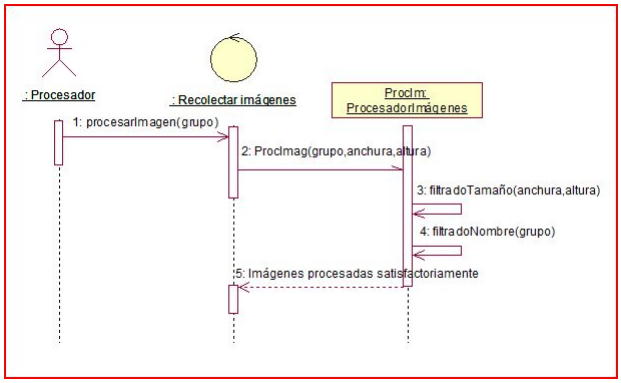


Figure 6.57: Diagrama de secuencia Procesar imágenes, iteración 2

PROCESAR INFORMACIÓN

Las imágenes y los vídeos obtenidos necesitan ser procesados. En ambos casos, para cada vídeo, se lanza una instancia de una clase que se encarga de filtrar esa información según una serie de criterios que expondremos después.

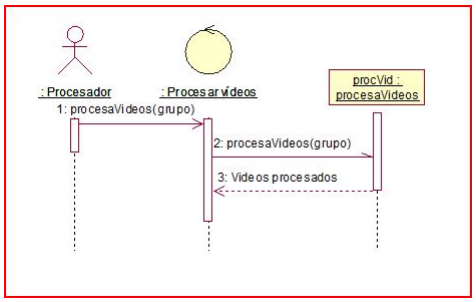


Figure 6.58: Diagrama de secuencia Procesar vídeos, iteración 2

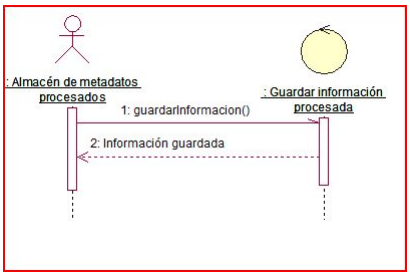


Figure 6.59: Diagrama de secuencia Guardar información procesada,iteración 2

6.3.5.3. Gestión de la interfaz web

Para el registro de los usuarios, cada vez que un usuario rellene el formulario con sus datos, el cliente web mandará esa solicitud al servidor web, que a su vez solicita a la clase de control que procese esos datos. Una vez comprobados, estos datos se registran físicamente en el almacén de perfiles, devolviendo un mensaje que informa del resultado de la operación.

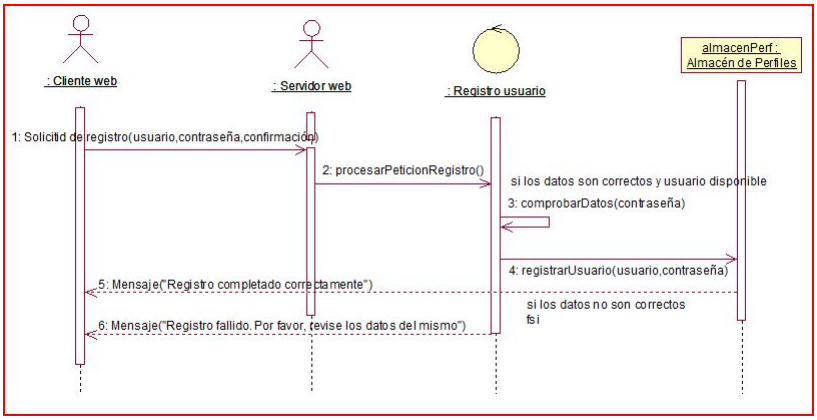


Figure 6.60: Diagrama de secuencia Registro usuario, iteración 2

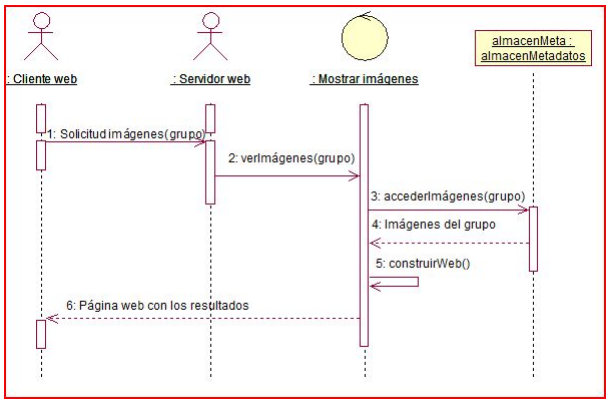


Figure 6.61: Diagrama de secuencia Mostrar imágenes, iteración 2

La muestra de los datos que soliciten los usuarios requiere que se preparen y ordenen los datos adecuadamente. De este modo, para todos los tipos de información, cada vez que un cliente web solicite información sobre un grupo, sea del tipo que sea, se accederá al almacén de metadatos para leer la información que necesitemos y construiremos una página web de forma dinámica para mostrar esos datos.

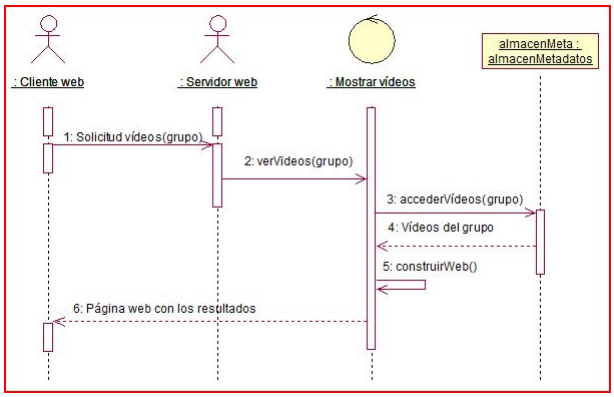


Figure 6.62: Diagrama de secuencia Mostrar vídeos, iteración 2

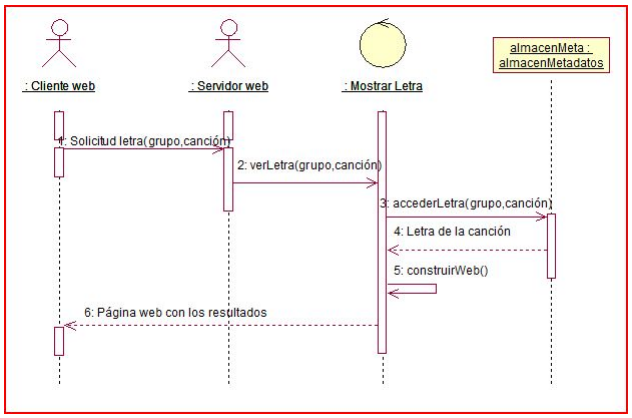


Figure 6.63: Diagrama de secuencia Mostrar letras, iteración 2

Capítulo 7

Implementación

Este capítulo tiene como objetivo proporcionar una visión general de las técnicas utilizadas para la implementación del sistema.

7.1. RMI

RMI(Java Remote Method Invocation) es un mecanismo ofrecido por Java para invocar un método de manera remota. Un detalle de importancia es que forma parte del entorno de ejecución de Java, lo que provee al desarrollador de un mecanismo simple para comunicar aplicaciones distribuidas basadas únicamente en Java. Las aplicaciones basadas en RMI suelen dividirse en dos programas, un servidor y un cliente. Un servidor RMI típico crea una serie de objetos remotos, registra esos objetos permitiendo que se los referencia de forma remota, y espera a que clientes RMI invoquen métodos de esos objetos. RMI proporciona el mecanismos mediante el que el cliente y el servidor se comunican y pasan información en ambos sentidos.

Si bien no entraré en una descripción muy detallada del funcionamiento y las avanzadas características de RMI, creo necesario dar una pequeña explicación de su funcionamiento. Las aplicaciones distribuidas basadas en RMI deben llevar a cabo las siguientes tareas:

- ***Localizar objetos remotos***. Las aplicaciones que desarrollemos pueden utilizar diferentes mecanismos para obtener referencia a los objetos remotos. La manera más sencilla(que es la utilizada en este proyecto) es registrar los objetos remotos mediante el servidor de nombres rmiregistry.
- ***Comunicarse con los objetos remotos***. Los detalles de la comunicación entre los objetos remotos son manejados por el sistema RMI. Para el programador, la comunicación entre objetos se asemeja a la utilizada normalmente en programas Java.

Sin entrar en más detalles, veamos una figura simple que ilustra el funcionamiento de una aplicación distribuida basada en RMI. El servidor que implementa los

objetos remotos invoca al rmiregistry para asociarle un nombre a un objeto remoto. El cliente busca el método remoto utilizando el nombre del objeto remoto como argumento y ejecuta uno de sus métodos.

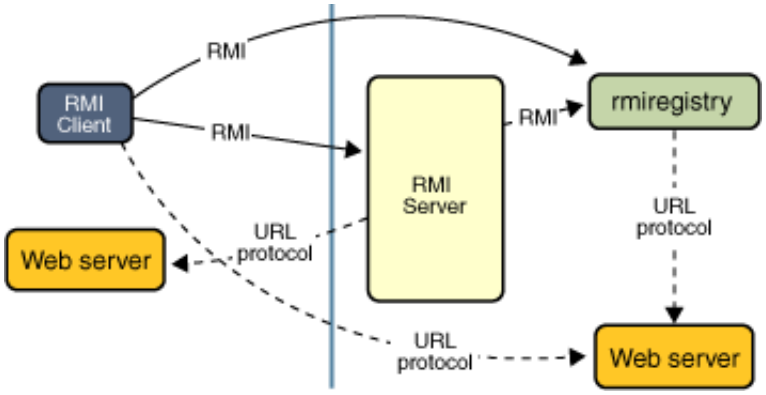


Figura 7.1: Funcionamiento básico de RMI

Como ya expuse en el diagrama de despliegue del diseño del sistema, toda la comunicación entre los distintos componentes del sistema ha sido realizada con RMI, con la excepción de la comunicación entre los clientes y el cliente y el servidor, donde hemos implementado un sencillo protocolo basado en el paso de objetos, por sencillez y eficiencia. Para ello, tal y como describimos en el diagrama de despliegue ya expuesto, los diferentes componentes se comunican entre sí, haciendo unas veces de servidor de objetos y otras veces de cliente de objetos. Adicionalmente, RMI también ha sido de gran utilidad para la automatización de todo el proceso de ejecución de la aplicación. La razón principal de su utilización ha sido su sencillez de utilización, ya que reduce notablemente la complejidad del proceso de comunicación entre los componentes de nuestra aplicación, permitiendo concentrarnos en otros aspectos del desarrollo del sistema.

7.2. Streaming de audio

Para el streaming de audio propiamente dicho, hemos utilizado el API que proporciona GStreamer, un framework multimedia libre multiplataforma que permite realizar infinidad de operaciones sobre medios muy diversos. Si bien se trata de una librería bastante compleja y que cuenta con funcionalidades muy avanzadas, la funcionalidad que nos ocupa es el streaming UDP de audio en mp3 por medio de pipelines. Las pipelines son tuberías para la transmisión de todo tipo de datos que maneja la librería, en nuestro caso audio por mp3. La librería hace uso intensivo de hilos, de manera transparente al programador, lo cual le da una gran eficiencia en tiempo de ejecución. En resumidas cuentas, hemos creado pequeños métodos para iniciar la reproducción y detenerla, según

el protocolo especificado en el análisis de la aplicación y basándonos API en Java de esta potente librería.

7.3. Recolector

El recolector de imágenes se basa en un recolector de contenidos web clásico, como por ejemplo el que Google utiliza para descargar el código fuente de todas las páginas web de la red. El algoritmo que hemos utilizado para recolectar dicha información es parecido al ya citado recolector de Google, que se ocupa de rastrear nuevas páginas web para después almacenarlas y permitir su búsqueda mediante etiquetas. En nuestro caso, para cada grupo, lanzamos un hilo que parte de una página web base(por ejemplo, el enlace de la wikipedia correspondiente al grupo) a la que realizamos una petición web, guardándonos su código fuente y los enlaces a otras páginas web. Para cada una de estas páginas web, lanzamos otro hilo que hace lo propio, hasta que o bien hemos alcanzado un nivel de profundidad en el árbol o una cantidad de hilo mayor de las configuradas por el usuario. Para guardar los enlaces hemos utilizado tablas hash y ficheros. El recolector de vídeos se basa en la utilización del API[1] que nos proporciona YouTube. Esta API cuenta con una funcionalidad bastante amplia, permitiendo realizar casi cualquier operación realizable para los vídeos vía web, pero a nosotros simplemente nos interesa la posibilidad de buscar vídeos y vídeos relacionados por medio de palabras clave(el grupo autor de una canción). Así, cada vez que se comparte una canción de un grupo que no estaba en el sistema, se lanza un proceso de recolección de vídeos que, a través de una serie de funciones de esa API, realiza las peticiones al servidor de YouTube y genera el código HTML necesario para mostrar esos vídeos en la página web del sistema.

Por último, tenemos el recolector de letras, para el que hemos optado por el API web de `http://www.chartlyrics.com/`. Esta API funciona por medio de peticiones web. De todas las funcionalidades que proporciona, hemos utilizado la búsqueda por grupo y canción. Para llevar a cabo la petición, hay que construir un enlace que incluya los parámetros de búsqueda. Una vez mandamos la petición, el servidor web nos responde con un código con etiquetas tipo XML que tiene la siguiente estructura:

[1]Una interfaz de programación de aplicaciones o API(del inglés application programming interface) es el conjunto de funciones y procedimientos(o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

```
<GetLyricResult>
    <TrackChecksum>string</TrackChecksum>
    <TrackId>int</TrackId>
    <LyricChecksum>string</LyricChecksum>
    <LyricId>int</LyricId>
    <LyricSong>string</LyricSong>
    <LyricArtist>string</LyricArtist>
    <LyricUrl>string</LyricUrl>
    <LyricCovertArtUrl>string</LyricCovertArtUrl>
    <LyricRank>int</LyricRank>
    <LyricCorrectUrl>string</LyricCorrectUrl>
    <Lyric>string</Lyric>
</GetLyricResult>
```

Figura 7.2: Estructura de la respuesta a una petición de búsqueda de letras

De este modo, nuestro algoritmo se basa en lanzar, para cada canción que esté siendo compartida, un hilo que construya en enlace y extraiga la letra de la canción de la respuesta que recibe, guardando en un fichero todo el texto que va desde $<Lyric>$ hasta $</Lyric>$.

7.4. Procesador

El funcionamiento de los procesadores es bastante sencillo, tanto para las imágenes como para los vídeos. En cuanto a las imágenes, una vez reunidas todas las imágenes de un grupo, se busca por el nombre del grupo en el enlace y se filtran las imágenes de un tamaño inadecuado. Este filtrado es necesario ya que las páginas web contienen infinidad de pequeñas imágenes que actúan a modo de iconos o que están presentes en la página web pero no nos interesan. Así, el procesador de imágenes se encarga de leer secuencialmente cada fichero de enlaces guardando en una tabla hash sólo aquellas que cumplan ambas condiciones. El procesador de vídeos, por su parte, se encarga de separar los vídeos del grupo que estamos buscando y los vídeos relacionados con estos, de modo que tengamos por un lado los vídeos de cada grupo y por otro lado los vídeos de grupos semejantes. El modo de filtrado ha sido desarrollar una serie de métodos que buscan el grupo tanto en el título del vídeo como en su descripción, separando así los vídeos.

7.5. Servidor Web

El servidor web ha sido implementado por medio de servlets, cuya utilidad es generar páginas web de forma dinámica a partir de los parámetros de la petición

que envíe el navegador web. En total, en esta fase del desarrollo, hemos escrito un total de 7 servlets: Imagenes, Letras, Vídeos ProcesaImagenes, ProcesaLetra, ProcesaVideos y Registrar. Los tres primeros se conectan al servidor principal para obtener una lista de las canciones y artistas compatidos actualmente en el sistema. Los servlets de tipo Procesa una vez que el usuario ha elegido el artirsta y/o canción , buscarán dicho artista en el almacén de metadatos, extrayendo la información pertinente.

Capítulo 8

Ampliaciones del segundo prototipo

Este capítulo enumera las mejoras y amplicaciones llevadas a cabo durante el desarrollo del segundo prototipo de la aplicación, aparte de todo el trabajo necesario para portar el modelo de datos a una base de datos relacional :

- Búsqueda de conciertos en fechas próximas para los grupos compartidos en el sistema.
- Búsqueda de las biografías de los grupos.
- Búsqueda de las carátulas de los álbumes y los tags de los mismos.
- Modificación de la adquisición de las imágenes. Después de estudiar otras alternativas como Flickr o Picasa, nos hemos decantado por last.fm, que proporciona una cantidad de resultados más adecuado a lo planeado en un inicio. Además de obtener unos buenos resultados, se obvia la necesidad de filtrar los resultados que obtenemos, ya que todos ellos cumplen las condiciones de filtrado que imponíamos al recolector anterior.
- Posibilidad de que los usuarios escriban reseñas de discos, que podrán ser consultadas en la página web y en las que otros usuarios podrán publicar comentarios escritos.
- Posibilidad de que los usuarios escriban reseñas de conciertos, así como escribir comentarios en estas entradas.
- Implementación de sesiones y Cookies en la página web, de modo que sólo se muestren ciertos contenidos a los usuarios registrados en el sistema y datos rellenados en formularios de unas páginas se guarden mientras dure la sesión y se puedan utilizar para mostrar los contenidos solicitados por los usuarios en las otras pestañas de la página web.

- Amplicación de la información de los usuarios, guardando nuevos datos sobre los usuarios, tal y como se especifica en la semántica de la base de datos, en la sección6.2.2.

- Búsqueda de grupos similares a los grupos que estén siendo compartidos en el sistema, de modo que los usuarios de la aplicación tengan la posibilidad de conocer grupos similares a aquellos que les gusten. Además, con el fin de mejorar la información que el usuario puede obtener de estos grupos similares, se ha preparado el sistema de recolección de información de modo que se busque también toda la información disponible para estos grupos similares, incluidos los grupos similares a esos grupos similares. Así, conseguimos que el usuario pueda conocer mucho más fácilmente a los grupos similares a aquellos que le gustan, consultando la información que le proporcionamos sobre ellos.

- Mejora en la eficiencia del proceso de recolección y procesamiento, tratando de buscar un paralelismo en la ejecución de las diferentes tareas que lo componen, dentro de cada grupo, que se procesapor separado, de manera concurrente. La mejora de rendimiento que así obtenemos es apreciable, si bien supone un mayor esfuerzo para su coordinación.

- Hemos añadido todos los nuevos tipos de información a la página web y subsanado algunos errores que no habían sido advertidos con anterioridad, a base de realizar diversas pruebas con tantos casos particulares como han sido posibles.

Capítulo 9

Conclusiones

Este capítulo contiene las conclusiones a las que hemos llegado una vez finalizado el proyecto, así como las posibles mejoras o líneas futuras en un posible desarollo posterior de la aplicación.

En primer lugar, el diseño e implementación de la aplicación supone el trabajo de mayor envergadura que haya llevado a cabo hasta ahora, lo cual conlleva un gran esfuerzo y aprendizaje en muy diversas áreas. La aplicación que hemos desarrollado es de cierto tamaño, lo cual nos ha permitido adquirir cierta experiencia en el proceso de desarrollo de una aplicación en una mayor medida de lo que habíamos hecho antes, permitiendo además integrar en una misma aplicación distintos conocimientos adquiridos durante la titulación, como la ingeniería del software, las bases de datos, el desarrollo de aplicaciones orientadas a objetos, el diseño e implementación de páginas web, el paradigma cliente-servidor, el desarrollo de aplicaciones distribuidas o el uso de APIs externas de distintos servicios. Por ello, considero que la aplicación que nos ocupa ha resultado lo suficientemente diversa y complicada como para resultar una experiencia enriquecedora y edificante.

Por un lado, el desarrollo de las interfaces gráficas del cliente de la aplicación y de la página web han supuesto una buena introducción, por supuesto mejorable, al desarrollo de interfaces de usuario más cercanas a la realidad de las aplicaciones informáticas profesionales. En general, la idea principal detrás de toda la interfaz gráfica ha sido buscar que ésta sea sencilla y intuitiva, además de lo más limpia posible, buscando no aturdir al usuario de la aplicación con detalles innecesarios que no comprenda o que no pueda configurar. El desarrollo de la página web ha supuesto una excelente introducción al mundo de los servidores web, habida cuenta de que la importancia de las aplicaciones web tiene una importancia mayor cada día.

El seguimiento de las diferentes fases de un proceso de desarrollo ha sido también muy beneficioso, pues nos ha permitido comprobar realmente las ventajas de llevar a cabo un buen análisis y diseño de la aplicación antes de la codificación de la misma, obteniendo sin duda un mejor resultado del que hubiéramos

obtenido de no habrlo hecho. Ahora bien, como detallaremos posteriormente cuando hablemos de las mejoras propuestas para el proyecto, habría resultado interesante completar ese proceso de desarrollo, incluyendo aspectos que no se han tratado demasiado como la realización de un buen plan de pruebas, el seguimiento estricto de la planificación del proyecto o un esfuerzo por optimizar el rendimiento de la aplicación.

En cuanto al desarrollo de la aplicación en sí misma, es interesante destacar el haber profundizado en conceptos como los sistemas distribuidos y el paralelismo de la misma, por encima de los factores ya citados con anterioridad en este mismo capítulo. En concreto, ha resultado complicado y muy instructivo el ser capaces de controlar todos los diversos procesos independientes que forman el sistema, teniendo en cuenta que hemos buscado el paralelismo dentro de éstos procesos en la medida de lo posible, con el fin de obtener un rendimiento aceptable para el sistema.

La ampliación y mejora de la aplicación en el segundo prototipo ha supuesto un gran avance para la aplicación, tanto en el modelo de datos como en la mejora del rendimiento general del sistema y su funcionalidad. El diseño e implementación de una base de datos para el programa supone una mejora notable, como ya expusimos en la sección 6.2, dentro del diseño de la aplicación. Aparte de todas las ventajas ya mencionadas en dicha sección, cabe resaltar que la adaptación del modelo de datos que hemos llevado a cabo nos permitiría, en un futuro, llevar a cabo amplicaciones en la funcionalidad de la aplicación de un modo más sencillo de lo que habría resultado antes de diseñar e implementar dicho modelo. Esto resulta de gran utilidad en un tipo de aplicación como el que nos ocupa, donde las funcionalidades que podamos querer añadir a la misma puede cambiar frecuentemente.

Dicho esto, pasemos a exponer todas aquellas mejoras o líneas futuras de desarrollo para la aplicación. La aplicación, en su estado actual, contiene áreas que , ya sea por motivos de tiempo o por la escasez de experiencia en esas áreas, no ha podido llevarse a cabo con tanta profundidad como me hubiera gustado. Veamos pues por separado todas las áreas en las que se podría mejorar o ampliar lo desarrollado.

Primero, habría resultado interesante profundizar más en la planificación del proyecto, desarrollando una planificación rigurosa que fuera revisado y replanteado frecuentemente, así como estimaciones de costes para el proyecto. Llevar a cabo dichos procesos de manera estricta supondría una condición fundamental para llevar a buen puerto un proyecto profesional.

Además, resultaría muy interesante trabajar en un buen plan de pruebas, para analizar el comportamiento ante los múltiples casos de prueba que se pudieran dar. Si bien hemos tratado de tener en cuenta tantas posibles situaciones particulares como ha sido posible, un buen plan de pruebas permitiría descubrir casos en los que la aplicación no se comporta como debiera, corrigiéndolos y mejorando nuestra aplicación.

Otros aspectos que resultarían de gran interés en el futuro son la optimización del rendimiento de la aplicación y la base de datos. Así, de contar con un espacio de tiempo suficiente, podríamos llevar a cabo un estudio detallado de diferentes opciones que nos permitieran mejorar el rendimiento de nuestra aplicación, ya sea optimizando alguna de las secciones ya desarrolladas o considerando construir un cluster de servidores o un servidor proxy caché web. El objetivo sería mejorar en la medida de lo posible el rendimiento del sistema, en el caso de que éste tuviera que trabajar con un gran volumen de usuarios concurrentes, peticiones e información. Así pues, el utilizar un cluster de servidores incrementaría el rendimiento del sistema y nos exigiría conocimientos de arquitecturas de computadores y paralelismo que ahora mismo no poseemos. Por otro lado, el servidor proxy caché web permitiría que, ya que frecuentemente la información a la que desean accder los usuarios es similar, al almacenarla en la caché, la información sobre los grupos compartidos en el sistema se fuera guardando, de modo que sólo habría que procesar aquellos que no hayan sido compartidos nunca. En cuanto a la base de datos, sería interesante, una vez adquiridos nuevos conocimientos, optimizar el modelo y las consultas en torno al mismo, con el objetivo de que el sistema fuera capaz de dar servicio a muchos usuarios de manera concurrente, lo que implicaría estudiar estrategias que permitieran distribuir la carga de las peticiones a la base de datos. Todo esto supondría un gran esfuerzo y un área de mejora interesante para la aplicación, además de una ampliación de conocimientos bastante importante y de gran utilidad.

La seguridad de la aplicación es otro área de gran interés. El caso hipotético de una ampliación y puesta en producción de nuestra aplicación implicaría una seria posibilidad de ataques,intrusiones o denegaciones de servicio a nuestro sistema. Para evitarlo, sería necesario desarrollar todo un sistema de seguridad y un sistema de control de errores para los posibles bugs del sistema, en el caso de que, por ejemplo, nuestro programa permitiera la compra de canciones de manera similar a iTunes o en general cualquier servicio que implicara un pago a través de la red. Para ello, se hacen necesarios, además de los conocimientos de seguridad en redes informáticas ya adquiridos, los principios de criptografía y teoría de la información que se imparten en el ciclo superior de la titulación.

Por último , se podría mejorar el rendimiento del servidor web, ya sea profundizando en las opciones que nos brinda Tomcat o considerando la migración a un servidor web más eficiente, que nos de un mejor rendimiento para servir a los potenciales clientes de la aplicación.

Capítulo 10

Bibliografía

Este capítulo contiene la bibliografía y los recursos web que hemos utilizado durante el desarrollo del proyecto.

Bibliografía

[1] Ben Wang : *INTEGRATED PRODUCT, PROCESS AND ENTERPRISE DESIGN*. Ed Chapman and Hall

[2] Abraham Silberschatz : *FUNDAMENTOS DE BASES DE DATOS* . Ed McGraw-Hill

[3] Bruce Eckel : *PIENSA EN JAVA*

[4] Kevin Muckar, Todd Lauinger y John Carnell : *FUNDAMENTOS DE BASES DE DATOS CON JAVA: JDBC, SQL, J2EE, EJB, JSP y XML* . Ed Anayamultimedia

[5] Jason Brittain y Ian F. Darwin: *TOMCAT 6.0: LA GUIA DEFINITIVA* . Ed Anaya multimedia.

http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Java
http://www.revista.unam.mx/vol.1/num2/art4/index.html
http://es.wikipedia.org/wiki/MySQL
http://dev.mysql.com/doc/refman/5.0/es/index.html
http://es.wikipedia.org/wiki/Tomcat
http://tomcat.apache.org/tomcat-6.0-doc/
http://www.mysql.com/products/workbench/
http://download.oracle.com/javase/tutorial/rmi/overview.html
http://www.chartlyrics.com/api.aspx
http:
//code.google.com/intl/es/apis/youtube/developers_guide_java.html
http://code.google.com/p/gstreamer-java/
http://www.ibm.com/developerworks/aix/library/au-gstreamer.html?
ca=dgr-lnxw07GStreamer
http://jwbf.sourceforge.net/doc/
http://www.u-mass.de/lastfm/doc/
http://www.lastfm.es/api/intro
http://blog.programmableweb.com/2008/02/21/25-music-apis/

Índice de figuras