AN AUTOMATIC SONG ANNOTATION SYSTEM

BY

IRENE ZELLER SANCHO

Advisor: Isaac Gelado Fernández

Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Telecomunication Engineering
in the Universitat Politècnica de Catalunya, 2009

Abstract

The amount of multimedia content in the audiovisual sector, as well as on the Internet, is increasing a lot, and Music is one of the most outstanding forms of multimedia content requested by users. Every year, new songs, artists and genres appear in the market. Managing this musical content is, thus, becoming a very complex task. The present document presents the design and implementation of a system, that aims to solve the problem related to multimedia content management.

A mi familia.

Acknowledgments

Este proyecto no hubiera sido posible sin la ayuda de varias personas. Quiero agradecer todo el soporte técnico y ayuda a mi tutor Isaac Gelado, quién, amablemente, me ha guiado y respaldado en todo momento, y quién me ha dado luz en lo momentos difíciles. También quiero dar las gracias a Xavier Vives por la confianza total que ha depositado en mi dándome la oportunidad de realizar este proyecto.
Muchas gracias también a mi familia por haberme apoyado todos estos años, sin este apoyo nada de esto hubiera sido posible. Y especialmente a Javi por aguantarme estos últimos meses.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

CENIT	Centro para el desarrollo tecnológico industrial.
CCMA	Corporació Catalana de Mitjans Audiovisuals.
ISMIR	The International Society for Music Information Retrieval.
FAM	File Alteration Monitor.
CPU	Central Processing Unit.
CORBA	Common Object Requesting Broker Architecture.
OMG	Object Management Group.
ORB	Object Request Brokers.
IIOP	Inter-ORB Protocol.
IDL	Interface Definition Language.
SMB	Server Message Block.
CIFS	Common Internet File System.
BSD	Berkeley Software Distribution.
DBMS	Database Management System.
OR-DBMS	Object-relational database management system.
ODBC	Open Database Connectivity.
IODBC	Independent Open DataBase Connectivity.
JDBC	Java Database Connectivity.
MVC	Model View Controller.
OGNL	Object-Graph Navigation Language.
JSP	Java Server Pages.
AJAX	Asynchronous JavaScript and XML.
JSON	JavaScript Object Notation.
EJB	Enterprise Java Beans.
JVM	Java Virtual Machine.

POSIX	Portable Operating System Interface.
SDM	Systems Development Method.
IT	Information Technology.
MPEG	Moving Picture Experts Group.
VFS	Virtual File System.
TCP	Transmission-Control-Protocol.
HTTP	HyperText Transfer Protocol.
IP	Internet Protocol.
SSL	Secure Sockets Layer.
ERM	Entity-Relationship Model.
WAR	Web Application Archive.
XML	Extensible Markup Language.
HTML	HyperText Markup Language
JNDI	Java Naming and Directory Interface.
SDK	Software Development Kit.
IDE	Integrated Development Environment.
URL	Uniform Resource Locator.
URI	Uniform Resource Identifier.
OOP	Object-Oriented Programming.
UML	Unified Modeling Language.
W3C	World Wide Web Consortium.
JAR	Java Archive.
WAR	Web Application Archive.

Chapter 1

Introduction

1.1 Motivation

This thesis emerged from the i3media project. The Spanish government funded the i3media project with the objective of developing new tools for the whole value chain of the audiovisual sector. I3media is a CENIT (*Centro para el desarrollo tecnológico industrial*) project and it is focused on promoting cooperation between the public and the private sector in R&D&I.

The i3media project covers several fields of development, such as: audio, video and text processing; retrieval and data-mining models for multimedia objects; modular search tools to sit on a wide range of media management platforms, query interfaces that integrate secure access and rights management, etc

The project contains ten work packages. Specifically, this project is a part of the work package five, whose target is to develop annotation and summarization systems for audio files. It covers two fields: music audio processing and spoken audio processing. Two project partners are involved in the first research line: Noufer and CCMA (*Corporació Catalana de Mitjans Audiovisuals*).

Noufer is a small company whose objective in the project is to implement systems related to music audio processing. CCMA is a group of companies of the audiovisual sector of Catalonia. The public television of Catalonia (TVC) and the public Radio of Catalonia (Catalunya Ràdio) form part of this group, and both are interested in improving the management of their audiovisual contents (mainly audio, video and music). Both companies CCMA, on behalf of Catalunya Ràdio, and Noufer found common synergies that the i3media project aims to achieve.

Music programmers assign parameters to the songs, such as tempo, energy or others, to facilitate the music file retrieval process. This action is called song annotation and it is currently done by hand by Catalunya Ràdio employees. It is known that annotating songs manually has a huge cost, but automatic annotating tends to be quite inaccurate and, in some cases, subjective.

Noufer has developed a semi-automatic system to annotate songs, formed by music processing libraries, to help music programmers. This system needs feedback from users to improve its results. This strategy will increase efficiency compared to manual annotation, and increase accuracy compared to automatic annotation.

CCMA delivers a dataset that is composed of a collection of annotated songs. This company also defines the user requirements and integrates the music processing libraries in a final multimedia content management system.

This report presents the integration process to use this library in the production scope. The design is based on Catalunya Ràdio users requirements. The approach has been implemented to be as efficient as possible and to let users interact with the music audio processing results in a straightforward manner.

1.2 Objectives

The main goal of this thesis is to develop a sturdy and stable music management system. This system will allow the usage of the audio libraries for songs processing. The system has been designed to easily work in any company of the media sector in a production environment. To achieve this goal, the following objectives are outlined:

1. To analyse the user requirements of the media environment, specifically requirements of Music programmers of Catalunya Ràdio.
2. A system arquitecture that fullfills the user requirements. This objective can be decomposed into:
    - The design and implementation of a processing server that uses the processing libraries and serves requests to process new songs added to the system.
    - The design and implementation of a detailed database model. The database will store the data generated by the processing system and the validations of this data done by users.
    - The design and implementation of a web application that presents the results of the audio processing. This web application will let users validate these results and listen to the songs that users are validating.
3. To adjust the design to the operating environment requirements. The operating environment used by the music programmers at Catalunya Ràdio is Microsoft Windows and all songs are stored in Windows Servers. The processing libraries are developed for GNU/Linux operating system. Consequently, the integration approach design must deal with this environment.

Chapter 2

Background

Several frameworks, libraries and other technologies have been used to assist the implementation of this project. Here, we describe the main technologies that are used through the application. This chapter aims to ease the understanding of next chapters, specifically the implementation chapter.
These technologies are itemized in:

1. **Audio annotation:** This section shows the state-of-the-art of the audio annotation.

2. **Audio processing:** It explains the libraries used in this project to perform audio processing tasks.

3. **File monitoring:** It shows different libraries that provide event notification on filesystems modifications.

4. **Application distribution technologies:** It details the libraries used to communicate the application through the computer network.

5. **Database:** It presents a comparison amongst different databases to select the best suited for this project. This section also explains two API's used to connect the system to the database.

6. **Web application:** It explains the frameworks, tools and libraries used to implement the Web Application.

2.1 Audio annotation

Many literature about automatic annotation systems is currently available in digital and traditional engineering journals, such as ISMIR [16] or online digital libraries such as IEEE [11]. Many developers are implementing systems for automatic annotation of songs, and almost all of them find a clear common point: achieving a completely automatic annotation system is not possible. Each person feels Music in a different way and, consequently, the way each person describes a song with words could be absolutely different. Hence, developers tend to develop semi-automatic music annotation systems, where users validate the system proposals.

A consortium called ISMIR (The International Society for Music Information Retrieval) was developed in 2000. This was the first forum for people studying music retrieval, and allowed exchanging experiences and problems concerning music retrieval that helped to move towards viable solutions. Every year, the consortium organizes a conference in a different city. In this conference different topics and problems about music retrieval and annotation are discussed and it is the main source of information about music annotation.

This consortium is a clear example that the music annotation field is increasing every year and also that the state-of-the-art is still far away from being able to provide perfect systems of annotation and retrieval of songs. Furthermore, the complexity of finding a common criterion for describing songs is enhanced in the different development groups. The main problem in this kind of systems is finding a flexible way to describe songs and letting users obtain a customized system. Two listeners may use drastically different words when describing the same piece of music. However, words related to some aspects of the audio content, such as instrumentation and genre, may be largely agreed upon by a majority of listeners. This agreement suggests that it is possible to create a computer system that can learn the relationship between audio content and words.

Music annotation systems are being developed by different consortiums [39] [46] and they tend to differ in two types of parameters. The first type consists of physical parameters that are extracted from songs such as energy, spectral density, spectral flatness, etc. These parameters are physical qualities of songs that are completely objective. The second type of parameters are subjective, such as the genre, the tempo, the ending (how the song ends) or energy, but using a subjective value, such as, more energetic, less energetic, etc. This second class of parameters tends to be more valuable for users than the first, the objective class. Subjective parameters can be extracted using a training system with a data set of annotated songs. Using this training set is possible to relate the physical qualities to the descriptive qualities. Hence, descriptive parameters will be extracted automatically from new songs added to the system without previous annotation.

A more complex approach to improve results and to allow the system to be customized is using artificial intelligence techniques [43]. Artificial intelligence requires a training stage where parameters previously extracted are validated by users. A very popular artificial intelligence approach are neural networks. It consist of a mathematical model or computational model that tries to simulate the structure and/or functional aspects of biological neural networks. In most cases artificial neural networks are an adaptive systems that changes their structure based on external or internal information that flows through the network during the learning phase. Neural networks can be considered as non-linear statistical data modeling tools. They can

be used to model complex relationships between inputs and outputs or to find patterns in data.
This new artificial intelligence approach allows extracting subjective parameters with improved accuracy. Eventually this system will be completely customized for each different user, as it will learn the different criterion of each listener for describing songs with words. This approach supposes that the user is always the same for a concrete application, or at least different users but with the same criteria about how to describe a song. If different users use the system, the improved results will make no sense, since the system will be learning from different users criteria. However, a different instantiation of the Neural Network can be done for each user in the system.
Some prototypes have been developed, such as the prototype of the UPF [38] (Universitat Pompeu Fabra), which tries to find the mood parameter. However, there are not yet commercial or free software products that provide the feature of annotation music with descriptive words. Therefore, this is an excellent challenge to test a module that provides this option and, thanks to the collaboration between the companies (the development company and the industrial company), implement a customized tool for music programmers.

2.2 Audio processing

2.2.1 FFmpeg: libavcodec and libavformat

Libavcodec and libavformat are two libraries that form part of the FFmpeg [9] project. FFmpeg is a computer program that can record, convert and stream digital audio and video in numerous formats. FFmpeg is a command line tool that is composed of a collection of free software/open source libraries. It includes libavcodec, an audio/video codec library used by several other projects, and libavformat, an audio/video container mux and demux library.

2.2.2 Audiotool

The AudioTool library is the project core. It has been developed by Noufer. It processes the songs and extracts objectives parameters of them (i.e. energy, spectral flatness, etc.). In the future, these libraries will be evolved and improved and the extracted parameters types will be more advanced. It will also implement artificial intelligence to allow training the system with validation results, performed by the users, and improve the audiotool results. The physical parameters that the system currently extracts are:

- Energy: an estimation of the energy evolution. The library analyzes the energy by blocks of a number of seconds. The function returns an array with the energy values by blocks of a number of seconds in a normalized logarithmic value.

- Energy Rate: an estimation of the maximum difference of energy in logarithmic value.

- Ending: an analysis of how a song ends and whether the song ends like a fade or like a cut. A fade represents a song that finishes softy and a cut corresponds to a song that ends abruptly. The algorithm has been developed calculating the ratio of energy decreasing in the last seconds of the song. The library assumes the song energy is more or less stable until the final block. Consequently, it is possible to approximate the energy of the song by two straight lines (one horizontal and the other one with a slope). The slope of the second straight line shows if the song is a fade or a cut. Other way to calculate this parameter is by using two straight lines with any slope (not one horizontal like in the first algorithm). It is possible to figure out if it is a fade or a cut calculating the angle between the two lines. The first approach offers a higher number of correct answers, so it is the approach used by the library.

- Tempo: tempo estimation is extremely complex (unless the song contains a marked rhythm). The best results currently obtained by the development groups are far from being optimal. The calculation is complex because of the complexity of defining the tempo in question. For instance, an audio signal composed of a beat of drum every second would have 60 beats per minute. However, if the same signal has a beat of cymbal between beats of drum, beats per minute would be 120. Consequently, obtained results are not good as desired. After some assessment, the best approach developed has been an onset detector with a wave rectification.

- Modulation 4Hz: this parameter detects the signal voice in a song. The voice modulation in a song is around 4 Hz (this value has been calculated empirically). The wave is rectified in order to calculate the modulation. Then, the power of the output signal of a 4 Hz filter is calculated.

- Spectral Centroid: a measure that characterizes the spectrum. This value indicates where the "center of mass" of the spectrum is. Perceptually, it has a robust connection with the impression of "brightness" of a sound.

- Spectral skewness: a measure of the spectral symmetry.

- Spectral kurtosis: an estimation of the presence of transients series and their locations in the frequency domain. This metric helpfully supplements the well-know classical power spectral density and completely eradicates non-stationary information.

- Spectral flatness: a measure for audio spectrum. A high spectral flatness indicates that the spectrum has a similar amount of power in all spectral bands - this would sound similar to white noise, and the graph of the spectrum would appear relatively flat and smooth. A low spectral flatness indicates that the spectral power is concentrated in a relatively small number of bands - this would typically sound like a mixture of sine waves, and the spectrum would appear "spiky".
- Zero-crossing rate: the rate of sign-changes along a signal, (i.e., the rate at which the signal changes from positive to negative or back). This feature has been used heavily in both speech recognition and music information retrieval.

Other parameters with more advanced information such as the mood or the role (man, woman or duet) will be extracted in the next library release. The artificial intelligence feature will also be included.

2.3 File monitoring

2.3.1 FAM (File Alteration Monitor)

FAM [8] (File Alteration Monitor) provides a subsystem developed for Unix-like operating systems. The FAM subsystem allows applications to watch certain files and be notified when they are modified. This greatly aids some applications, because before FAM existed, such applications had to read the disk repeatedly to detect changes, which resulted in high disk and CPU usage. For example, a file manager application can detect if some file has changed and can then update a displayed icon and/or filename. Although FAM may seem unnecessary now that many newer kernels include built-in notification support (details to follow), using FAM provides benefits: FAM enables applications to work on a greater variety of platforms.
The main problem of FAM is that during the creation of a large amount of files (for example during the first login in a desktop environment) it slows down the entire system, using many CPU cycles.

2.3.2 Dnotify

Dnotify [6] is a file system event monitor for the Linux kernel. It has been obsoleted by Inotify, but will be retained for compatibility reasons. Dnotify has many disadvantages over Inotify, the new module. A given program has to use one file descriptor for each directory that it was monitoring. This can become a bottleneck due to the limit of file descriptors per process could be reached. The use of file descriptors along with Dnotify also proved to be a problem when using removable media. Devices could not be unmounted since file descriptors kept the resource busy. Another drawback of Dnotify is the level of granularity, since programmers can only monitor changes at the directory level.

2.3.3 Inotify

Inotify [13] is a Linux kernel subsystem that provides file system event notification and aims to substitute FAM. Its major functionality allows reindexing of changed files without scanning the file system for changes every few minutes, which would be very inefficient. By being told that a file has changed directly by the kernel, rather than actively polling, it can achieve change-to-reindexing times of only about a second. It can also be used to automatically update directory views, reload configuration files, log changes, backup, synchronize, and upload. We use Inotify to identify new songs added to the system.

2.4 Application distribution technologies

2.4.1 CORBA

The Common Object Requesting Broker Architecture (CORBA) [14] [42] is a middleware defined by the Object Management Group (OMG) [26] that enables software components written in multiple computer languages and running on multiple computers to work together. CORBA provides a framework for the development and execution of distributed applications. The distributed applications introduces a whole new set of difficult issues. However, sometimes there is no choice; some applications by their very nature may be distributed across multiple computers. For instance, the data used by the application are distributed, the computation is distributed, application users are distributed or the data are distributed. In this case, application users are distributed and they interact with the engine server via the application, and they share objects that are executed in the server. CORBA essentially is an object bus, enabling a client to invoke methods on remote objects contained in a server, independently of the programming language the objects have been written in and of their location. The client-server interaction is mediated by Object Request Brokers (ORBs), present both on client and on server sides and communicating usually via the Internet Inter-ORB Protocol (IIOP).

CORBA objects can be either collocated with the client or distributed on a remote server, without affecting their implementation or use: the ORBs will take care of the details. The methods of CORBA objects are defined using the Interface Definition Language (IDL), they accept as input parameters and return as values CORBA data-types and can raise exceptions.

CORBA is designed to be OS-independent. CORBA is available in Java (OS-independent), as well as natively for Linux/Unix, Windows, Sun, Mac and others.

Figure 2.1 shows the CORBA reference model, which collaborate to provide the properties briefly outlined before.

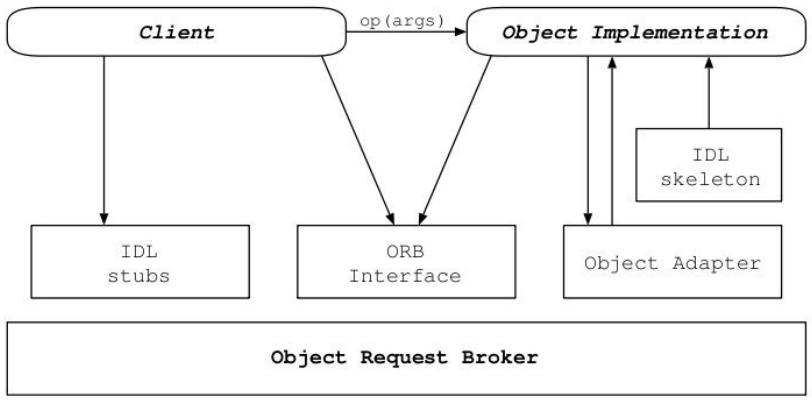


Figure 2.1: Components of the CORBA reference model

To sum up, the main concepts that form part of the middleware CORBA and are used in this project are:

- IDL: CORBA uses an interface definition language (IDL) to specify the interfaces that objects will present to the outside world. CORBA then specifies a mapping from IDL to a specific implementation language such as C++ or Java.

- ORB: An object request broker (ORB) is a piece of software that allows programmers to make program calls from one computer to another via a network. ORBs promote interoperability of distributed object systems because they enable users to build systems by piecing together objects- from different vendors- that communicate with each other via the ORB. Some ORBs, such as CORBA systems, use an Interface Description Language (IDL) to describe the data which is to be transmitted on remote calls.

- IIOP: In distributed computing, General Inter-ORB Protocol (GIOP) is the abstract protocol by which object request brokers (ORBs) communicate. IIOP (Internet Inter-Orb Protocol) is the implementation of GIOP for TCP/IP.

In object-oriented languages, the ORB takes the form of an object with methods enabling connection to the objects being served. After an object connects to the ORB, the methods of that object become accessible

for remote invocations. The ORB requires some means of obtaining the network address of the object that has now become remote. The typical ORB also implements many other methods, but they are out of the scope of this thesis.
CORBA has been used to communicate the server engine, written in C++, with the Java Web Application. The implementation, used in this project, for developing the C++ CORBA server is OmniORB4. Java SDK provides an IDL compiler called `idl2java` and it is not necessary other implementation, like in C++.

OmniORB4

OmniORB4 [27] is a high performance CORBA ORB for C++ and Python. It is freely available under the terms of the GNU Lesser General Public License (for the libraries), and GNU General Public License (for the tools). It provides an IDL compiler to generate the C++ code that provides the object mapping as defined in the CORBA specification.

2.4.2 SAMBA

SAMBA [31] is a free software re-implementation of SMB/CIFS networking protocol. Server Message Block (SMB) operates as an application-layer network protocol mainly used to provide shared access to files, printers, serial ports, and miscellaneous communications between nodes on a network. SMB was renamed to Common Internet File System (CIFS) and added more features, such as including support for symbolic links, hard links, larger file sizes etc. Samba provides file and print services for various Microsoft Windows clients and can integrate with a Windows Server domain, either as a Primary Domain Controller (PDC) or as a domain member. It can also be part of an Active Directory domain. Samba runs on most Unix and Unix-like systems, such as GNU/Linux, Solaris, AIX and the BSD variants. SAMBA is standard on nearly all distributions of GNU/Linux and is commonly included as a basic system service on other Unix-based operating systems as well. SAMBA is released under the GNU General Public License. The name SAMBA comes from SMB (Server Message Block), the name of the standard protocol used by the Microsoft Windows network file system.

2.5 Database

2.5.1 Database comparison

This project requires a Database Management System (DBMS) to create and manage a database that stores the project data. A DBMS consists of a set of computer programs that controls the creation, maintenance, and the use of the database.

A comparison about general and technical information of some DBMS has been performed to decide the better approach for this project.

In concrete, four databases have been chosen: MySQL [25] , PostgreSQL [30] , Informix [12] and Oracle [29]. These are representative database, used by several companies and projects. For instance, MySQL is the database used in projects such as Facebook, and Wikipedia; PostgreSQL is used by Skype, Yahoo or MySpace; Oracle by Amazon and SAP and Informix by CCMA (Corporació Catalana de Mitjans Audiovisuals).

These four databases are available for different operating systems, such as Windows, GNU/Linux and Unix and provide basic features such as:

- Transactions: a unit of work performed within a database management system (or similar system) against a database, and treated in a coherent and reliable way independent of other transactions. In database environment has two main purposes: providing reliable units of work that allow correct recovery from failures and providing isolation between programs accessing a database concurrently
- ACID: it means atomicity, consistency, isolation, durability. It is a set of properties that guarantee that database transactions are processed reliably.
- Referential integrity: in relation database it refers to any field in a table that is declared a foreign key. It can contain only values from a parent table's primary key or a candidate key. For instance, deleting a record that contains a value referred to by a foreign key in another table would break referential integrity.
- Unicode: is a computing industry standard that allows computers to represent and manipulate text expressed in most of the world's writing systems consistently.
- Graphical user interface: it allows people to interact with the database in a more intuitive way.

However, MySQL does not always provide neither referential nor integrity nor transactions as default. InnoDB [2] (a storage engine for MySQL) should be used to provide these features.

In all these database the maximum database size is unlimited or is really huge (such as Informix).

They also support natively temporary tables. However, materializes views are only supported by Oracle. A materialized view takes a different approach to temporary views in which the query result is cached as a concrete table that may be updated from the original base tables from time to time. Both, MySQL and PostgreSQL, can emulated it using different techniques. Informix does not support this feature.

All these databases provide basic indexes (such as B-/B+ tree). All of them, except MySQL, provide more advanced indexes (such as Hash, Expression, Partial, etc.)

They also support different database capabilities, such as inner, outer and merge joins, blobs and clobs, trigger, cursors, function and procedure (refer to internal routines written in SQL and/or procedural language like PL/SQ) and external routine (refers to the one written in the host languages, such as C or Java).

Two of these databases, Oracle and PostgreSQL, are object-relational database management system (ORDBMS). This kind of DBMS consists of a database management system with an object-oriented database model: objects, classes and inheritance are directly supported in database schemas and in the query language. In addition, it supports extension of the data model with custom data-types and methods.

Regarding the license of these databases, Oracle and Informix are propietary software, MySQL can be GNU General Public License (version 2) or proprietary, and PostgreSQL is BSD license.

The only DBMS not proprietary in any case is PostgreSQL. This database provides basic and advanced features, comparable, in some cases, to Oracle. Moreover, only PostgreSQL provides variable length arrays (including text and composite types) up to 1GB in total storage size, or at least, it is the only DBMS that documents this feature. This feature is necessary in this project. Thus, the selected DBMS is PostgreSQL.

2.5.2 ODBC

ODBC [24] is a standard software API, whose acronym means Open Database Connectivity, and provides a method for using database management systems (DBMS). The designers of ODBC aimed to make it independent of programming languages, database systems, and operating systems. ODBC implementations run on many operating systems, including Microsoft Windows, Unix, Linux, OS/2, OS/400, IBM i5/OS, and Mac OS X. Hundreds of ODBC drivers exist, including drivers for Oracle, DB2, Microsoft SQL Server, IBM Lotus Domino, MySQL, PostgreSQL and desktop database products such as FileMaker, and Microsoft Access. In this project, it was necessary a driver for the PostgreSQL database and for operating system Linux. The implementation, used in this project, is IODBC [15] (Independent Open DataBase Connectivity), that offers an open source implementation.

2.5.3 JDBC

JDBC [18] is an API for the Java programming language that defines how a client may access to a database. It provides methods for querying and updating data in a database. JDBC is oriented towards relational databases. It allows multiple implementations to exist and be used by the same application. The API provides a mechanism for dynamically loading the correct Java packages and registering them with the

JDBC Driver Manager. The Driver Manager is used as a connection factory for creating JDBC connections. JDBC connections support creating and executing statements. These may be update statements such as SQL's CREATE, INSERT, UPDATE and DELETE, or they may be query statements such as SELECT.

2.6 Web application

The web application is implemented in Java, specifically using JavaEE [19]. JavaEE is a part of the Java project that provides functionalities to deploy fault-tolerant, distributed, multi-tier Java software, based mainly on modular components running on an application server. We use the Struts2 framework that provides some Ajax plug-ins. The application server is Jboss. The business logic of the application, that manages mainly the database access, is encapsulated by Enterprise JavaBeans 3.0 server-side. Here, the main components of the Application Web are detailed:

2.6.1 Struts2

Struts2 [32] [41] is an extensible framework for creating enterprise-ready Java web applications. The framework is designed to streamline the full development cycle, from building, to deploying, to maintaining applications over time. It implements the Model-View-Controller (MVC) design pattern. MVC is an architectural pattern used in software engineering. This pattern isolates business logic from input and presentation, permitting independent development, testing and maintenance of each.

From a high level, Struts2 is a pull-MVC (or MVC2) framework; this is slightly different from a traditional MVC framework in that the action takes the role of the model rather than the controller, although there is some overlap. The pull comes from the views ability to pull data from an action, rather than having a separate model object available. The Model-View-Controller pattern in Struts2 is composed of five core components: actions, interceptors, value stack/OGNL [3], result types and results/view technologies. These component are:

- **Action:** The action mappings are the basic "unit-of-work" in the framework. Essentially, the action maps an identifier to a handler class. When a request matches the action's name, the framework uses the mapping to determine how to process the request. An Action is an adapter between the contents of an incoming HTTP request and the corresponding business logic that should be executed to process this request. The controller (RequestProcessor) will select an appropriate Action for each request, create an instance (if necessary), and call the **execute** method.

- **Interceptors:** Interceptors allow to define code to be executed before and/or after the execution of an Action method. There are many use cases for Interceptors, including validation, property population, security, logging, and profiling.
- **Value stack/OGNL:** Object-Graph Navigation Language is an open-source Expression Language for Java, which, while using simpler expressions than the full range of those supported by the Java language, allows getting and setting properties (through defined setProperty and getProperty methods, found in JavaBeans), and execution of methods of Java classes.
- **Result type:** Struts2 supports many result types. They can be visual or interactions with the environment. To configure an action to execute a result of a specific type, the type attribute is used. If the attribute is not supplied, the default type dispatcher is used (this will render a JSP result).
- **Results/View:** The most common result is Java Server Pages (JSPs). Although it is not the only way to render results. For instance, it can be JSON result or Stream result.

Figure 2.2 shows the model, view and controller to the Struts2 high level architecture. The controller is implemented with a Struts2 dispatch servlet filter as well as interceptors, the model is implemented with actions, and the view as a combination of result types and results. The value stack / OGNL provide common thread, linking and enabling integration between the other components.

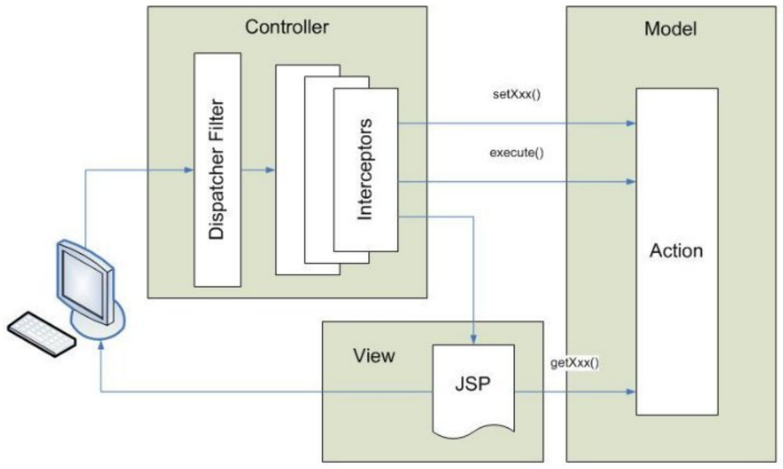


Figure 2.2: The MVC / Struts2 Architecture

The web.xml web application descriptor file represents the core of the Java web application, thus, it is also part of the Struts2 framework core. In the web.xml file, Struts defines its FilterDispatcher, the Servlet Filter class that initializes the Struts framework and handles all requests. This is for configuring a basic web application. In order to customize the web application execution environment two files are used. The **struts.properties** configuration file provides a mechanism to change the default behavior of the framework, and the **struts.xml** configuration file to configure the components for the web application.
Figure 2.3 shows how the configuration of a Struts2 application can be broken into three separate files.

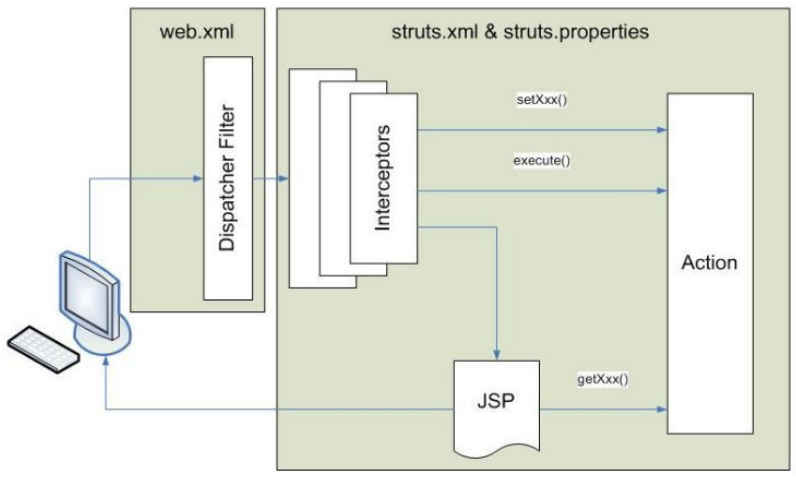


Figure 2.3: Configuration file scope for framework elements

2.6.2 AJAX

AJAX (Asynchronous JavaScript and XML), is a group of interrelated web development techniques used on the client-side to create interactive web applications or rich Internet applications. Using AJAX, web applications can retrieve data from the server asynchronously in the background without interfering with the display and behaviour of the existing page. Data is usually retrieved using the XMLHttpRequest object. Struts2 provides AJAX plug-ins. This project uses two different plugins: Dojo and JSON.

Dojo

Dojo Toolkit [7] is an open source modular JavaScript library (or more specifically JavaScript toolkit) designed to ease the rapid development of cross platform, JavaScript/Ajax based applications and web sites. One important feature of Ajax applications is asynchronous communication of the browser with the server: information is exchanged and the page's presentation is updated without a need for reloading the whole page. Traditionally, this is done with the JavaScript object XMLHttpRequest. Dojo provides an abstracted wrapper (dojo.io.bind) around various web browsers' implementations of XMLHttpRequest, which can also use other transports (such as hidden IFrames) and a variety of data formats. Using this approach, it is easy to have the data a user enters into a form sent to the server ”behind the scenes”; the server can then reply with some JavaScript code that updates the presentation of the page.

JSON

JSON [23] , JavaScript Object Notation, is a lightweight computer data interchange format. It is a text-based, human-readable format for representing simple data structures and associative arrays (called objects). The JSON format is often used for serialization and transmitting structured data over a network connection. Its main application is in Ajax web application programming, where it serves as an alternative to the XML format.

2.6.3 EJB3.0

Enterprise JavaBeans (EJB) [21] is a managed, server-side component architecture for modular construction of enterprise applications. It encapsulates the business logic of an application. The EJB specification provides a standard way to implement the back-end 'business' code typically found in enterprise applications (as opposed to 'front-end' interface code). Such code was frequently found to address the same types of problems, and it was found that solutions to these problems are often repeatedly re-implemented by programmers. Enterprise JavaBeans were intended to handle such common concerns as persistence, transactional integrity, and security in a standard way, leaving programmers free to concentrate on the particular problem at hand.

2.6.4 Google guice

The integration between EJB3.0 and Struts2 has been performed with Google Guice [10] . It is an open source software framework for the Java platform released by Google under an Apache license. It provides support for dependency injection using annotations to configure Java objects. Guice allows implementation classes to be programmatically bound to an interface, then injected into constructors, methods or fields using an @Inject annotation. When more than one implementation of the same interface is needed, the user can create custom annotations that identify an implementation, then use that annotation when injecting it.

2.6.5 Applet

An applet is any small application that performs one specific task; sometimes running within the context a larger program perhaps as a plugin. However, the term typically also refers to programs written in the Java programming language which are included in an HTML page.
A Java applet [17] is an applet delivered in the form of Java bytecode. Java applets can run in a Web browser using a Java Virtual Machine (JVM), or in Sun's AppletViewer, a stand-alone tool for testing applets. Applets are used to provide interactive features to web applications that cannot be provided by HTML alone, such as music players or calculators.

2.6.6 JavaZoom

Javazoom [20] is a project that provides a set of libraries, applets (half free, half shareware), servlets, free web services and open source projects. Applets available are navigation system, banner rotator, scrollers, text effects. Servlets are upload and download components, chat solution and mail composer JSP front end. Open source projects are Java MP3 Decoder, WinAmp clone and Ogg Vorbis Javasound SPI. Specifically, in this project the jlGui libraries (a Music Player application for the Java Platform) is used. It provides the functionalities of the an advanced player.

2.6.7 Jboss Server

An application Server, referred to software framework, has been used to host the developed web application. The selected application Server is Jboss [22]. It is a free software/open-source Java EE-based application server. It is Java-based and is usable on any operating system that Java supports, including many POSIX platforms (like Linux, FreeBSD and Mac OS X) and Microsoft Windows. The version is JBoss AS 5.1, the current version of 2009. It operates as a Java EE 6 application server, deploys Enterprise JavaBeans 3.0 by default and supports JDBC, JSP and Servlets. These are essential features for the developed web application.

Chapter 3

Methodology, planning and budget

3.1 Methodology

The software development process followed in this project is the spiral model [36]. The spiral model consists of combining elements of both design and prototyping-in-stages, in an effort to combine advantages of top-down and bottom-up concepts.

In the software development process, the top-down and bottom-up approaches play a key role.

Top-down approaches emphasize planning and a complete understanding of the system. It is inherent that no coding can begin until a sufficient level of detail has been reached in the design of at least some part of the system. The Top-Down Approach is done by attaching the stubs in place of the module. This, however, delays testing of the ultimate functional units of a system until significant design is complete.

Bottom-up emphasizes coding and early testing, which can begin as soon as the first module has been specified. This approach, however, runs the risk that modules may be coded without having a clear idea of how they link to other parts of the system, and that such linking may not be as easy as first thought. Re-usability of code is one of the main benefits of the bottom-up approach.

Modern software design approaches usually combine both top-down and bottom-up approaches. Although an understanding of the complete system is usually considered necessary for good design, leading theoretically to a top-down approach, most software projects attempt to make use of existing code to some degree. Pre-existing modules give designs a bottom-up flavour. Some design approaches also use an approach where a partially-functional system is designed and coded to completion, and this system is then expanded to fulfill all the requirements for the project.

Top-down is also a programming style, the mainstay of traditional procedural languages, in which design begins by specifying complex pieces and then dividing them into successively smaller pieces. Eventually, the components are specific enough to be coded and the program is written. This is the exact opposite of the bottom-up programming approach which is common in object-oriented languages such as C++ or Java.

In a bottom-up approach the individual base elements of the system are first specified in great detail. These

elements are then linked together to form larger subsystems, which then in turn are linked, sometimes in many levels, until a complete top-level system is formed. This strategy often resembles a ”seed” model, whereby the beginnings are small, but eventually grow in complexity and completeness.
The spiral model is also known as the spiral lifecycle model (or spiral development), it is a systems development method (SDM) used in information technology (IT). This model of development combines the features of the prototyping model and the waterfall model. Figure 3.1 shows the spiral model:

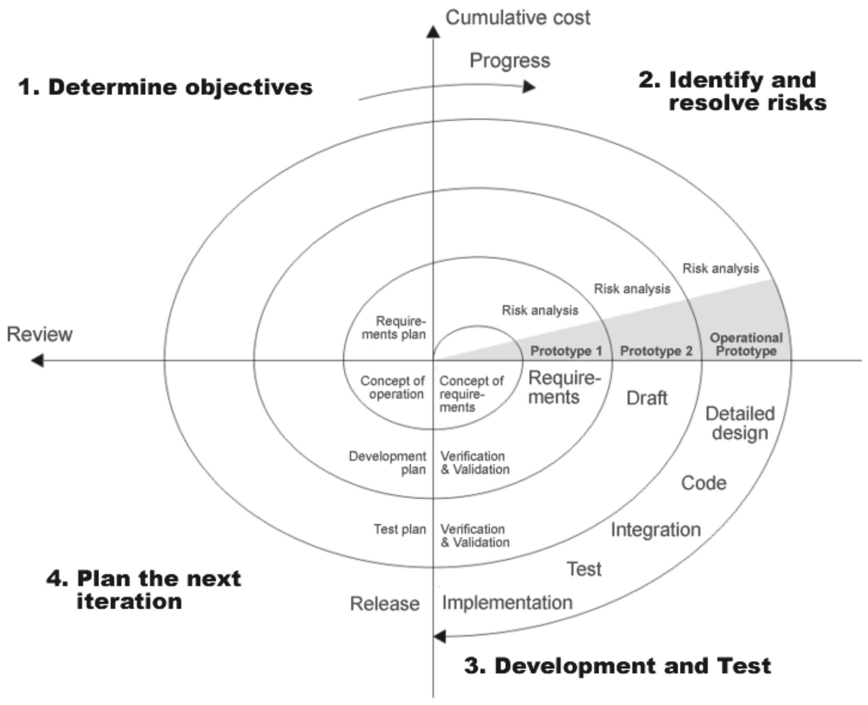


Figure 3.1: Spiral model

The IT model is defined as the study, design, development, implementation, support or management of computer-based information systems. And the waterfall model is a sequential software development process, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of conception, initiation, analysis, design (validation), construction, testing and maintenance.
The spiral model is intended for large, expensive and complicated projects.
The main advantages of this process are the following:

1. The spiral model promotes quality assurance through prototyping at each stage in systems development.

2. The software is produced early in the software life cycle.

3. It facilitates high amount of risk analysis.

The methodology followed in this project is composed of four stages:

1. The first stage consisted of determining objectives. In this phase of the project the customers need and requirements (problems and opportunities) were defined and documented in order to achieve a business outcome. Two meetings with the final users, the music programmers of Catalunya Ràdio, of the application were held. The user requirements were identified and also the technical requirements.

2. The second stage was identifying and resolving risks. A first proposal of the project was developed. It analyzed the requirements of the users and it also identified and evaluated the technical requirements. The range of possible solutions were analysed and were selected preferred options. It was studied the details of the cost, benefits and risks of the selected solutions.

3. The next stage was to implement and test the first prototype of the new system from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.

4. Then a second prototype is evolved by a fourfold procedure:

   (a) Evaluating the first prototype in terms of its strengths, weaknesses, and risks;

   (b) Defining the requirements of the second prototype;

   (c) Planning and designing the second prototype;

   (d) Constructing and testing the second prototype.

3.2 Planning and budget

The project is composed by four main milestones:

1. Design and implementation of the C++ Server

2. Design and implementation of the Database

3. Design and implementation of the Web application

4. CORBA connection between C++ Server and the Web Application

These four project milestones are connected amongst them. Consequently, the implementation is not completely modular. The implementation modules overlap in some periods of the project development. This project is developed by one person a part-time work. Thus, one day of work implies 4 work hours. This person performs different work types:

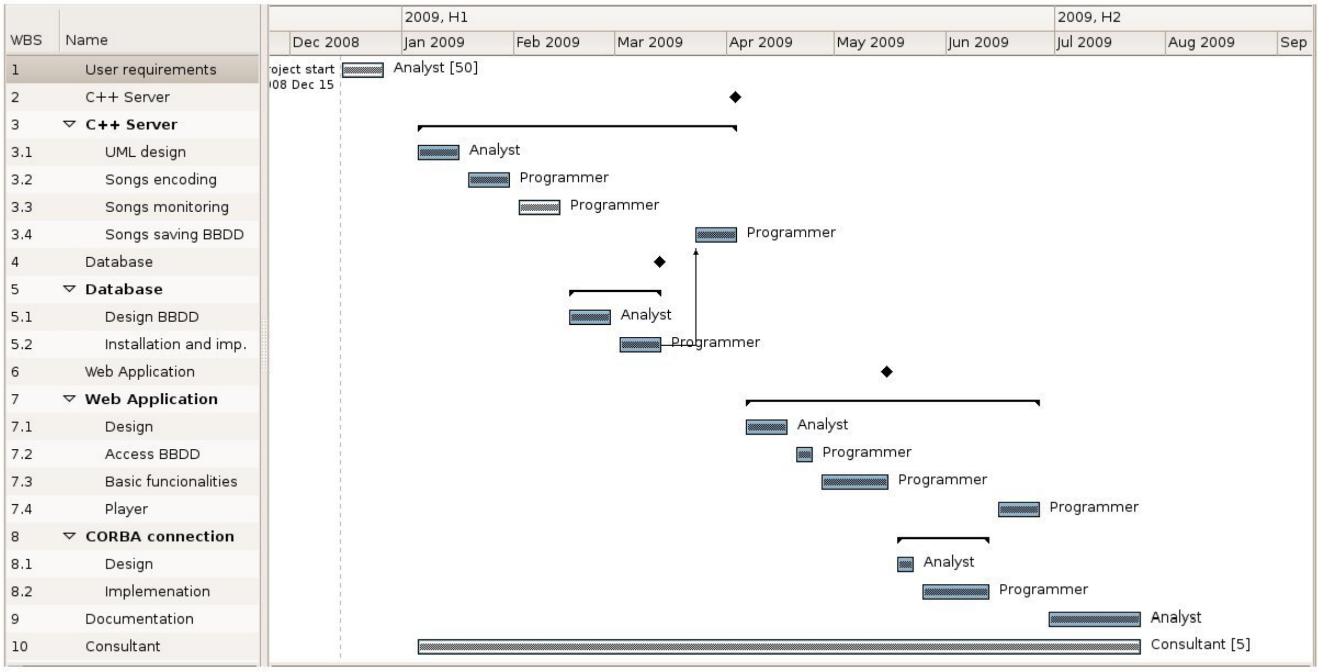
- Programmer: it includes implementation work.
- Analyst: it includes design and documentation work.

A consultant also is involved in the project and works one hour per week. The consultant tries to resolve the project bottlenecks and to improve the design and implementation of the project. Consequently, different tasks are performed related to three different employment status: programmer, analyst and consultant. Table 3.1 shows the prize per hour (in euros) of the three employment status:

Table 3.1: Employment status

Status	Assigned to	Cost(euros)
Analyst	Irene Zeller	33,20
Programmer	Irene Zeller	28,30
Consultant	Isaac Gelado	42,82

These prizes correspond to real prizes that the company, that performs this project, should pay per hour to the employees with these three employment status. If this project was sold to other companies these prizes will include an additional percentage.
The figure 3.2 shows a Gantt chart that illustrates the expected project schedule. It presents start and finish dates of tasks and summary elements of the project.



WBS	Name
1	User requirements
2	C++ Server
3	**C++ Server**
3.1	UML design
3.2	Songs encoding
3.3	Songs monitoring
3.4	Songs saving BBDD
4	Database
5	**Database**
5.1	Design BBDD
5.2	Installation and imp.
6	Web Application
7	**Web Application**
7.1	Design
7.2	Access BBDD
7.3	Basic funcionalities
7.4	Player
8	**CORBA connection**
8.1	Design
8.2	Implemenation
9	Documentation
10	Consultant

Figure 3.2: Final project tasks

Table 3.2 shows planned tasks (showed in the Gantt chart), the duration of these tasks and the expected resource usage:

Table 3.2: Project tasks

Identifier	Name	Start	Finish	Work(days)	Cost(euros)	Assigned to
1	SoPa project	Dec 15	Jul 24	Milestone	**18831.78**	
2	User requirements	Dec 15	Dec 26	5d	**664**	Analyst
3	C++ Server	Jan 5	Apr 3	20d	**4724**	
3.1	UML design	Jan 5	Jan 16	10d	1328	Analyst
3.2	Songs encoding	Jan 19	Jan 30	10d	1132	Programmer
3.3	Songs monitoring	Feb 2	Feb 13	10d	1132	Programmer
3.4	Songs saving BBDD	Mar 23	Apr 3	10d	1132	Programmer
4	Database	Feb 16	Mar 13	20d	**2460**	
4.1	Design BBDD	Feb 16	Feb 27	10d	1328	Analyst
4.2	Installation and implementation	Mar 2	Mar 13	10d	1132	Programmer
5	Web Application	Apr 6	Jun 26	20d	**4724**	
5.1	Design Web	Apr 6	Apr 17	10d	1328	Analyst
5.2	Access BBDD	Apr 20	Apr 24	5d	566	Programmer
5.3	Basic functionalities	Apr 27	May 15	15d	1698	Programmer
5.4	Player	Jun 15	Jun 26	10d	1132	Programmer
6	CORBA connection	May 18	Jun 12	20d	**2362**	
6.1	Design CORBA	May 18	May 22	5d	664	Analyst
6.2	Implementation CORBA	May 25	Jun 12	15d	1698	Programmer
7	Documentation	Jun 29	Jul 24	20d	**2656**	Analyst
8	Consultant	Jan 5	Jul 24	7d	**1241,78**	Consultant

However, the final budget is higher because of these reasons:

- Some implementation and design problem have delayed the implementation phase.
- New requirements have appeared.
- Modification in the requirements, set at the beginning of the project, have been identified.

The main areas, whose duration has been increased, and the increased days are:

- Song encoding: 5 days work.
- Song monitoring: 10 days work.

- Improvements in the Web: 10 days work.
- Player: 5 days work.
- CORBA implementation: 5 days work.
- Documentation: 5 days work.
- More hours of consultant work

Next chapters explain the problems emerged during the project and justify these increased expense. Consequently, the final budget is higher than the estimated and is 22689.32 euros, which is 3857.54 euros higher than the planned budget. Table 3.3 shows the final project tasks and costs:

Table 3.3: Final project tasks

Identifier	Name	Start	Finish	Work(days)	Cost(Euros)	Assigned to
1	SoPa project	Dec 15	Jul 24	Milestone	**22689.32**	
2	User requirements	Dec 15	Dec 26	5d	**664**	Analyst
3	C++ Server	Jan 5	Apr 3	55d	**6523.88**	
3.1	UML design	Jan 5	Jan 16	10d	1328	Analyst
3.2	Songs encoding	Jan 19	Feb 6	15d	1698	Programmer
3.3	Songs monitoring	Feb 2	Oct 23	20d	2365.88	Programmer
3.4	Songs saving BBDD	Mar 23	Apr 3	10d	1132	Programmer
4	Database	Feb 16	Mar 13	20d	**2460**	
4.1	Design BBDD	Feb 16	Feb 27	10d	1328	Analyst
4.2	Installation and implementation	Mar 2	Mar 13	10d	1132	Programmer
5	Web Application	Apr 6	Jun 26	55d	**6444.64**	
5.1	Design Web	Apr 6	Apr 17	10d	1328	Analyst
5.2	Access BBDD	Apr 20	Apr 24	5d	566	Programmer
5.3	Basic functionalities	Apr 27	Jun 19	15d	2852.64	Programmer
5.4	Player	Jun 15	Jul 24	15d	1698	Programmer
6	CORBA connection	May 18	Jun 12	25d	**2928**	
6.1	Design CORBA	May 18	May 22	5d	664	Analyst
6.2	Implementation CORBA	May 25	Jun 12	20d	2264	Programmer
7	Documentation	Jun 29	Jul 24	25d	**3346,56**	Analyst
8	Consultant	Jan 5	Aug 28	8d	**1370. 24**	Consultant

Chapter 4

Project requirements

4.1 User requirements

The implemented system is a specific development for professional users of media sector, specifically for musical programmers of Catalunya Ràdio. User requirements is a key step to produce a system that can be used easily. This is the reason why two meetings have been held between the musical programmers of Catalunya Ràdio and the technical department of CCMA. The main purpose of the meetings was to identify the user requirements and how to apply these requirements to the system design and implementation.

The problem presented by musical programmers was the difficulty and high expense of annotating the songs parameters in a completely manual way. They consider that a semi-automatic system would be very useful for them, and it would help to standardize the criterion of annotating songs. In these meetings a set of requirements was identified:

- Music programmers consider that it is not possible to automate completely the song annotation process. Although, the automatic system would be very accurate, the probability of a correct anotation will never be very high because of the divergences in the criterion between different music programmers. So, it is basic to include the feature of validating all the parameters extracted by the application server. Thus, the developed system will be semi-automatic. It will suggest a parameter and final users will validate or change it.

- Ideally, the system should be intelligent and it should be learning with user validations. Consequently, the probability of a correct annotation will increase as songs are added to the system.

- The generated data should be portable to other systems in an easy way. Currently, music programmers use a program to select the broadcasted songs called Music Master, and all the generated data should be showed from these program. However, the whole system may be completely independent of Music Master, so this program ought be changed in the future.

- Currently, music programmers do the work of annotating songs from the Music Master application.

This application does not allow listening to the songs they are working with. A good alternative for them would be to implement a new system user interface to let users interact with the proposed parameters and to listen to the song from it.

- The system user interface should be connected to the data base of the Music Master. The information generated by the system about processed songs would be saved in a new database, as well as in the Music Master database.
- The system user interface should let users manage the songs. It should contain a simple and advanced search.
- The designed system must let users easily change the parameters and its ranges. Each music programmer has her or his own criterion. Not only with the final parameters, but also in the ranges and the types of parameters.
- Each new song added to the system should be processed and stored. When users start the system, the new songs should be indicated.
- The system management should be simple and rapid.

4.2 Technical requirements

Other type of requirements have to be take into account. Both companies, CCMA and Noufer, have technical requirements that have to be followed in the project implementation. Some other technical requirements have been imposed to achieve a more efficient and stable system. The project is composed mainly by three parts:

1. The server engine that manages the audio processing of the songs.
2. The database where the generated data is stored.
3. The Web Application, where data is presented to final users.

The technical requirements identified are the following, formed by functional and not functional requeriments:

4.2.1 No functional requirements

1. The final user will connected to the developed system from Windows operating system.

2. Songs are stored in Windows servers and their format is MPEG-1 (Audio Layer 2), called MP2. It is an audio compression format defined by ISO/IEC 11172-3 together with MPEG-1 Audio Layer I and MPEG-1 Audio Layer III (MP3). While MP3 is much more popular for PC and Internet applications, MP2 remains a dominant standard for audio broadcasting.

3. The library for the audio processing is compiled in Linux operating system. The required format of input to the process is WAV format, 22050 Hz, 16 bits and one channel.

4. Users mainly use Windows operating system.

5. The server engine must be available 24 hours per day.

6. The generated code should be as efficient as possible to let the system be as faster as possible.

7. The system should be easily configurable.

4.2.2 Functional requirements

1. The server engine should identify new songs automatically.

2. The system should present the generated data.

3. The system should contain a search engine.

4. The system should allow final users to listen to the songs.

5. Administrator users should control the server engine from the Web.

4.2.3 Database requirements

1. Two different databases should be installed. One for saving the information about songs and other one for managing Web Application users.

Table 4.1 shows all identified requirements, technical and functional, of the whole system:

Table 4.1: Project requirements

Identifier	Description
REQ1	The system should be completely independent
REQ2	Final users use Windows O.S.
REQ3	Songs are stored in Windows Server
REQ4	The AudioTool library is compiled for GNU/Linux
REQ5	Songs format is MPEG-1 (Audio Layer 2)
REQ6	The required format of songs is WAV, 22050 Hz, 16 bits and one channel
REQ7	System must be stable
REQ8	Server must be available 24 hours per day
REQ9	The system should be rapid
REQ10	The system should be easily configurable
REQ11	New Songs should be automatically identified
REQ12	Data should be presented to final users
REQ13	Data should be able to be validated
REQ14	Users should be able to listen to songs
REQ15	Several users can be connected to the Web Application at the same time
REQ16	Data related to songs should be independent to data related to users

Chapter 5

Implementation

Several implementation and design decisions have been made to meet the user requirements during the implementation of the system. This chapter explains these decision in a detailed way, and compares each adopted implementation approach with other choices that have not been taken. Here, we also describe the emerged problems during the system development and the way they have been solved.
Figure 5.1 shows the main three modules of the designed system:

1. Server Engine: it manages the audio processing for the songs.

2. Database: it stores the data generated in the audio processing.

3. Web Application: it presents the data to professional users.

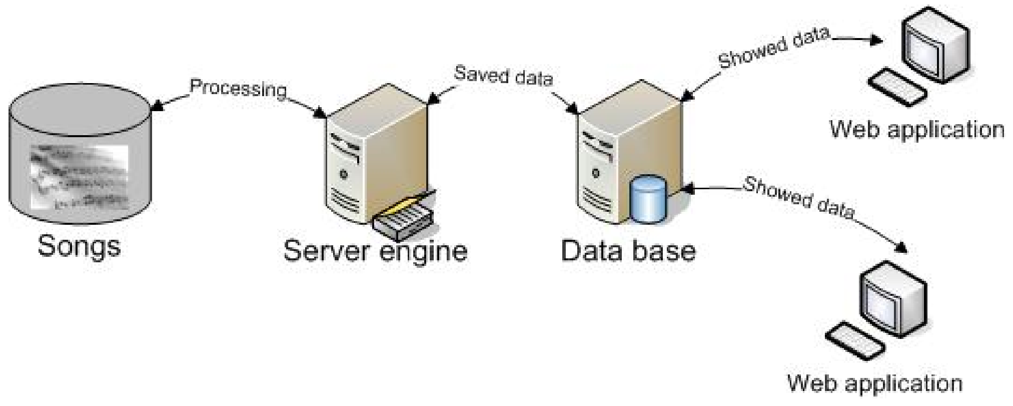


Figure 5.1: Main modules of the system

All these three modules are connected and interact amongst them. This chapter explains the three modules and the way they interact at different levels of design.

5.1 System Environment

The C++ server engine runs on a GNU/Linux machine. The Audiotool library is available: both, for Windows and GNU/Linux operating system. First, a performance test was done for each of the available versions of the library.

The Windows version does not make things easy for audio processing. This version has dependencies with DirectX [5] and with Microsoft VisualC++. These are clear disadvantages, since the installation of this library is more complex in Windows than in GNU/Linux and an standard integrated development environment (IDE), like Eclipse or NetBeans, is not possible to use in an easy way.

Other disadvantage is the needed time for processing songs (much more longer than in Linux). In the first test, with the libraries compiled for Windows, the time for encoding and compiling a song was more or less the duration of the song. In contrast with the last version of the application compiled for Linux, the required time to encode and process a song of 4 minutes duration is about 3 seconds. This important variation in the time for processing a song depends on more factors, not only on the operating system. The last approach has several code improvements and the way the songs are encoded is also improved. However, the operating system and how the libraries are compiled are the most significant factors.

Despite of the fact that the final users and the technical department of the company utilize mainly Windows machines and not GNU/Linux, the advantages were significant enough to choose the Linux compiled version and, consequently, to work in GNU/Linux operating system. Moreover, the design will include an OS-independent client application.

It is clear that one of the main GNU/Linux features is its stability, and it is recommended for working in networks. Thus, the database is also installed in a Linux machine as well as the Web Server JBoss.

The songs servers are installed in Windows machines. The server engine is connected to these Windows server with the CIFS networking protocol, implemented by the Samba server.

User connects to the Web Application mainly from Windows machines. However, the system is implemented to let users connect from any operating system. We follow the W3C [34] standards and, thus, any browser running on any operating system works with the Web Application. The server application has been developed and compiled in Linux system.

Figure 5.2 shows the run-time environment of each of the modules:

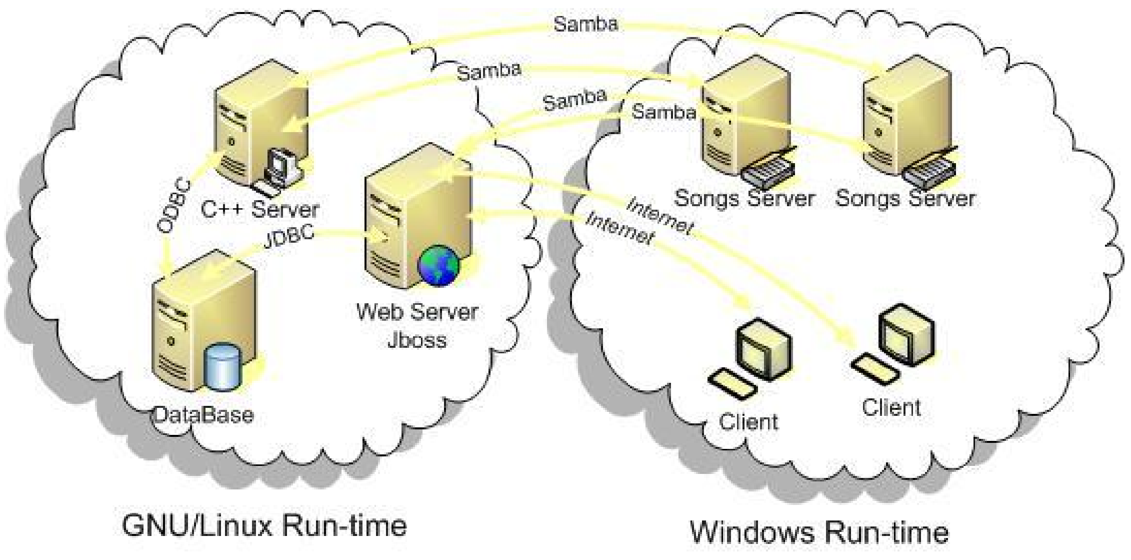


Figure 5.2: Run-time environment of each of the modules

5.2 Server engine

The server engine is written in C++ and it has been designed using Object-oriented programming. The C++ application has been designed to exploit the main features of this type of programming. It is prepare to provide code reuse, and self-sufficient reusable units of programming logic, enabling collaboration through the use of linked modules (subroutines). It tends to consider data and behaviour separately. Two main references [35] [44] have been followed to design and program the C++ application.
The server engine has two main functionalities:

- Management of the audio processing workflow.
- Web Application clients interaction.

Figure 5.3 shows the main steps of the management audio processing workflow:

- Identifying new songs added to the system. The application reads from the database the servers to scan or monitor, waiting for new songs.
- Encoding new songs to the correct format to process them. A new encoded song is stored temporarily in the server machine.
- Processing the encoded song with the AudioTool library. After that, the new song is deleted from the server.

- Saving generated data into the database and some other information about the song, such as the path of the song.

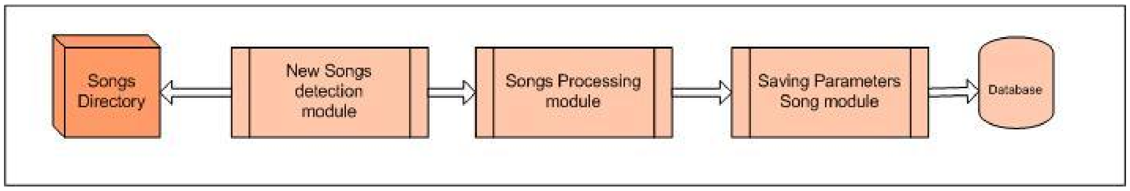


Figure 5.3: Audio processing workflow

In order to allow the interaction between the C++ Server and the Web Application, both components are connected. The middleware used to connect them is CORBA. The server engine is prepared to reply three requests types:

- To reload the parameters of the configuration analysis saved in the system.
- To stop the functionality of processing new songs.
- To start again the functionality of processing new songs.

Here, the two main functionalities of the application, **management of the audio processing workflow** and **Web Application interaction**, are detailed.

5.2.1 Audio processing workflow

The implemented classes to achieve the main processing song modules are outlined in figure 5.4 :

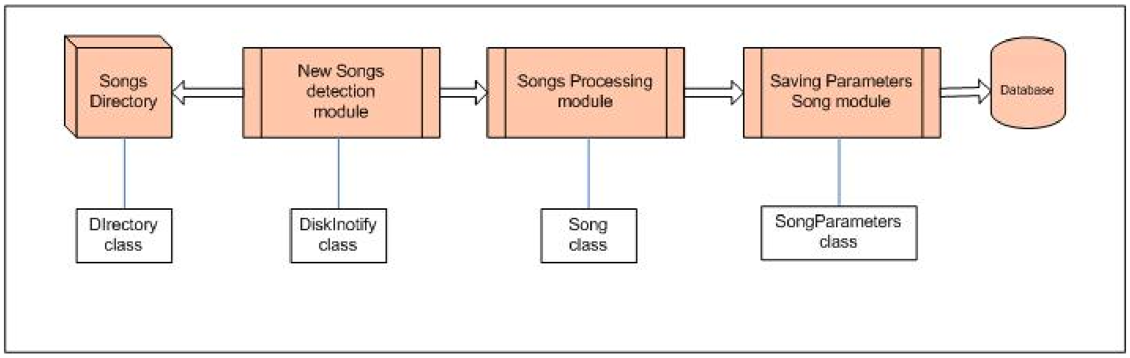


Figure 5.4: Implemented classes

Below, the main modules of the song processing and the implemented classes are explained. Also, the problems emerged during the implementation and the decisions that were made for each problems are detailed.

New songs detection module

A big development setback that we found was the event notification. The first library, that we chose, to detect events in the songs directories was FAM (File Alteration Monitor). We thought that FAM would be better than other options so it enables applications to work on a greater variety of platforms, and not only on GNU/Linux operating system. However, some problems emerged during the integration of the FAM library. FAM only detects two types of events: when a new file is created and when some file has changed. The application processes the songs as soon as it identifies a new song, whether the song has been fully written to disk or not. The code using FAM tend to encode the songs before they are completely copied and, as a result, the process fails. This problem has not straightforward solution. Although changed events are notified while the songs are being copied, it is not possible to recognize which changed event is the last one and, consequently, the song is completely copied. The unique solution, that we found, is to introduce in the process a time out when a new file notification is identified. This solution is inaccurate, since it is really complex to estimate the required time to copy a song. This time depends on different factors, such as song size or network features. If the estimate time is much longer than necessary, the process is not efficient and, if is too short, is easy to encode the song before the song is copied.
Other possible solution is to try to encode in each event, even though the application returns an error, until the song is completely copied. Nevertheless, it is not a possible solution, since the libraries returns the error -84, not only when the song is not completely copied, but when the library does not recognize the type of file. Thus, it will appear also if a directory is being tried to encode. This solution may cause fatal infinite loops in the application.
As a consequence, the Inotify library is chosen to perform the event notification. Inotify allows to identify more events, and especially when a file is closed and thus completely copied. Therefore, the problem is solved with this library. It also has other advantages: Inotify is a Linux kernel subsystem and does not need an external library and it is also faster than FAM.

Other big setback is to identify new songs in Windows Servers. Windows operating system uses different file-system to GNU/ Linux. GNU/Linux uses a virtual file system (VFS) to access to files. The purpose of the VFS is to allow client applications to access different types of concrete file systems in a uniform way. A VFS can, for example, be used to access local land network storage devices transparently without the client

application noticing the difference. It can be used to bridge the differences in Windows, Mac OS and Unix file-systems, so applications can access files on local file systems of those types without having to know what type of file system they are accessing.
Nevertheless, Inotify is directly connected to VFS and only access to the files the first time. Hence, it is not able to see the updates done in the CIFS servers, except if the updates are performed from a Unix machine connected to the CIFS server. We thought that SAMBA was able to create a bridge directly to Inotify (without VFS) and the updates in the server were noticed by the Inotify process. Nonetheless, this functionality is yet not implemented and currently it is not possible to detect updates in a Windows server with Inotify. A clear requirement defined by final users is that songs are stored in Windows Servers, so a solution should be achieved not using file-system events.
Any possible solution is as efficient as desired. Moreover, it is probably that, in a near future, network file updates will be implemented in GNU/Linux. Hence, we keep the Inotify process (it is the optimal way to identify new files) and implement a new (temporary) solution in the main application to identify updates in the network file-system. This design allows us to delete the temporary solution in an easy way.
The new file-system change approach is implemented as a new class which runs as a new thread. It contains a timer and on each time out it rescans the Windows servers and compares the scanned files with the latest scan. If a new song is found, it will be processed.

This new approach has concurrency problems. If the song is copied from a GNU/Linux machine, the Inotify process has concurrency problems with the new scan process, and new song should be processed twice. In order to avoid this problem a protected variable type semaphore is included in the application. This semaphore restricts the access to shared resources in this parallel programming environment, and avoids Inotify identifies a song as new if before it has been identified by the other process. Hence, the rescanning process is atomic (it is not interrupted) and when a new song is identified by Inotify, first of all it makes sure that the new song has not been identified before by the other scan process. To ensure that this song has not been processed before by the other process, a C++ standard template library container is used: set type. Set is a kind of associative containers that stores unique elements, and in which the elements themselves are the keys. Thus, it is used to avoid repeating the process. If an element with the same value exists in the container, the insert function returns false. Figure 5.5 shows how the Inotify process tries to insert new songs in the set container of the scan process to ensure that these songs have not been processed before.

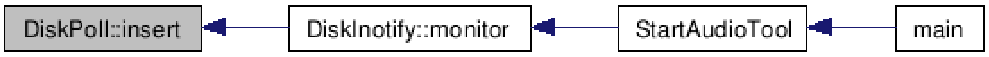


Figure 5.5: Insert new song in the set container of DiskPoll

Therefore, it is not possible to separate completely the Inotify process of the Windows scan process because of the concurrency problems. The Inotify class should use the Windows scanning process. To solve this problem, a new input in the configuration file of the application is created. The application only creates the new thread and, thus, the scanning process if it is indicated in an input of the configuration file. Figure 5.6 shows the configuration file of the application, and the line that indicates this thread should be created:

```
    # configuration file audiotool.inp
windows share = true # if is connected to a Windows server
bbdd name = postgre  # name bdbb
bbdd user = postgres   # user bbdd
bbdd password = postgres # password bbdd
IP Corba = 192.168.163.17 #IP Corba
Port Corba = 1025 #Port Corba
```

Figure 5.6: Configuration file input

Both classes, the Inotify process (DiskInotify) and the Windows scanning (DiskPoll), perform some modules in common. The functionalities of both classes are exactly the same:

- To read from the database the directories to scan or to monitor.
- To scan or to monitor these directories
- To identify new songs
- To order to other class to process the song

Actually, each class has its own features, but some units of programming logic are exactly the same and they can be reusable. Therefore, the better approach is to create a base class (called Disk) that provides the functionalities in common and derived classes that inherit attributes and behaviour of the base class, as shown in figure 5.7 :

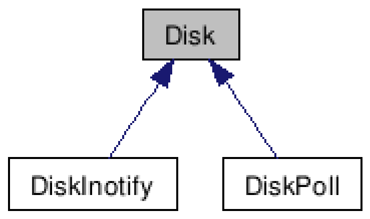


Figure 5.7: Class Disk, DiskInotify and DiskPoll

Songs processing module

The implemented class for processing new songs is the class Song. The class Song has two main functionalities:

- To encode songs to the correct format
- To process encoded songs

Songs Encoding The first design approach made use of the FFmpeg command line tool to encode the songs. The integration was really simple, so only one command line lets the application change songs format. However, debugging the code and identifying errors was really complicated and the integration design was not stable.
Hence, in the second design of the system, the libraries libavcodec and libavformat were used to encode the songs. FFmpeg uses both libraries. The new approach was more complex and it was necessary an on-line article [37] about how to use these libraries. This article covers how to read video and not audio. It seems that audio streams should work pretty much the same way. Nonetheless, some important differences forced to find other complementary documentation to achieve to use these libraries.
The songs dataset delivered to perform the first assessment of the libraries contained several corrupted songs. It caused a lot of interruptions in the application that were controlled with the new way to encode songs.
The new way to encode songs is able to provide all audio input formats and audio output formats supported by FFmpeg. All input formats are directly supported and the output format should be easily changed. The class contains four private functions that allow changing the format, as shown in the figure 5.8 and in the figure 5.9 :

- **void dumpFile (std::string format)** Sets the new format to encode
- **void setbitRate (int bitRate)** Sets the bit rate.

- **void setsampleRate (int sampleRate)** Sets the sample rate.
- **void setchannels (int channels)** Sets the channels.

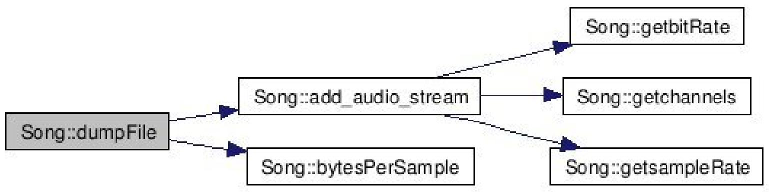


Figure 5.8: Getting the parameters of the song

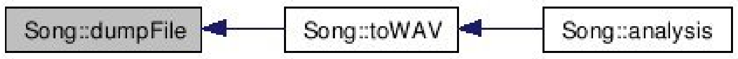


Figure 5.9: Setting the format

Song Processing A new encoded song is temporarily stored in the local machine. The Song class processes the new song with the library AudioTool. The generated information about the song processing is passed as parameters to the new class SongParameters. Eventually, the encoded song is deleted from the local disk.

Saving parameters song module

The last step is to save the generated parameters into the database. The SongParameters class performs the task of saving the song information generated before. In order to achieve this task, more classes, related to the database, are necessary.

The connection to the database is controlled by the class ControlODBC. This class opens and closes the connection, as well as facilitates performing a query or cleaning the connection resources. ControlODBD class is used by all application methods that need some interaction with the database.

The user requirements reveal that seldom only one song will be processed. A set of songs usually will be added to the servers and processed. Hence, it is obvious that open and close the connection with the database in each interaction was not optimal. Thus, other class is necessary to optimize the interactions with the database in this application module. The new implemented class to manage the database and to optimize the connection is the class BBDDManager. This class closes the connection when detects that no

interactions have been performed in a period of time.
Only in this situation is necessary to manage in this way the database connection, Thus, the SongParameters class is like a BBDDManager and it has been implemented as a base class and derived class, as shown in the figure 5.10 and in the figure 5.11 :

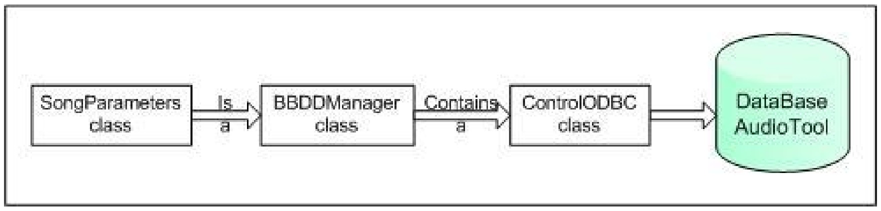


Figure 5.10: Implemented classes

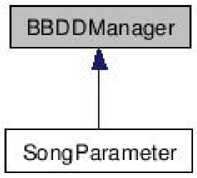


Figure 5.11: BBDDManager and SongParameter

The BBDDManager class contains a signal handler that has an alarm. The alarm alerts when there are not actions in the database in a fixed time (the fixed time is 10 seconds, but it could vary if necessary), and then the application disconnects from the database. When a new interaction with the database is performed the alarm is set to 0 again. This approach allows the system to be more efficient with the database interactions.
Figure 5.12 and 5.13 shows how the alarm is set to 10 seconds and how the timer closes the connection:

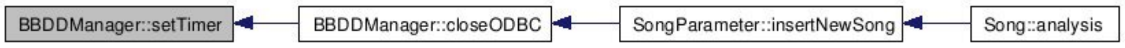


Figure 5.12: Setting the alarm to 10 seconds

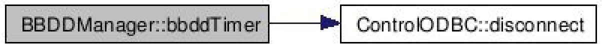


Figure 5.13: Disconnect the database when the timer is out

It is important to take into account that all methods in C++ receives an implicit parameter that consists of the memory direction of the invoked object (this). As a result, a normal method is not able to implement a signal handlers. Thus, the method with the signal handler should be a static method.

5.2.2 Web Application clients interaction

The second functionality of the C++ application is the connection with the Java Web Application. This feature lets users manage and have remote control of the C++ server from the Web Application. C++ application and Web Application are connected by means of CORBA middleware.
Two main questions have been solved to meet this requirement: the CORBA connection and the implementation of different methods that perform the required actions. Next, these two topic are explained in a detailed way.

CORBA connection The CORBA connection is performed in a different thread instantiated in the class ConfigurationAnalysis. A method has been implemented to perform this task: orbThread. It runs in an independent thread. Figure 5.14 shows this method:

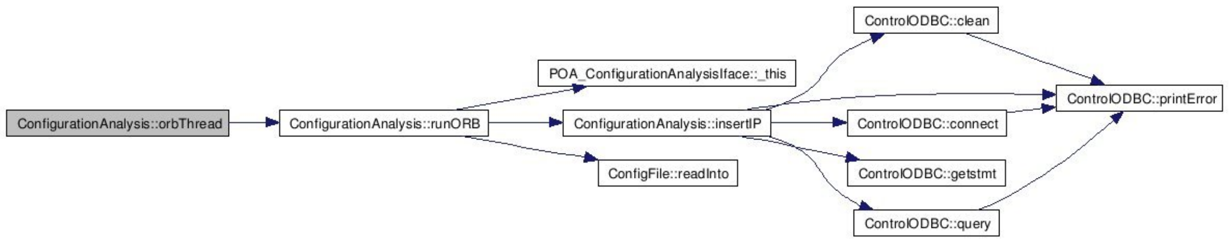

ConfigurationAnalysis::orbThread
ConfigurationAnalysis::runORB
POA_ConfigurationAnalysisIface::_this
ConfigurationAnalysis::insertIP
ConfigFile::readInto
ControlODBC::clean
ControlODBC::connect
ControlODBC::getstmt
ControlODBC::query
ControlODBC::printError

Figure 5.14: CORBA Connection method

Figure 5.14 shows that this method uses the class ControlODBC, and, thus, the method uses the database. It uses it to save the connection data or IOR as a string. The Interoperable Object Reference (IOR) defines the format of a reference to a remote object. The typical IOR usually contains the protocol version, server address and a byte sequence identifying the remote object (object key).
The Web Application needs this objects to communicate with the server application. Both, Web Application and C++ server, share the same database, so it is possible to share the IOR object as a string through the database.
Typically, IOR is shared using the Naming Service. The Naming Service is an object repository. Application

servers export object references into the naming service, providing an associated name. The naming service stores these object references in its database, keyed by the supplied name. Later, clients retrieve objects from the naming service by providing a name. The naming service returns the object reference with the matching name. However, this service adds load to the system. Moreover, a shared database is yet implemented and it is the easiest way to share the IOR object.

CORBA (more precisely, IIOP) uses raw TCP/IP connections in order to transmit data. Nonetheless, if the client is behind a very restrictive firewall or transparent proxy server environment that only allows HTTP connections to the outside through port 80, communication may be impossible, unless the proxy server in question allows the HTTP CONNEC method or SOCKS connections as well. Last CORBA implementations, though, support SSL or TCP transport and can be easily configured to work on a single port. OmniORB4 can support multiple network transports. All platforms (usually) have a TCP transport available. Unix platforms support a Unix domain socket transport. Platforms with the OpenSSL library available can support an SSL transport.

The first CORBA connection approach, developed for this project, was not configured to work on a single port. It uses raw TCP/IP connection and the firewall made impossible the communication. The option to configure the connection parameters of the omniORB4 was analysed. These parameters are configured by means of command lines argument. However, the CORBA connection is implemented with a thread in a static class, and , thus, it is firstly created. Hence, it is not possible to pass the arguments through the arguments of the main class. CORBA has its own configuration file that solve this problem. However, this application yet contains one configuration file and managing the application with two configuration files may be confusing. Thus, the configuration file of the application is used to enter the desired connection dates. This file contains two inputs to configure the CORBA connection: one for the port and other for the IP, as shown in the figure 5.15 :

```
    # configuration file audiotool.inp

windows share = true # if is connected to a Windows server

bbdd name = postgre  # name bdbb

bbdd user = postgres   # user bbdd

bbdd password = postgres # password bbdd

IP Corba = 192.168.163.17 #IP Corba

Port Corba = 1025 #Port Corba
```

Figure 5.15: Configuration file for IP and Port CORBA connection

The application reads these inputs and performs the line argument, as shown in figure 5.16 :

```
ConfigFile config( "audiotool.inp" );

std::string IP;

std::string Port;

if ((!config.readInto(IP,"IP Corba"))||(!config.readInto(Port,"Port Corba"))){

FATAL("Port or IP of Corba");

}

std::string ORB("endPoint");

std::string IP_Port("NULL");

IP_Port.assign("giop:unix:").append(IP).append(":").append(Port);

int argc = 0;

char**argv = NULL;

std::cout<<IP_Port<<std::endl;

const char* options[][2] = { { ORB.c_str(), IP_Port.c_str() }, { 0, 0 } };

orb = CORBA::ORB_init(argc,argv,"omniORB4",options);
```

Figure 5.16: Application code where the Endpoint command line is performed

The command line arguments takes the form -*ORBEndpoint*, followed by the details of the connection. If the line argument is not passed from the main class, as in this application, it takes the form *endPoint*, as shown in the figure 5.16.

These details are selected with the endpoint parameters family. An endpoint is the name for the entity on one end of a transport layer connection. The simplest is plain endPoint, which chooses a transport and interface details, and publishes the information in IORs. Endpoint parameters are in the form of URIs, with a scheme name of **giop:**, followed by the transport name. Different transports have different parameters following the transport.
Figure 5.17 shows the alternatives IIOP addresses [28] :

```
-ORBendPoint <endpoint uri>
-ORBendPointNoPublish <endpoint uri>
-ORBendPointNoListen <endpoint uri>
         <endpoint uri> = "giop:tcp:<host>:<port>" |
                         *"giop:ssl:<host>:<port>" |
                         *"giop:unix:<filename>"   |
                         *"giop:fd:<no.>"          |
                         *"<other protocol>:<network protocol>:<options>"
                         * may not be supported on the platform.
```

Figure 5.17: Multiple alternative IIOP addresses

The host must be a valid host name or IP address for the server machine. It determines the network interface on which the server listens. The port selects the TCP port to listen on, which must be unoccupied. Either the host or port, or both can be left empty. If the host is empty, the ORB publishes the IP address of the first non-loopback network interface it can find (or the loopback if that is the only interface), but listens on all network interfaces. If the port is empty, the operating system chooses a port.
Multiple TCP endpoints can be selected, either to specify multiple network interfaces on which to listen, or (less usefully) to select multiple TCP ports on which to listen.
If no endPoint parameters are set the ORB assumes a single parameter of **giop:tcp::** , meaning IORs contain the address of the first non-loopback network interface, the ORB listens on all interfaces, and the OS chooses a port number.
Servers must be configured in two ways with regard to transports: the transports and interfaces on which they listen, and the details that are published in IORs for clients to see. Usually the published details will be the same as the listening details, but there are times when it is useful to publish different information.

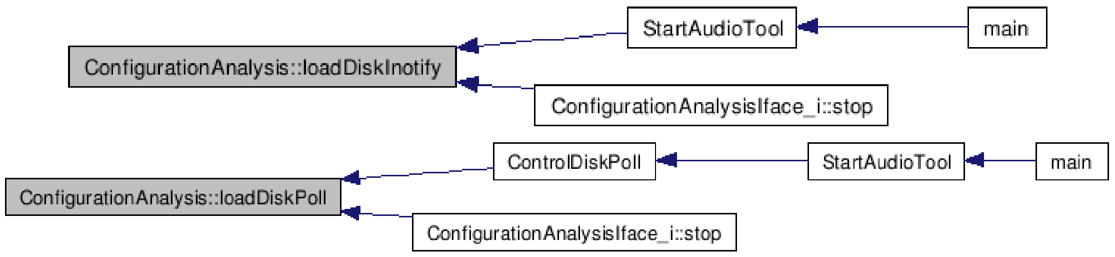
CORBA Methods The connection purposes are three:

- To reload the parameters of the configuration analysis saved in the system.
- To stop the funcionality of the server of processing songs.
- To start again the funcionality of processing songs.

The class ConfigurationAnalysis, where the CORBA connection thread has been created, contains three methods to reply the request of the Web Application. These methods are:

- **void load (void)** It reloads from the database the configuration parameters of songs analysis. These parameters are stored in the same class ConfigurationAnalysis. Thus, when the class Song is processing a song uses this class to get the configuration parameters.
- **void cleanDisk (void)** It deletes objects and threads that perform the task of detecting new songs in directories. These objects and threads can be:
  - Only a DiskInotify instantiation, created by the thread called StartAudioTool.
  - Or also a DiskPoll instantiation, created by the thread called ControlDiskPoll. This thread and this class are only created if the application is connected to Windows directories.

  These instantiations are loaded by the class main in the class ConfigurationAnalysis to let this class delete them. Figure 5.18 shows how the main class creates both threads, StartAudioTool and ControlDiskPoll, and both instantiations are loaded in the class ConfigurationAnalysis. When a stop request arrives, this class deletes both classes and set both variables to NULL.



ConfigurationAnalysis::loadDiskInotify
StartAudioTool
main
ConfigurationAnalysisIface_i::stop
ConfigurationAnalysis::loadDiskPoll
ControlDiskPoll
StartAudioTool
main
ConfigurationAnalysisIface_i::stop

Figure 5.18: Loading classes to ConfigurationAnalysis

- **void playDisk (void)** It creates new instantiations of the classes that have been deleted by the previous method: cleanDisk. It is solved like a concurrency problem (details to follow).

Consequently, more than one class needs the ConfigurationAnalysis class. It should be shared only one instantiation of this class to coordinate actions across the application. In order to share this single instantiation of the ConfigurationAnalysis class a design pattern Singleton [40] has been implemented. Figure 5.19 shows the singleton ConfigurationAnalysis class, that contains a single instantation performed by itself.

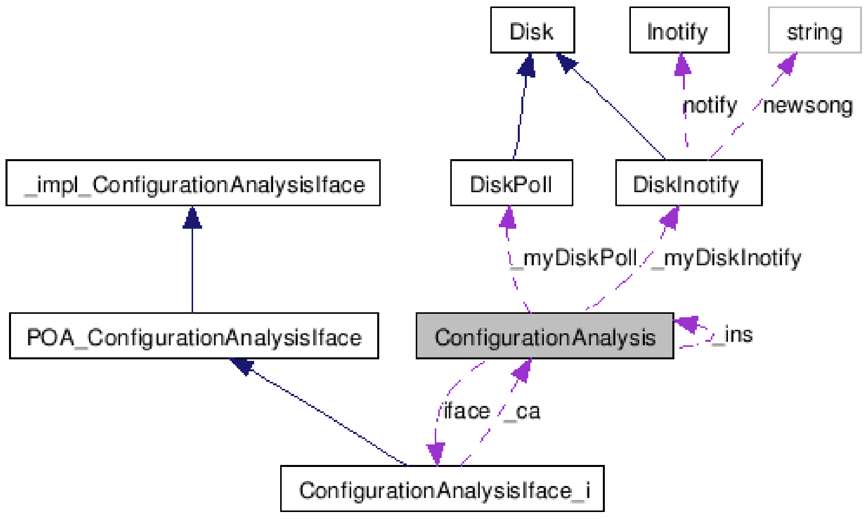


Figure 5.19: The Singleton class ConfigurationAnalysis

Figure 5.20 shows the singleton class shared by other classes.

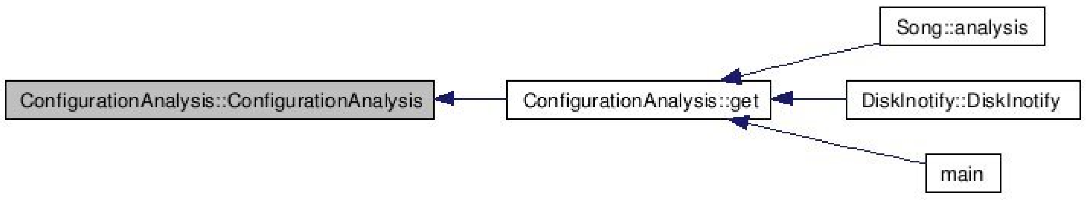

Song::analysis
ConfigurationAnalysis::ConfigurationAnalysis
ConfigurationAnalysis::get
DiskInotify::DiskInotify
main

Figure 5.20: Classes that use the Singletion class ConfigurationAnalysis

The functionality of playing and stopping the song processing of the application is solved like a concurrency problem. The main function creates both threads that perform the task of detecting new song. These threads are created and deleted if a request from the Web Application arrives. These threads are created inside a while. The figure 5.21 shows how this while works:

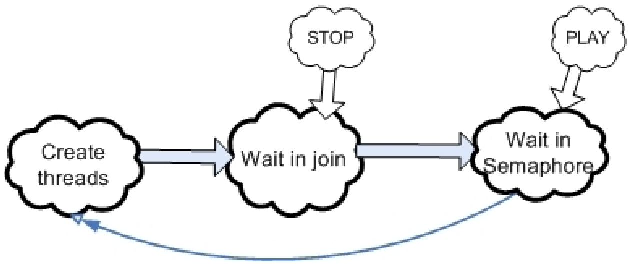


Figure 5.21: The creation threads while

First of all, the main class creates both threads (or only one if the application is not connected to Windows servers). This action is performed in an atomic way. Hence, until both threads and both instantiations (one in each thread) are created other actions can not be executed. This function also loads these instantiations to the class ConfigurationAnalysis.
After, the main class waits until a stop request arrives and, thus, these threads are cancelled. This action is performed thanks to the function: `pthread_join(pthread_t thread, void **valueptr)` , that suspends execution of the calling thread until the target thread terminates.
Then, the main class stops being blocked in the `phtread_join` function. Then, it finds a semaphore and waits in this semaphore until the action of playing the server again is called. When some user from the Web requests this action, this semaphore is released and, consequently, the main function starts again the while and creates again the threads.
This semaphore is created in the ConfigurationAnalysis class and shared with the main class. The class ConfigurationAnalysis contains the stopping and playing methods that unlock and lock this semaphore. Figure 5.22 shows this semaphore called by the main class:

ConfigurationAnalysis::getSemaphore
main

Figure 5.22: Semaphore used by the main function

Previously, we have explained how the main function manages the creation of threads. However, how the methods of playing and stopping works and how the instantiations are deleted and the threads are cancelled have not been explained. Next, these point are detailed.
First of all, when the object DiskInotify and DiskPoll are created, the thread identifier in question, that creates this objects, is passed as a parameter. In the case of DiskInotify, both identifiers, of both threads, are passed as parameters. If the second thread is not created this parameter is passed as a NULL. Figure 5.23 shows the constructors of both objects:

```
DiskPoll::DiskPoll( pthread_t threadidDiskPoll)

DiskInotify::DiskInotify(pthread_t threadid, pthread_t threadidDiskPoll)
```

Figure 5.23: Constructors of DiskPoll and DisInotify, where the threads identifiers are passed

The class DiskInotify contains a signal SIGINT that deletes the resources when a user uses the key combinations Ctrl-C. Therefore, this class needs both identifiers to delete both threads, if not the process does not suspend completely the execution. When a stop request is received, the ConfigurationAnalysis class deletes both instantiations, DiskInotify and DiskPoll. When these instantiations are deleted the destructor method of each one is executed. These classes contains the thread identifier, passed in the constructor. The destructor method uses them to cancel the thread in question. Both methods, the stopping method and the playing method, are atomic units of work. If not, important concurrency problems appear. Both methods share a boolean variable that indicates if the server is working or not. Thanks to this variable, threads are created or cancelled only if it is necessary. For example, if a user performs a stop request twice, only the first time will be performed. However, it is a dangerous shared resource, since if the boolean was changed erroneously, the application would be frozen and it would not be possible to start it again, unless the application would be restarted. To avoid this problem a semaphore is used. Figure 5.24 shows the semaphore shared amongst the stopping and playing method, and also the methods that perform the audio processing. This semaphore is used as a single instantiation thanks to the Singleton ConfigurationAnalysis class.

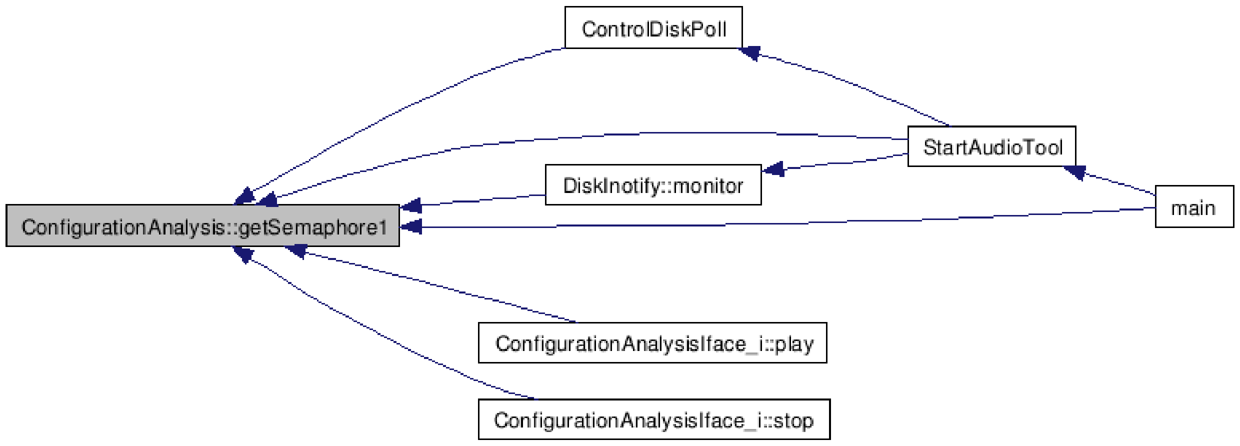

ControlDiskPoll
StartAudioTool
DiskInotify::monitor
ConfigurationAnalysis::getSemaphore1
main
ConfigurationAnalysisIface_i::play
ConfigurationAnalysisIface_i::stop

Figure 5.24: Semaphore

5.2.3 Multi-thread Application

To sum up, this application is composed of three or four threads. It depends of if the application is connected to a Windows server and, thus, a DiskPoll is created. Both threads, the main thread and the thread that performs the CORBA connection, are never cancelled; unless, obviously, that the application ends. The other two threads can be cancelled if a play request from the Web arrives, and created again if a stop request arrives. The thread that firstly is created is the thread that performs the CORBA connection, since it is created in a static class. Figure 5.25 shows these threads:

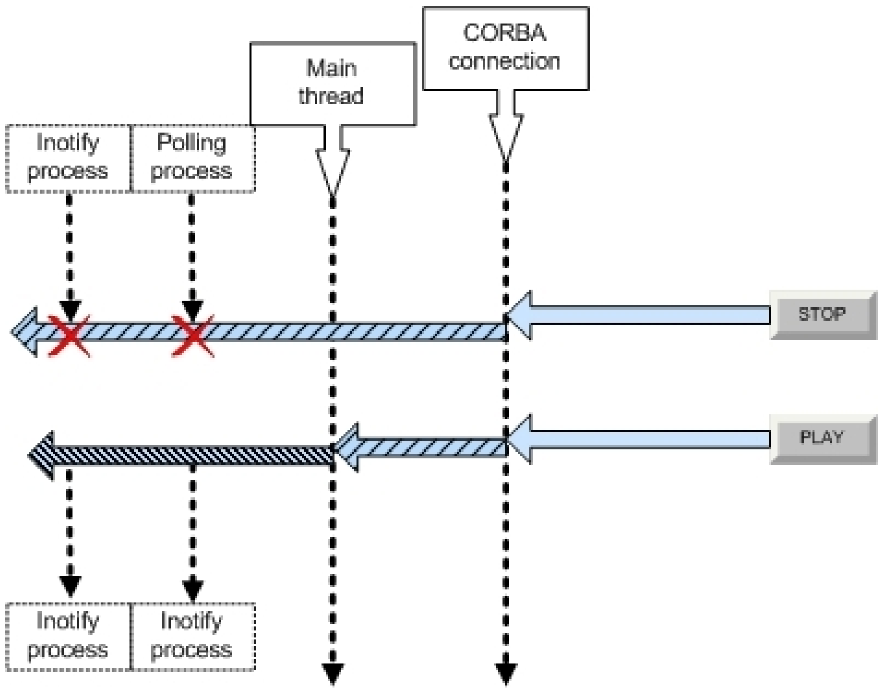


Figure 5.25: Application threads

The application uses two semaphores to resolve problems. Both semaphores are created in ConfigurationAnalysis class and are shared thanks to this Singleton class. The first semaphore allows to manage the creation of new threads. The other semaphore is shared by several application methods, and avoids concurrency problems creating and deleting threads as well as processing songs.

5.3 Database

Two databases have been designed and implemented. The first one contains data related to users. It is a very basic database with one table, that includes login, password, user name and identifier of each user. The second database is much more complex, and its implementation has involved a database design detailed in this section.

The database design [45] has been performed following an Entity-Relationship Model (ERM). ERM is an abstract and conceptual representation of data, a database modelling method, used to produce a type of conceptual schema or semantic data model of a system, in this case a relational database, and its requirements in a top-down fashion.

First, the data to be stored in the database was determined:

- Basic information about songs:
  - Song path: the path where the song is stored.
  - Song id: a different identifier for each song.
  - Song name: the name of the song in question
  - Song author: the author of the song in question.
- Advanced information about songs extracted from songs processing: the proposal parameters and the validation of users. These parameters will increase in the future, but currently there are three basic parameters:
  - Song energy rate: an integer with the value.
  - Song ending: it could be a C for the Cut and a F for Fade.
  - Song Sflatness: an array with the sflatness value.
- The parameters of the song processing, for each parameter a different number of inputs should be considered:
  - Energy rate has one input: seconds between sample.
  - Ending has two input: the last seconds to analyse and the seconds resolution.
  - Sflatness has one input: seconds between samples.
- The CORBA connection: it is composed by the machine IP where the application is running and the IIOR saved as a string.

The song path is not dealt as an only concept. The path is composed by three issues: the server, the directory and the file name. Both, the directory and the server, may change in the future. For instance, if the available disk space of one server becomes small to save more songs, the server would be changed by other with more space. However, the file system structure would be the same, and only the name of the server should be changed (one line of one table of the database). Thus, it also is necessary to store:

- The servers where songs are stored.
- The directories of these servers where songs are saved.

Once identified the data to be stored in the database, dependencies between this data were studied. The dependencies and the basic keys that should contain identifiers are:

1. Each song contains a value for each parameter (sflatness, energyrate and ending) and a new validated value by the users. Hence, each song contains an identifier.
2. Each song contains a unique path, that is composed by: a server, a directory and a file name. Several songs share a directory, a server and even, in some case, should share the same file name. Therefore, both, servers and directories contain their own identifier.
3. Each song parameter contains different analysis parameters. Thus, each song parameter has an identifier.
4. Each instantiation of the server application contains a connection data. In a machine only should be a instantiation of the server. Consequently, the IP of each machine, where a instantiation of the application is running, is an identifier.

All this leads to the deduction that the conceptual schema for the database modelling is the next Entity-Relationship diagram, shown in figure 5.27 shows each table that includes detailed specification of data elements, data types and relationships amongst them.

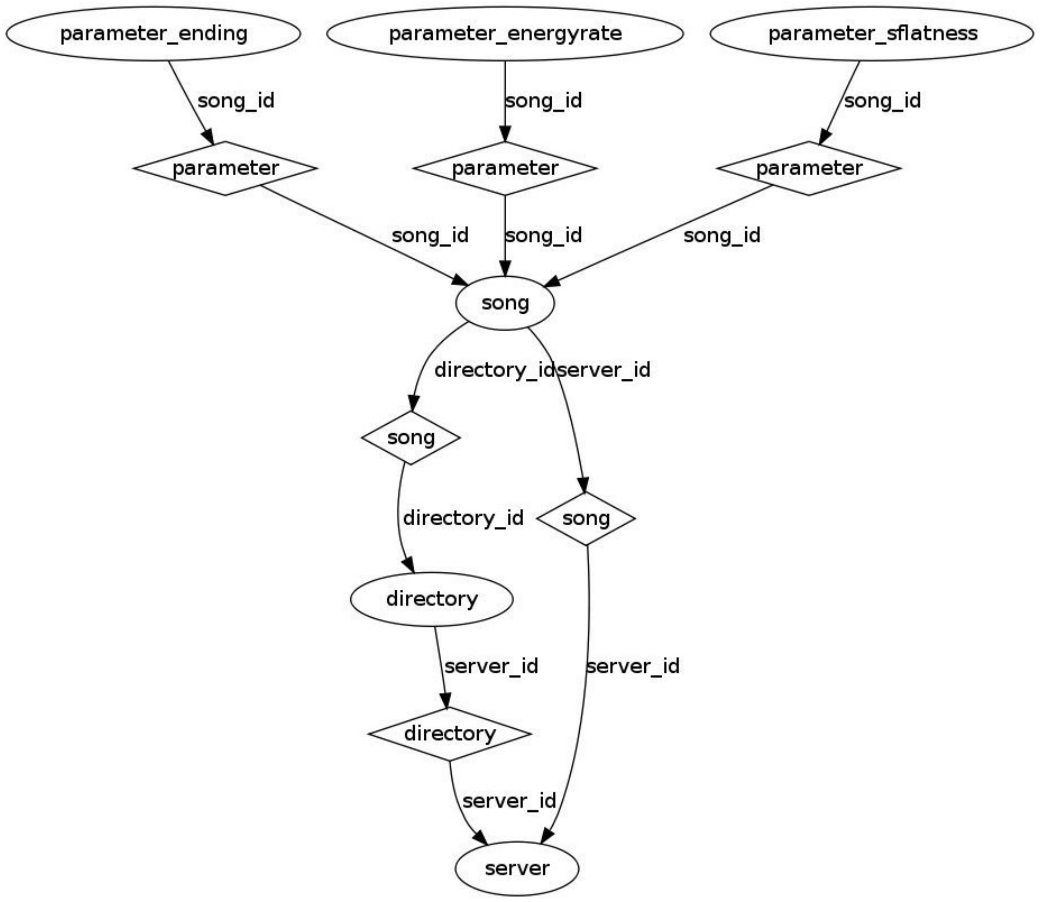


Figure 5.26: Entity-Relationship diagram

Now that the relationships and dependencies amongst the various pieces of information have been determined, it is possible to arrange the data into a logical structure which can then be mapped into the storage objects supported by the database management system. In our case, it consists of a relational databases, thus, storage objects are tables which store data in rows and columns. Each table represents an implementation of either a logical application object or a relationship joining one or more instances of one or more logical objects and contains a primary key. The relationships between tables are stored as links connecting child tables with parents. In the case of the table **song** is a complex logical relationships and it has links to more than one parent. Figure 5.27 shows each table that includes detailed specification of data elements, data types and relationships amongst them.

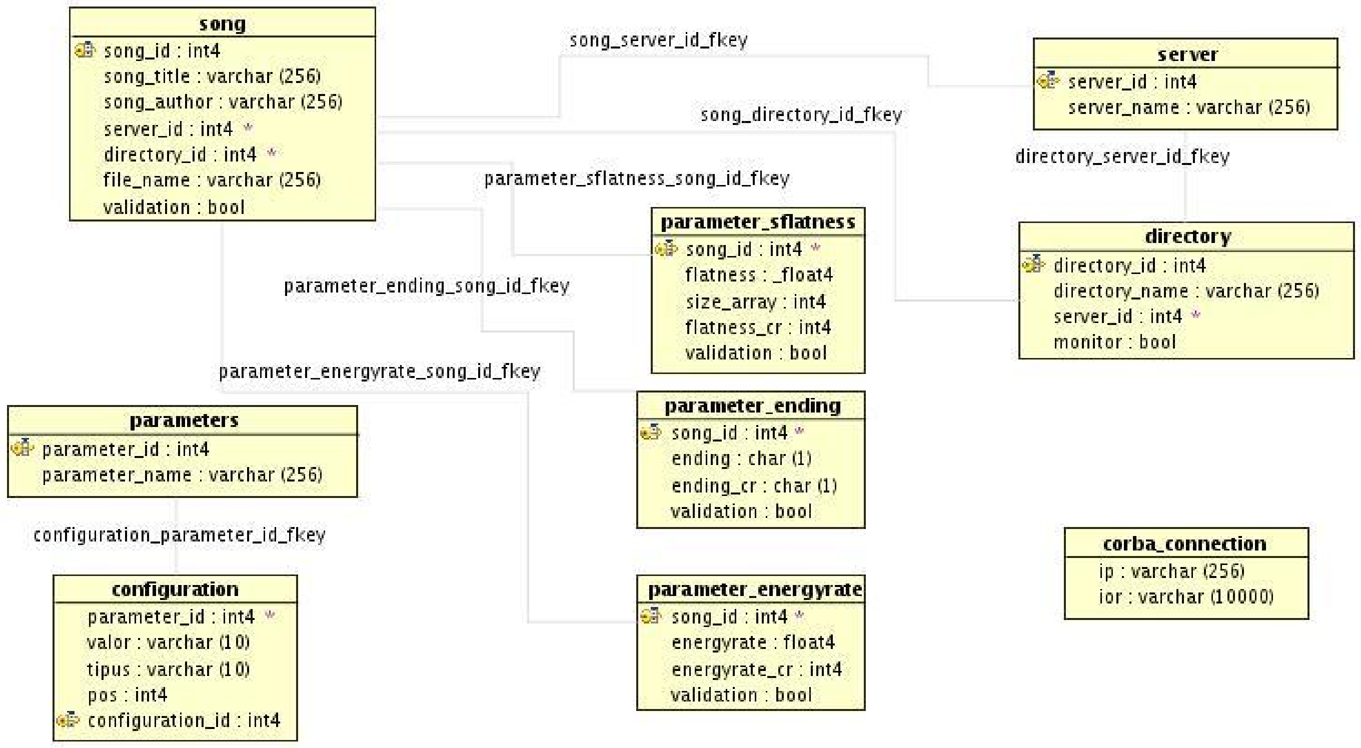


Figure 5.27: Database tables

5.4 Web Application

The Web Application provides the interface between final users, the C++ application and the database. The user Web interface is designed to be a user friendly interface and to let users perform common operations over the data in a straightforward way. The main actions implemented by the Web Application are:

- To show data generated by the C++ application. The Web Application shows if data is validated or not and the parameters extracted for each song (e.g. enery rate or ending).
- To let users validate data or change proposed values. The Web Application shows the lastest songs added to the system that have not been validated yet.
- To perform simple search: using the song title and author.
- To perform advanced search : using different song parameters.
- To listen to songs from Web Application. This action is performed using a Java applet.
- To show a chart with Sflatness rates.
- To allow performing requests to the C++ Server:
  - To change the parameters of the analysis, and reload them in the C++ application.
  - To stop the server. It will be not able to perform songs identification and, thus, to process new songs.
  - To start the server again. The server will start again songs notification.

In order to achieve these objectives a Web Application archive (WAR) has been implemented and deployed in a Server JBoss. The WAR is mainly composed by two components:

- A Web project with:
  - Java Server Pages (JSP) with CSS files, JavaScript files and images.
  - Java classes that performs the Web Application actions.
  - A web.xml file which defines the structure of the Web pplication. In this case, because of the used framework, the web.xml redirects the actions to the struts.xml file.
- A Java Applet included in a JAR file that contains the Player of the Web Application.

5.4.1 Web project

The Web Application contains several actions that perform different tasks:

- Database connection: all the actions in the Web Application require connecting to the database to retrieve or update data. Thus, the first step that should be solved is the connection with the database from the Web Application. In order to achieve this important task an EJB3.0 has been implemented and the API JDBC has been used.
- Web Application security: two main action have been implemented to provide security to the Web (details to follow).
- To Show and update data: the user interface allows to show generated data and to validate song parameters.
- To search new data: two searching types have been implemented: simple and advanced.
- To show flatness rate: it uses the google chart API [1].
- C++ Server connection: it uses CORBA middleware and allows final users to connect with the C++ application from the Web user interface.

Next, these actions are explained. In general, the designed approach makes use of the Dojo toolkit, specifically `dojo.io.bind.` The 'content' parameter is used to pass parameters to the server, without using a form tag; and JSON object is used as response data when only data is required. Figure 5.28 shows the JavaScript code that makes use of `dojo.io.bind` function, implemented to call the Validate action, that ensures that the song in question has not been validated before. The song identifier is passed thanks to the parameter **content**, and a JSON object is received as response data with a boolean that shows if this song is validated or not.

```
function validate(songid){
dojo.require("dojo.io.*");
dojo.require("dojo.event.*");
dojo.require("dojo.json");
var params = new Array();
params['songid'] = songid;
var request = new dojo.io.Request("Validate.action");
request.error = function(type, data, evt){
alert (data+"type:"+type+"event"+evt);
};
request.load = function(type, data, request) {
if(data.ok){
alert("Aquest parmetre ja ha estat validat");
}
else{
update(songid);
}
};
request.mimetype = "text/json";
request.content = params;
request.timeoutSeconds = 5;
request.timeout = function(type, data, evt){
alert("Estic cansat d'esperar");
};
dojo.io.bind(request);
}
```

Figure 5.28: JavaScript code with `dojo.io.bind` function

This is the simplest example of calling an action using this approach. However, it shows that the code is easy and more organized than parsing an XML. Moreover, it allows to perform the action in asynchronous way, faster than with the classic XML and to update only a HTML piece, not all HTML.

Database connection

In order to provide the interface between the Web Application and the database a set of classes have been implemented:

- A set of classes that allows including an Enterprise Java Beans 3.0. in the Web project, since the connection with the data base should not be performed directly from an action of the Web Application. It requires an EJB3.0.
- The ControlJDBC class, that allows to manage and control the main actions with the database.

Implementation and integration of Enterprise Java Beans 3.0 The connection to a database requires an Enterprise Java Beans 3.0. (EJB3.0). It is necessary to create a session EJB3.0. A session bean is a component created by a client for the duration of a single client/server session. A session bean performs operations for the client. Session bean objects are either stateless or stateful: maintaining conversational state across method calls and transactions. In this case, it is necessary a stateless session bean because of these reasons:

- It should only execute a single request business process in each interaction with the database.
- It does not maintain state across method invocations.
- After each method call, the container may destroy the Stateless Session Bean without any problem.
- The client should pass all client data that the bean needs as parameters to business logic methods each time it invokes a business method.
- It does not need to conserve system resource.

It is really easy to create a bean session. It should be implemented a remote or local interface, or both, and a Stateless Session Bean that contains the business logic. The Stateless Session Bean should be tagged as @Stateless. This marks the class as a stateless bean and the deployer will deploy that class as a stateless bean EJB container. In this case, only a remote interface is necessary and a session bean with the business logic.

However, the integration between EJB3.0 and Struts2 framework is not trivial. It is not possible to insert an EJB3.0 directly to an action of the Struts2 framework. In order to inject the EJB3.0 in Struts2, the Guice framework is used [4].

Guice is a, Java 5 based, dependency injection container, released by Google, that does constructor, field

and setter injection. Struts2 provides a plug-in for using Guice. In order to configure Guice a set of modules should be created. A module implements the **com.google.inject.Module** interface and contains bindings. Bindings are created with the **com.google.inject.Binder** that is passed to the **configure(Binder)** method of a module. A binding is a pair composed of a type (key) and an implementation. An implementation can be an object, a class or a provider (factory). The bindings are consulted by the Guice **Injector** when doing dependency injection. In the Guice Struts 2 plug-in the Guice **ObjectFactory** creates an **Injector** that then reads the module from the **guice.module** property in the struts.xml file and uses the bindings that are specified in that module to do dependency injection in Struts 2 actions and interceptors.
In order to enable the EJB to be injected into an action we have to create a binding for the client interface of the bean. It is not possible to create a binding to a class or instance, because the EJBs instances must be instantiated by the EJB container and not by Guice, so we use a provider that will lookup the EJB from JNDI. The Guice library includes a JNDI provider that is used, located in the **com.google.inject.jndi.JndiIntegration** class.
Figure 5.29 shows the code with the implementation of the **EJBModule**, which has only two bindings. The first binds the **javax.naming.Context** type to the **javax.naming.InitialContext** implementation. This means that when the Guice Injector finds a field or parameter of type **javax.naming.Context** , that needs to be injected (marked with the **@Inject** annotation), it will create a new object of type **javax.naming.InitialContext** and inject it. This binding is needed by the JNDI provider that is use in the next binding.

```
package audiotool.bean;
import com.google.inject.Binder;
import com.google.inject.Module;
import static com.google.inject.jndi.JndiIntegration.fromJndi;
import audiotool.bean.BBDDManagerRemote;
import javax.naming.Context;
import javax.naming.InitialContext;
public class EjbModule implements Module {
public void configure(Binder binder) {
// Bind Context to the default InitialContext.
binder.bind(Context.class).to(InitialContext.class);
// Bind the remote interface to the JNDI name using the JNDI Provider
binder.bind(BBDDManagerRemote.class).toProvider(fromJndi(BBDDManagerRemote.class,
"audiotool.bean.BBDDbean"));
}
}
```

Figure 5.29: Code EJBModule

The second binding binds the EJBs remote interface to the JNDI provider. The JNDI provider is returned from the call to the static **JndiIntegration.fromJndi (Class, String)** method. To get a JNDI provider we have to specify the expected class and JNDI name. This binding tells the Guice **Injector** that when it finds a field or parameter of type **audiotool.bean.BBDDManagerRemote** , that needs to be injected (marked with the **@Inject annotation)**, it should call the providers **get()** method to retrieve an instance to inject.

Figure 5.30 shows the way to instantiate an EJB from an action.

```
@Inject BBDDbean bean;
```

Figure 5.30: Inject EJB3.0

Implementation of the class ControlJDBC The class ControlJDBC allows to manage the connection of the database. This class opens and closes the connection, and allows to connect to both databases (user database and song data database). It is used by the EJB3.0 when it needs some interaction with the database. This class closes the connection, but not release all the resources used when an interaction with the database is performed. Methods, that interact with the database, have to release the resource. Figure 5.31 shows the interactions with this class.

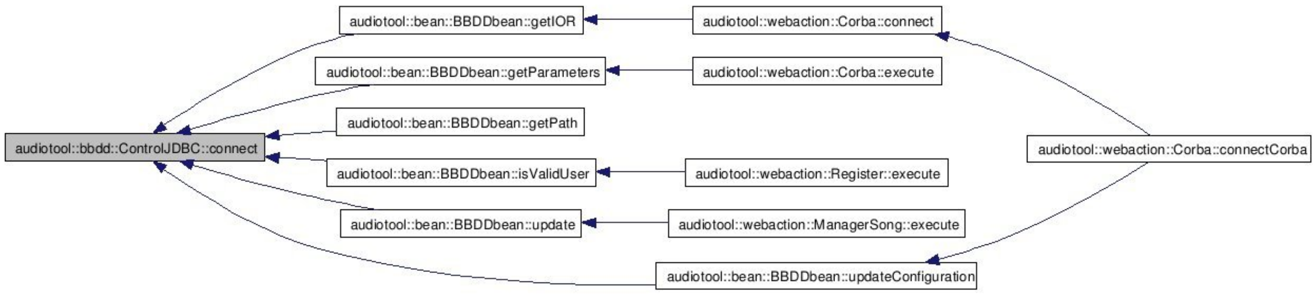

audiotool::bbdd::ControlJDBC::connect
audiotool::bean::BBDDbean::getIOR
audiotool::bean::BBDDbean::getParameters
audiotool::bean::BBDDbean::getPath
audiotool::bean::BBDDbean::isValidUser
audiotool::bean::BBDDbean::update
audiotool::bean::BBDDbean::updateConfiguration
audiotool::webaction::Corba::connect
audiotool::webaction::Corba::execute
audiotool::webaction::Register::execute
audiotool::webaction::ManagerSong::execute
audiotool::webaction::Corba::connectCorba

Figure 5.31: Class ControlJDBC

Web Application Security

In order to provide security to the Web Application two actions have been implemented:

- Register: it ensures that users contain login and password at the beginning of the application. Figure 5.34 shows the method that this action calls to perform this task, and figure 5.33 shows the graphical interface with the register.
- Authentication: it ensures that the user is validated before acceding to an action. It is implemented as an Interceptor. An Interceptor in Struts2 is a method that is executed before and after the execution of each action. It can modify the action flow. In each interaction this interceptor accesses to the session variables and it makes sure that the user has been registered. Figure 5.33 shows this action and its base class.

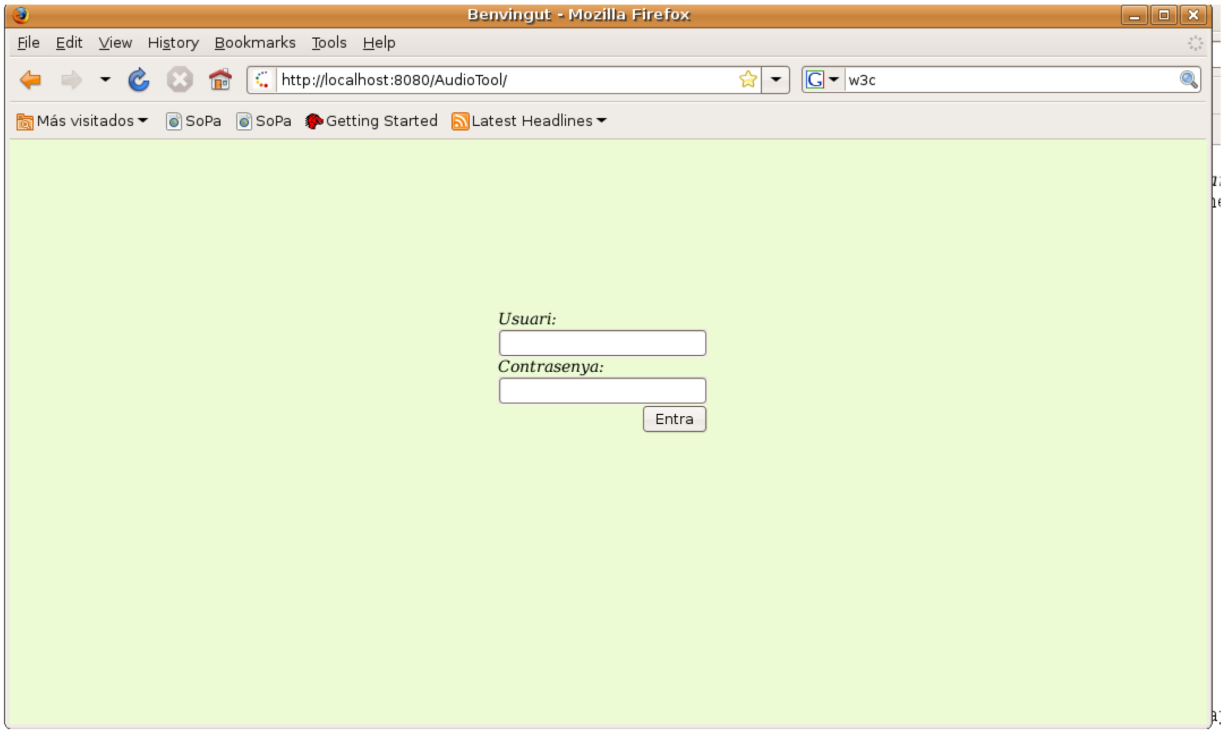

Benvingut - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/AudioTool/
w3c
Más visitados SoPa SoPa Getting Started Latest Headlines
Usuari:
Contrasenya:
Entra

Figure 5.32: Register Web

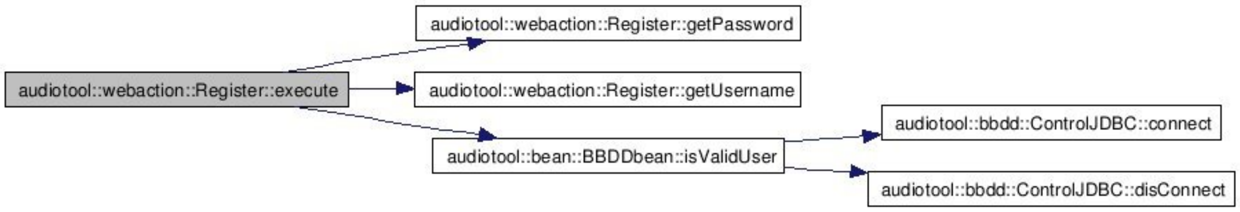

audiotool::webaction::Register::execute
audiotool::webaction::Register::getPassword
audiotool::webaction::Register::getUsername
audiotool::bean::BBDDbean::isValidUser
audiotool::bbdd::ControlJDBC::connect
audiotool::bbdd::ControlJDBC::disConnect

Figure 5.33: Register action

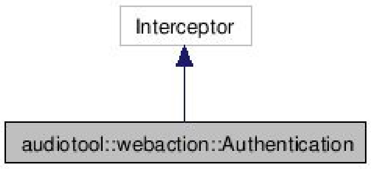


Figure 5.34: Authentication action

Showing and updating data

The main purposes of Web Application are to show the data produced by the audio processing server and to let final users perform common operation over this data.
Two actions have been designed to perform these tasks: one to retrieve data from the database and the second one to update the data. It has also implemented a class to store songs data, called ParameterSong. It is used to pass all needed parameters of each song amongst the different classes. It contains setter and getter methods for each parameter. The figure 5.35 shows the method populateParameterSong, that populates this class when retrieves data from the database.

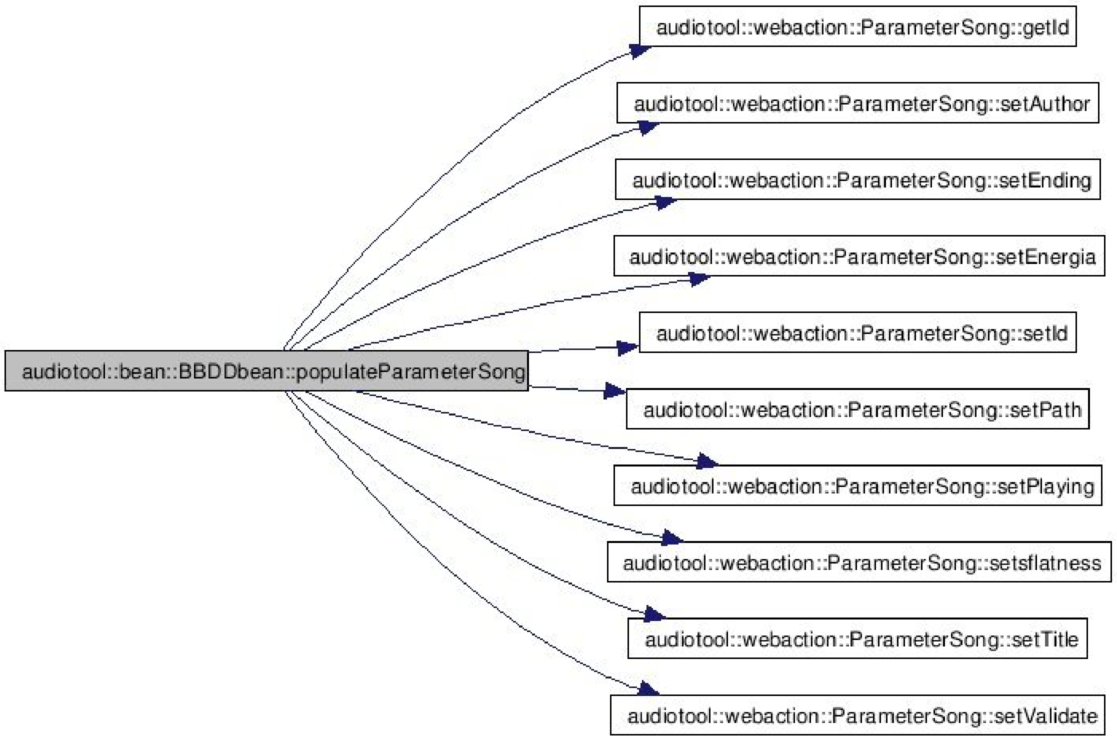

audiotool::bean::BBDDbean::populateParameterSong
audiotool::webaction::ParameterSong::getId
audiotool::webaction::ParameterSong::setAuthor
audiotool::webaction::ParameterSong::setEnding
audiotool::webaction::ParameterSong::setEnergia
audiotool::webaction::ParameterSong::setId
audiotool::webaction::ParameterSong::setPath
audiotool::webaction::ParameterSong::setPlaying
audiotool::webaction::ParameterSong::setsflatness
audiotool::webaction::ParameterSong::setTitle
audiotool::webaction::ParameterSong::setValidate

Figure 5.35: Populate song

Retrieving Data The first application action, after user validation, gets the five last songs of the database without validation (this number is configurable) and shows this data in the Web Application. Data is saved in a variable session and when a song is validated and updated the parameter of the song in question is changed. It is also possible to reload the showed data with new songs without validation if all the data has been validated or if the user decides to change the songs. Figure 5.36 shows the action that performs the task of retrieving songs data.

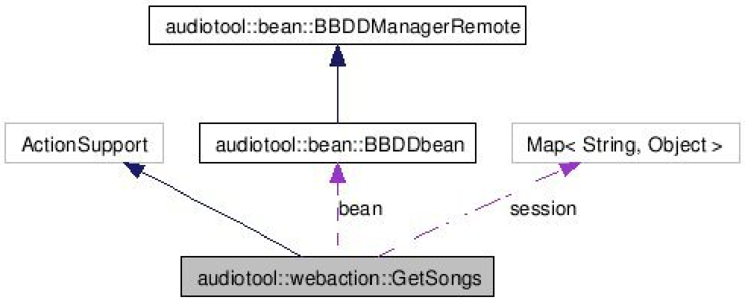
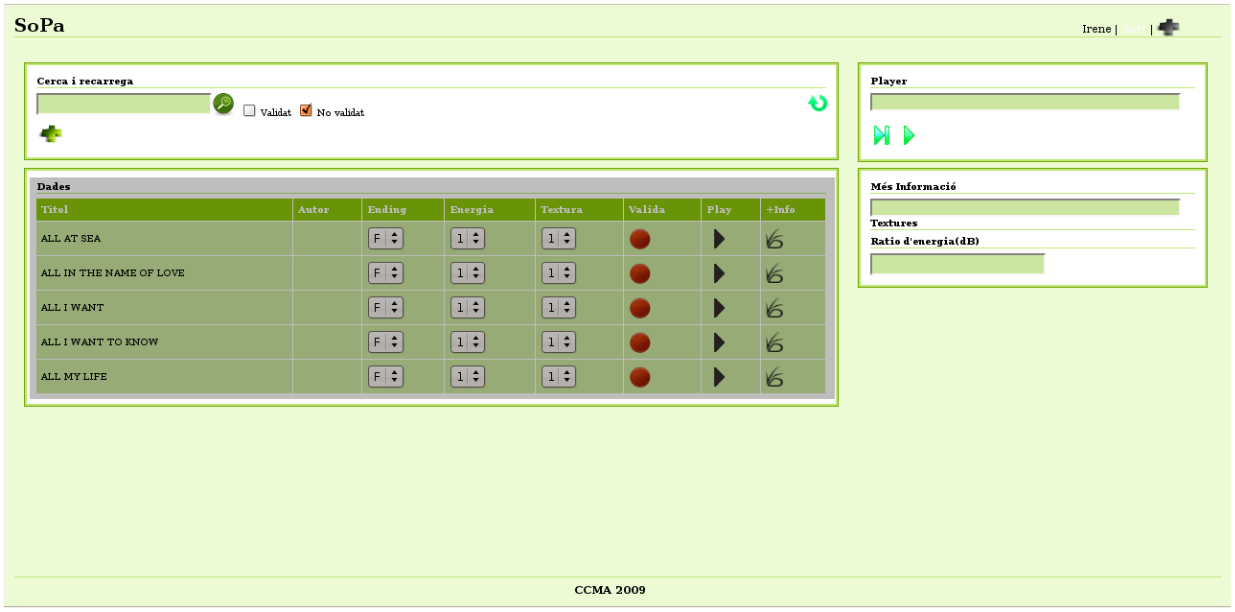

audiotool::bean::BBDDManagerRemote
ActionSupport
audiotool::bean::BBDDbean
Map< String, Object >
bean
session
audiotool::webaction::GetSongs

Figure 5.36: Retrieving songs data

Figure 5.37 shows the Web interface, presented after users validation.


SoPa

Irene |

Cerca i recarrega

Validat [✓ No validat]

Player

Dades

Titol	Autor	Ending	Energia	Textura	Valida	Play	+Info
ALL AT SEA		F	1	1	●	▶	
ALL IN THE NAME OF LOVE		F	1	1	●	▶	
ALL I WANT		F	1	1	●	▶	
ALL I WANT TO KNOW		F	1	1	●	▶	
ALL MY LIFE		F	1	1	●	▶	

Més Informació

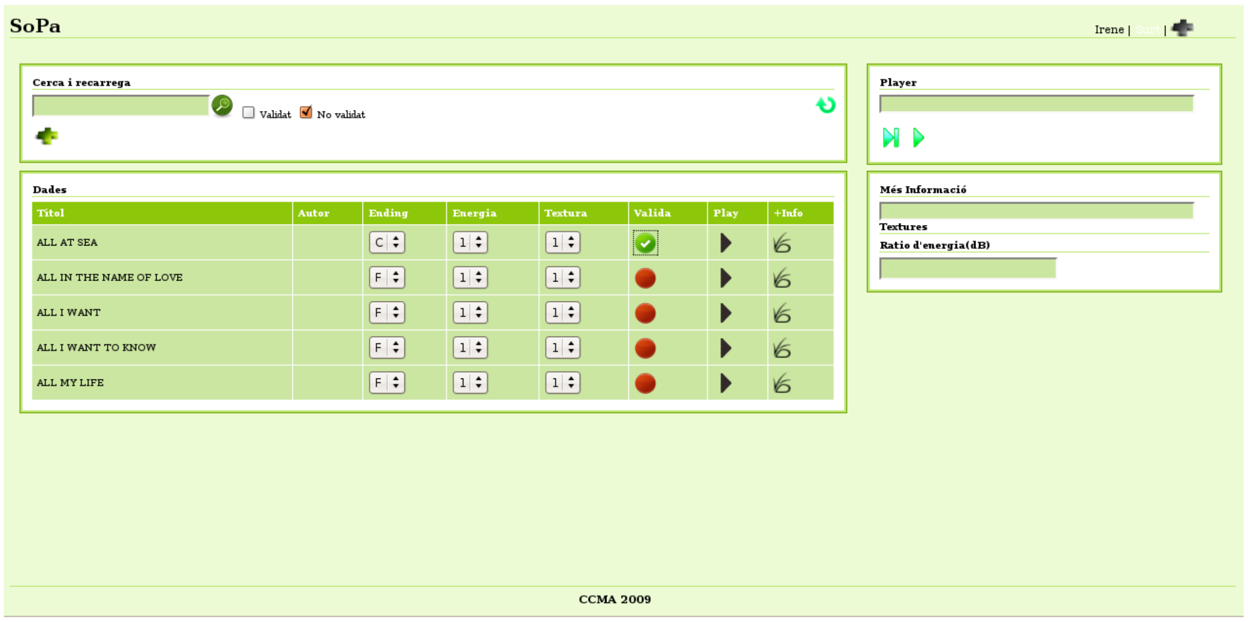
Textures

Ratio d'energia(dB)

CCMA 2009

Figure 5.37: Data Web

Updating Data Data songs is shown inside a table that contains drop-down lists for each parameter to be validated. The user can change parameters with these drop-down lists and eventually can validate all parameters with a round red button placed in the same row. This button calls a JavaScript function that uses `dojo.io.bind`. Using this approach, it is easy to send the data entered by users to the server "behind the scenes"; the server replies and a JavaScript function updates the presentation of the page. Figure 5.38 shows the new button after validating.


SoPa

Irene |

Cerca i recarrega

☐ Validat ☑ No validat

Player

Dades

Titol	Autor	Ending	Energia	Textura	Valida	Play	+Info
ALL AT SEA		C	1	1	✓	▶	
ALL IN THE NAME OF LOVE		F	1	1	●	▶	
ALL I WANT		F	1	1	●	▶	
ALL I WANT TO KNOW		F	1	1	●	▶	
ALL MY LIFE		F	1	1	●	▶	

Més Informació

Textures

Ratio d'energia(dB)

CCMA 2009

Figure 5.38: Web after validation

New song parameters are passed to the server when the action is called. The JavaScript function receives a JSON objects that indicates if the validation has been performed without error. If no error has occurred, the Javascript function updates the button to show that this parameter has been validated. If an error has occurred during the process, the function shows a pop-up with an error message.

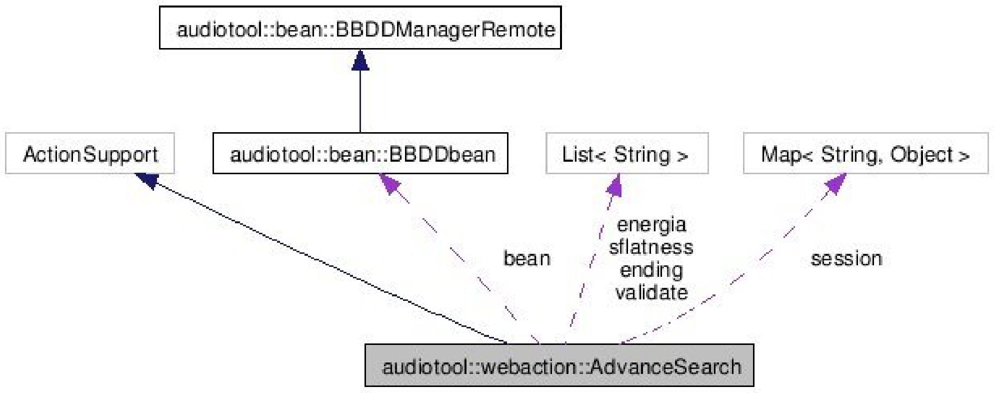
The design of the Web Application has been performed to avoid unnecessary interactions with the database. For instance, before updating a parameter in the database, it is checked that the parameter in question has been changed and it is different to the proposed parameter, and thus it is inevitable updating the database. However, in order to avoid concurrency problems with other possible users of the Web Application, before calling the action of updating the data, it is confirmed that the parameter has not been updated before. If the song has been validated a pop-up is showed to warn about it to the user.

Searching data

The Web Application contains two types of searches:

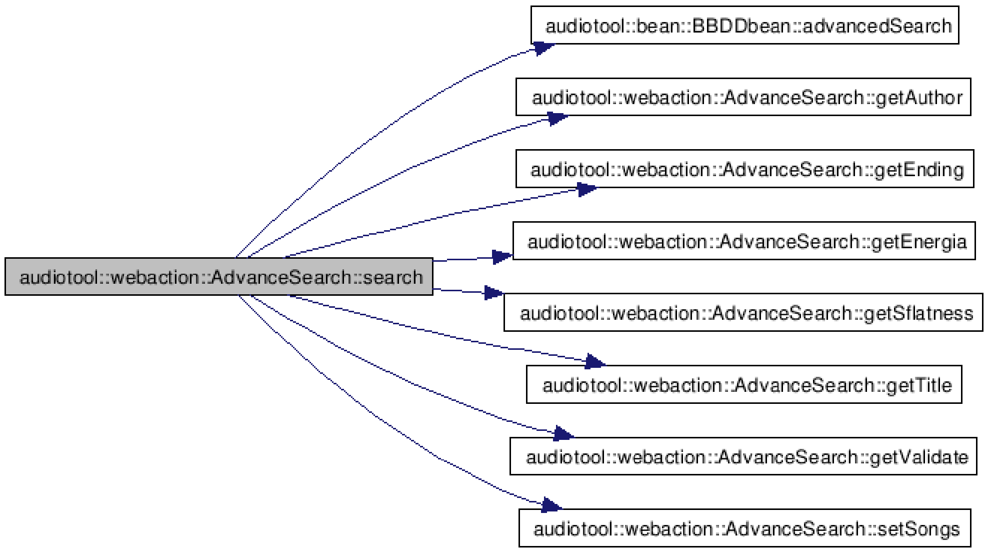
- The simple search: only performs the search over the field title and author. It allows users to select if the songs should be validated or not.
- The advanced search: users should perform the query over all the different parameters of the songs.

Both actions are called from JavaScript, and only the table where the data songs is showed is updated. Figure 5.39 shows the class that performs the advanced search, and figure 5.40 shows method that this action calls to retrieve the parameters from the Web.



audiotool::bean::BBDDManagerRemote
ActionSupport
audiotool::bean::BBDDbean
List< String >
Map< String, Object >
bean
energia
sflatness
ending
validate
session
audiotool::webaction::AdvanceSearch

Figure 5.39: Advanced search class



audiotool::webaction::AdvanceSearch::search
audiotool::bean::BBDDbean::advancedSearch
audiotool::webaction::AdvanceSearch::getAuthor
audiotool::webaction::AdvanceSearch::getEnding
audiotool::webaction::AdvanceSearch::getEnergia
audiotool::webaction::AdvanceSearch::getSflatness
audiotool::webaction::AdvanceSearch::getTitle
audiotool::webaction::AdvanceSearch::getValidate
audiotool::webaction::AdvanceSearch::setSongs

Figure 5.40: Methods called by the advanced search action

Figure 5.41 shows the methods that the simple search action shows.

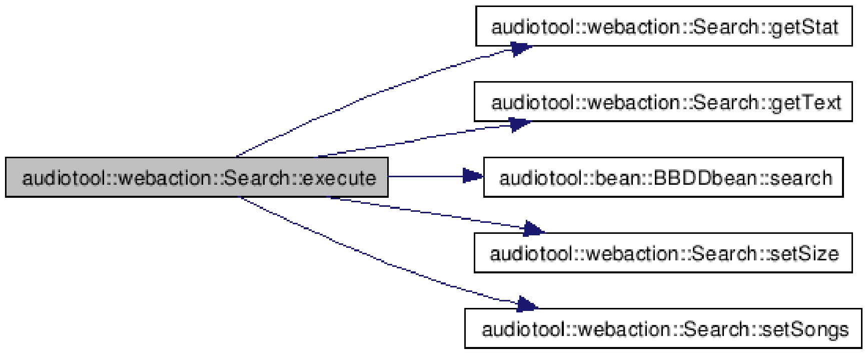
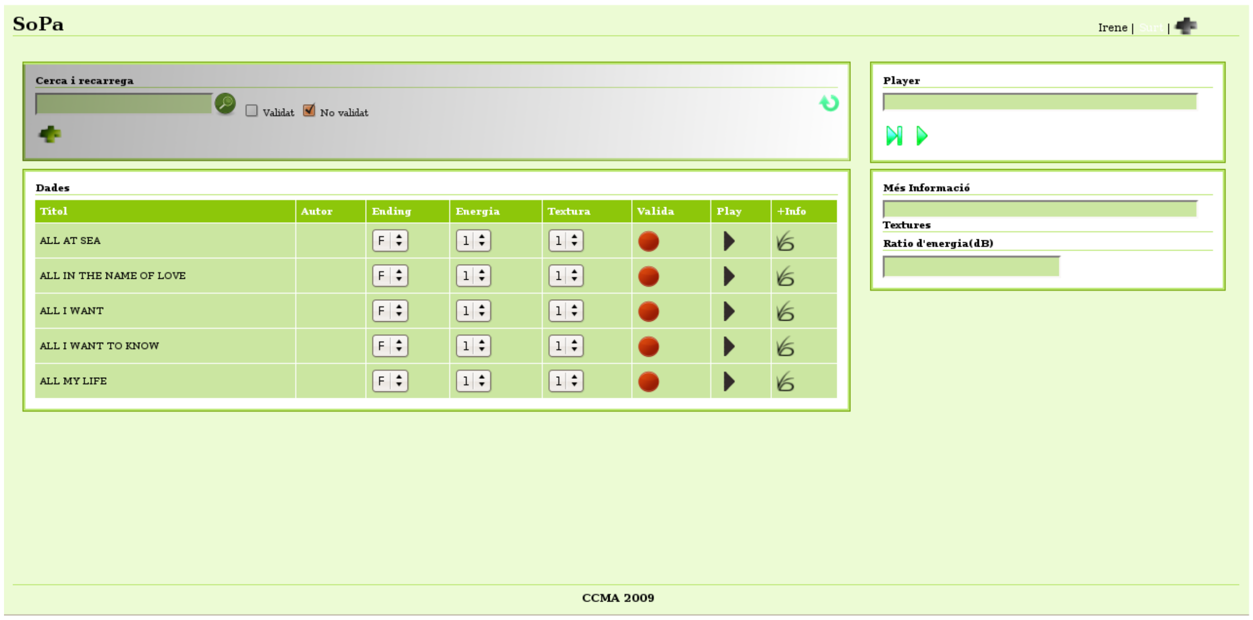

audiotool::webaction::Search::execute
audiotool::webaction::Search::getStat
audiotool::webaction::Search::getText
audiotool::bean::BBDDbean::search
audiotool::webaction::Search::setSize
audiotool::webaction::Search::setSongs

Figure 5.41: Methods called by the simple search action

Figures 5.42 and 5.43 show both search types in the Web interface.



SoPa

Irene |

Cerca i recarrega

☐ Validat ☑ No validat

Player

Dades

Titol	Autor	Ending	Energia	Textura	Valida	Play	+Info
ALL AT SEA		F	1	1	●	▶	
ALL IN THE NAME OF LOVE		F	1	1	●	▶	
ALL I WANT		F	1	1	●	▶	
ALL I WANT TO KNOW		F	1	1	●	▶	
ALL MY LIFE		F	1	1	●	▶	

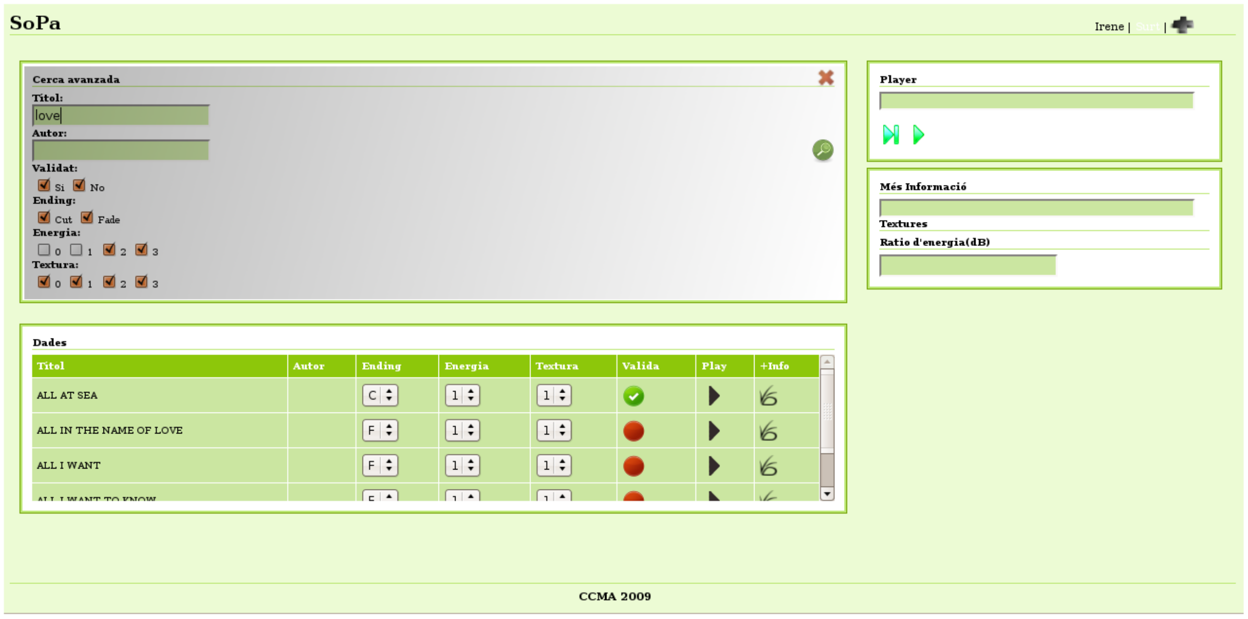
Més Informació

Textures

Ratio d'energia(dB)

CCMA 2009

Figure 5.42: Simple search in the Web


SoPa

Irene |

Cerca avanzada

Titol: love

Autor:

Validat: [✓ Si] [✓ No]

Ending: [✓ Cut] [✓ Fade]

Energia: ☐ 0 ☐ 1 [✓ 2] [✓ 3]

Textura: [✓ 0] [✓ 1] [✓ 2] [✓ 3]

Player

Més Informació

Textures

Ratio d'energia(dB)

Dades

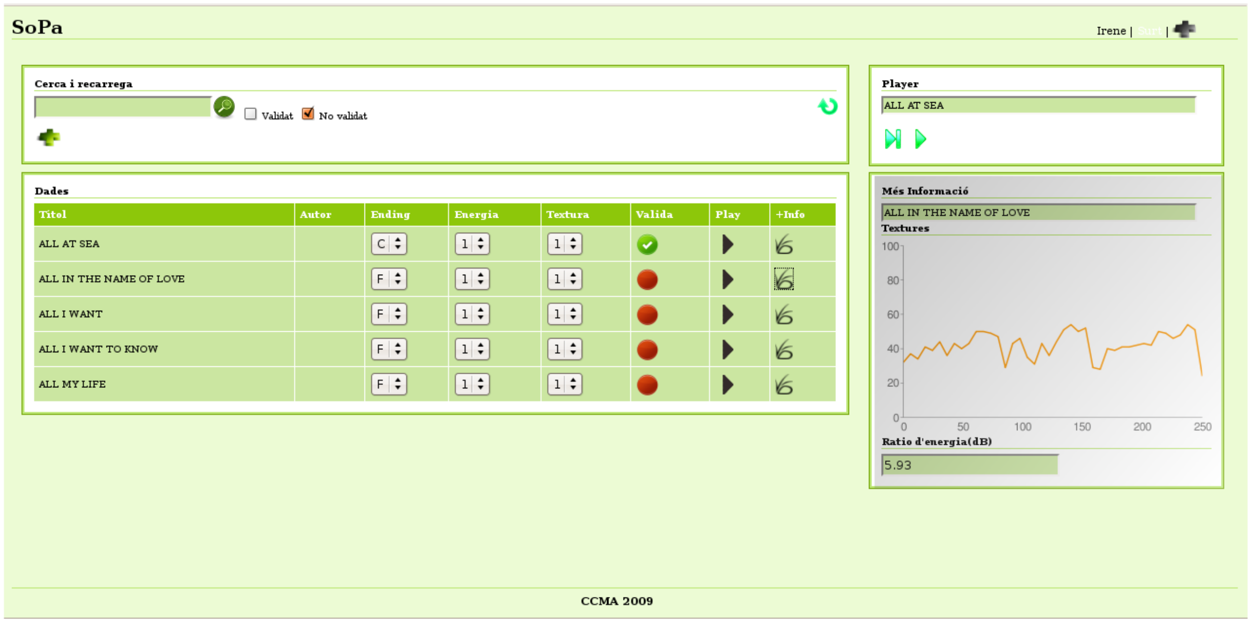
Titol	Autor	Ending	Energia	Textura	Valida	Play	+Info
ALL AT SEA		C	1	1	✓	▶	
ALL IN THE NAME OF LOVE		F	1	1	●	▶	
ALL I WANT		F	1	1	●	▶	
ALL I WANT TO KNOW		F	1	1	●	▶	

CCMA 2009

Figure 5.43: Advanced search in the Web

Flatness charts

The Web also shows the flatness rate of each song. The last button of each row allows showing this chart. The Google Chart API is used to perform this action. This API allows dynamically generating charts, using an URL with the data. Figure 5.44 shows the Web with the sflatness chart of a song.


SoPa

Irene |

Cerca i recarrega

Validat [✓ No validat]

Player

ALL AT SEA

Dades

Titol	Autor	Ending	Energia	Textura	Valida	Play	+Info
ALL AT SEA		C	1	1	✓	▶	
ALL IN THE NAME OF LOVE		F	1	1	●	▶	
ALL I WANT		F	1	1	●	▶	
ALL I WANT TO KNOW		F	1	1	●	▶	
ALL MY LIFE		F	1	1	●	▶	

Més Informació

ALL IN THE NAME OF LOVE

Textures

Ratio d'energia(dB)

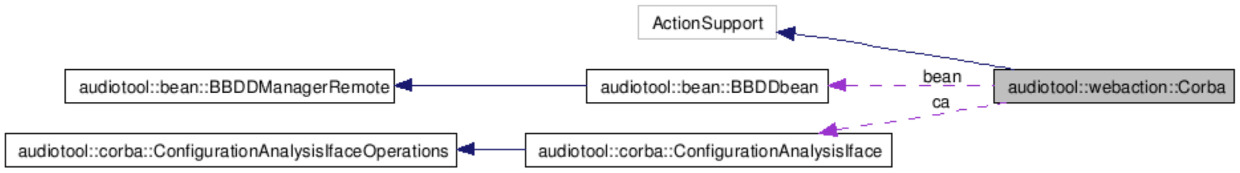
5.93

CCMA 2009

Figure 5.44: Flatness chart presented in the Web

CORBA connection

The connection with the C++ server is performed using the middleware CORBA. Figure 5.45 shows the action that performs the CORBA connection and the different tasks. This actions makes use of Java classes generated automatically by the IDL compiler of JavaSDK, called `idl2java`. These classes are saved in the package audiotool.corba.



ActionSupport
audiotool::bean::BBDDManagerRemote
audiotool::bean::BBDDbean
bean
ca
audiotool::webaction::Corba
audiotool::corba::ConfigurationAnalysisIfaceOperations
audiotool::corba::ConfigurationAnalysisIface

Figure 5.45: CORBA action

This connection has three functionalities:

1. To reload the parameters of the song analysis.

2. To stop the process that notifies new songs in directories.

3. To start again the process that notifies new songs in directories.

Figure 5.46 shows the methods called to perform previous functionalities.

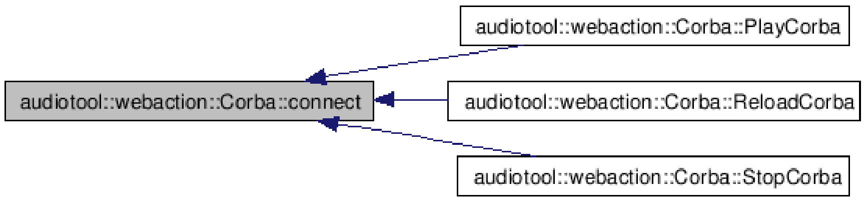
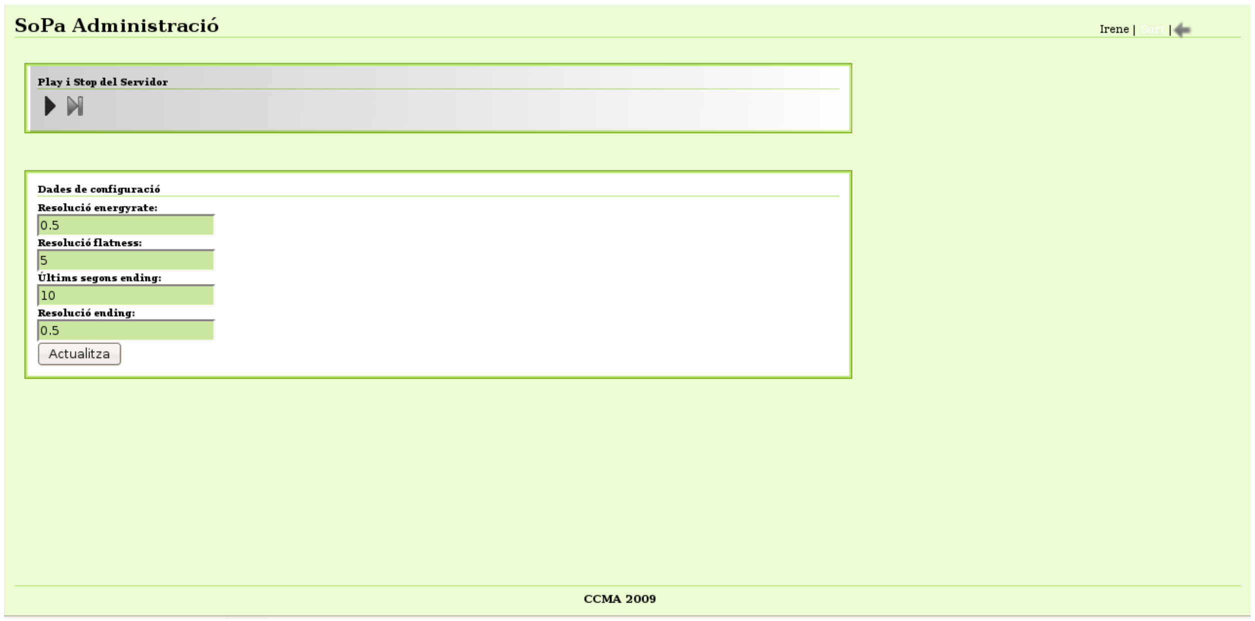

audiotool::webaction::Corba::PlayCorba
audiotool::webaction::Corba::connect
audiotool::webaction::Corba::ReloadCorba
audiotool::webaction::Corba::StopCorba

Figure 5.46: Methods called by CORBA action

Figures 5.47 and 5.48 shows the interaction from the Web with the C++ Server.



Dades de configuració

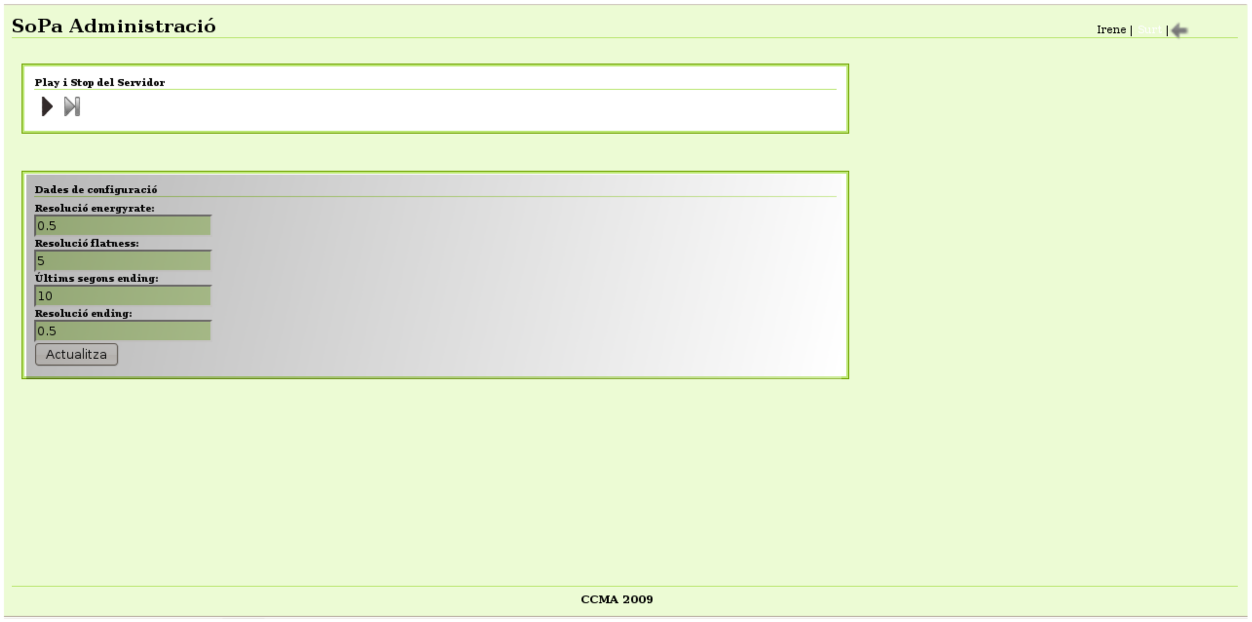
Resolució energyrate: 0.5

Resolució flatness: 5

Últims segons ending: 10

Resolució ending: 0.5

Figure 5.47: Play and stop the server from the Web


SoPa Administració

Irene |

Play i Stop del Servidor

Dades de configuració

Resolució energyrate: 0.5

Resolució flatness: 5

Últims segons ending: 10

Resolució ending: 0.5

Actualitza

CCMA 2009

Figure 5.48: Reload the analysis parameters from the Web

The connection data (IOR) is read from the database, and then the request to the C++ is performed. When the C++ replies the request, the Web application shows the result. The figure 5.49 shows how the Web Application gets the connection data.

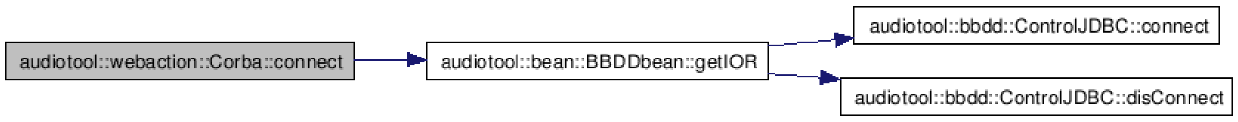

$$\begin{tikzcd}
 & & \text{audiotool::bbdd::ControlJDBC::connect} \\
\text{audiotool::webaction::Corba::connect} \arrow[r] & \text{audiotool::bean::BBDDbean::getIOR} \arrow[ur] \arrow[dr] & \\
 & & \text{audiotool::bbdd::ControlJDBC::disConnect}
\end{tikzcd}$$

Figure 5.49: CORBA connection

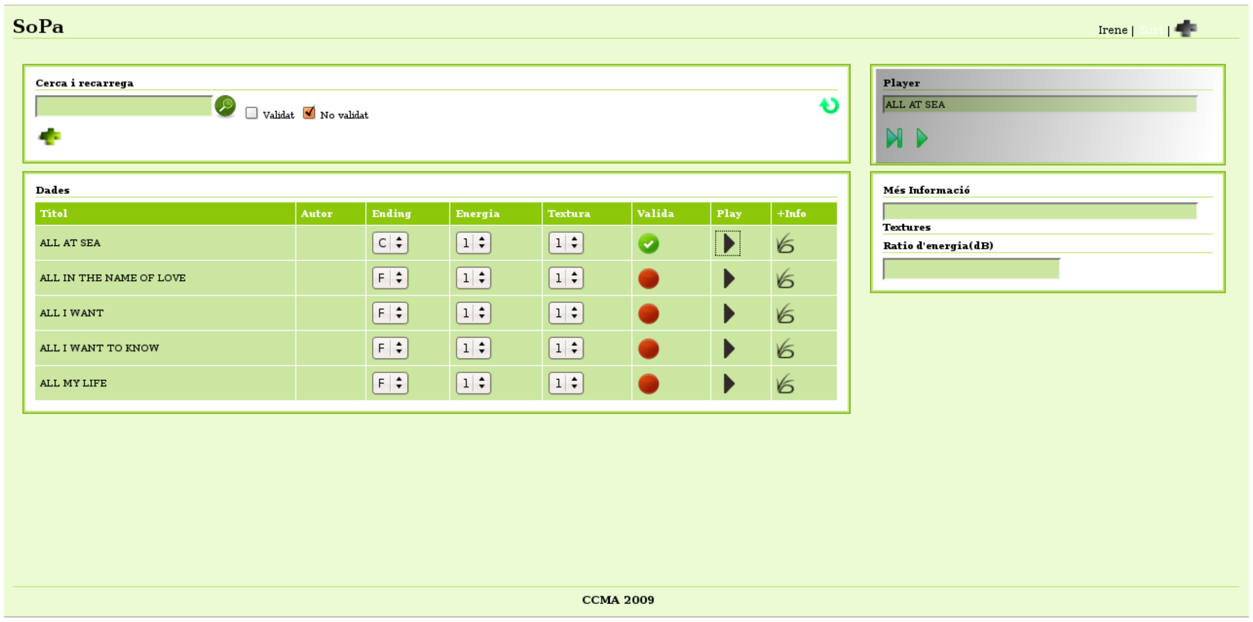
5.4.2 Applet player

In order to include a music player in the HTML page of the Web application a Java applet has been implemented. This applets makes use of the BasicPlayer API of the Javazoom project. This API provides a player that is able to play, stop, pause, resume and seek audio file or stream.
Songs are opened by the applet through an URL with a dynamic JSP that uploads the song. A JavaScript function manages the applet player. First of all, the function compares the retrieved song with the latest song opened by the applet. If the song is the same than the latest, the applet does not do anything. If songs are different, the function performs an action request to encode the retrieved song to the format WAV. If

no error occurs in the encoding process, the function calls the play method of the applet and starts playing the new song.

Some problems related to the Applet security and signed applet have appeared and the implementation of the applet has implied a delay. JARs used to implement the applet are signed, and when we deploy the applet in the browser an error occurs. It is not possible to have two different signs in the same applet. Thus, signed JARs were decompressed and included in a new JAR with the player code.

Figure 5.50 shows the player of the Web. Each row contains a button to play the song in question. The Web also contains a chart, where the name of the playing song is showed and two buttons allow pausing and playing songs.


SoPa

Irene |

Cerca i recarrega

☐ Validat ☑ No validat

Player

ALL AT SEA

Dades

Titol	Autor	Ending	Energia	Textura	Valida	Play	+Info
ALL AT SEA		C	1	1	✓	▶	
ALL IN THE NAME OF LOVE		F	1	1	●	▶	
ALL I WANT		F	1	1	●	▶	
ALL I WANT TO KNOW		F	1	1	●	▶	
ALL MY LIFE		F	1	1	●	▶	

Més Informació

Textures

Ratio d'energia(dB)

CCMA 2009

Figure 5.50: Player of the Web

Chapter 6

System test

In order to test the sturdiness, the stability and the rapidness of the implemented system two tests have been performed:

- A test bench to verify the correctness of the design and the implementation.
- A processes duration test to verify the system rapidness

6.1 Test bench

6.1.1 Resource released in C++ server [TEST1]

A song set, composed of 1000 song, was analyzed. This test had shown the next application problems that were solved:

- The application contained memory leaks problems and, thus, it failed. The Valgrind [33] tool was used to identify these problems and the program memory in question was released.
- Long songs (more than ten minutes) need longer vectors for saving the analysis data. The vector size was changed in order to save the generated data for longer songs.

6.1.2 Interoperability [TEST2]

The system has been tested connected to Windows servers and to GNU/Linux servers in order to ensure that it works fine in both run-time environment.

6.1.3 Concurrency in C++ server [TEST3]

Simultaneously, several songs have been added to different server with different run-time environment. The system has proved to be stable. It does not have concurrency problems between the two different threads that perform the event notification.

6.1.4 Integration with the Web application [TEST4]

The server has been stopped and played from the Web Application continuously during 30 seconds. The server has neither concurrency problems nor memory leaks, and it works fine after this assessment.

6.1.5 Stability in time [TEST5]

The server application has been running during three days. This application scans the directory files each minute and the directory contains more than 1000 songs. The server has proved to be stable.

6.1.6 Stability with different files [TEST6]

Corrupted songs and different file types have been added to different servers. The C++ application identifies them. If the file is not a song, the application does not start the analysis. Otherwise, if the file is a song (or at least, contains a song file name extension) the song analysis starts. The program uses try/catch blocks to catch Exceptions, and when the application fails because of the corrupted songs, this exceptions are handled. Figure 6.1 shows a code piece where two Exceptions are catched.

```
codec = avcodec_find_encoder(c->codec_id);
    if (!codec) {
     throw Exception(-1);
        //FATAL("codec not found\n");
    }

    /* open it */
    if (avcodec_open(c, codec) < 0) {
     throw Exception(-2);
        //FATAL("could not open codec\n");
    }
```

Figure 6.1: Throws inside a try/catch block to catch Exceptions

6.1.7 Resource release in Web Application [TEST7]

Different song searches have been performed from the Web application. The database contains more than 1000 songs and the retrieved song are more than 200 songs. Interactions with the database are a lot and it showed memory release problems with the database connection from the Java API JDBC. In order to solve this, the option finally was added to the try/catch block. This is a block of code which should be free of exceptions in all circumstances as it will be run. If the try block completes without an Exception, the finally block is executed. If there is an Exception, once the catch block completes, the finally block is also executed. Thus, all JDBC resources are always released. Figure 6.2 shows a simple example of usage of the try/catch block with the finally option.

```
public boolean isValidUser (String name, String password) throws SQLException {


ControlJDBC c = new ControlJDBC();
Connection con = c.connect("Users");


try{
String query = "SELECT * FROM users WHERE name = "+"'"+name+"'";
Statement s = con.createStatement();
try{
ResultSet resultSet = s.executeQuery(query);
try{
if (resultSet.next() == false) return false;
if (password.equals(resultSet.getString(3))) return true;
} finally {
resultSet.close();
}
} finally {
s.close();
}
} finally {
c.disConnect();
}
return false;


}
```

Figure 6.2: The try/catch block with the finally option

6.1.8 Concurrency in the Web application [TEST8]

Two users have tried to validate the same song at the same time. The system updates the validation that arrives first to the server. The second user is notified by the Web Application that the song in question has been validated before by other user.

6.2 Processes duration test [TEST9]

The processes duration of different system modules have been tested in order to assess the system rapidness. Next, the features of the test environment are explained:

- The server features where the C++ application is running are:
  - Operating system: Ubuntu Release 9.04 (jaunty) with Kernel Linux 2.6.28.16 GNOME 2.26.1
  - Memory: 1000.7MB
  - Processor: Intel(R)Xeon(R)CPU E5405 @2.00 GHZ Quad-Core
- The server features where the songs are stored are:
  - Operating system: Windows Server 2003
  - Memory: 2000.0MB
  - Processor: Intel(R)Xeon(R)CPU E5140 @ 2.33GHZ Dual-Core
- Both servers are connected to a 100 MB local network.
- The database is installed in the same server than the C++ application. Thus, the connection to the database is always local.
- The Web Application Server is also installed in the same server than the database. Thus, this connections is also always local.

The required time to perform the different system processes has been tested. Next, obtained results are presented.

6.2.1 Time spent processing songs

The time spent to process a song includes: the encoding time, the processing time, the needed time to save the data to the database and the required time to delete the encoded song from the server. These times have been measured in two ways: processing a song set stored in the same server where the application is and other set stored in a server connected to the local network. The only time that could be different is the encoding time, since it is the only step performed through the network. Obtained times are pretty much the same in both environments, thus, it has no sense to show all the measures. Results shows that the time spent processing a song is directly proportional to its size or length. Figure 6.3 shows this proportionality measured to 25 songs:

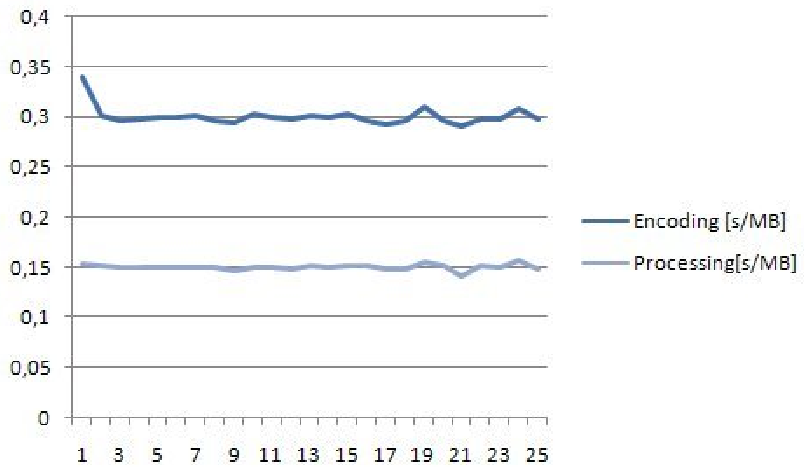


Figure 6.3: Seconds per MB processing songs

The time unit of these measures is microseconds. Thus, measures equal to zero implies that these values are less than 1 microsecond. Both measures, the time spent saving in the database and deleting the song, are less than one microsecond. Both processes are performed in local mode. If the database would be in other machine connected to the local network these times would vary. This variation would depend on two main factors: the network features and the new machine features.
In conclusion, the average seconds per MB that are needed to process a song are: 0.276 seconds per MB for encoding a song, 0.151 seconds per MB for processing a song and less than 1 microsecond/MB for saving the data in the database and removing the song. Currently, audio processing is extracting only three parameters. However, more parameters will be extracted in future library release and, consequently, the time spent processing a song will increase.

Since the songs average length is of 4 minutes and the size about 7.5 MB, the average time spent processing a song is 3.22 seconds. It is a very good result compared with times that we obtained with the Windows version of the library. Figure 6.4 shows the time spent in each action of the song processing:

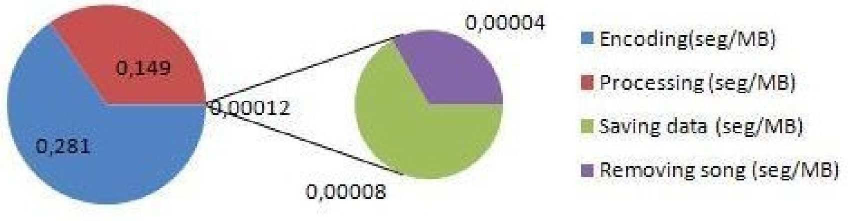


Figure 6.4: Time spent in each action

6.2.2 Time spent starting the server

Starting the server implies to create two threads and two instantiations of both classes: DiskInotify and DiskPoll. It also implies to load these instantiations in the class ConfigurationAnalysis. It will let this class delete them after. If the second thread is not created this time would be lower.

This time has been calculated 12 times and results have fluctuated between 20 milliseconds and 40 milliseconds, with an average of 30 milliseconds (0.03 seconds). This measure calculates the time that the C++ application takes to performs this task. However, this time is increased if we think in the time spent from a user performs the request to play the server until this user receives the reply. This time depends on more factor impossible to measure (such as the network features), however the obtained time (0.03 seconds) ensures that this task will be performed rapidly.

6.2.3 Time spent stopping the server

The time required to stop the server is much more lower than to play it. This time is composed by: the resources release of both instantiations (both Disk classes) and the cancellation of both threads. This time is always less than 1 millisecond. Although, like in the previous case, this time is not the required time to perform the task completely, it ensures that this task would be very fast.

6.2.4 Time spent rescanning directories

This task is only performed by the class DiskPoll, that is not always created (only when a Windows server is connected to the application). However, the required time to rescan directories has been calculated to show what this task implies in the application. This time involves to list all directory files and to compare each file with the latest rescanning, if a new song is detected it will be processed. However, the calculated time does not include any processing time, only rescanning and comparing time. This time has a clear dependency with the number of files that directories contain. Two measures have been taken: one in a directory with 1600 songs and other with 100 songs. The first measured time has an average of 10 millisecond (0.01 seconds), and the second one less than 1 millisecond. The required time to perform this task is small and not implies to block the application during a long time. However, the CPU that the application needs when this class scans the directories is much more than using Inotify process.

6.3 Test and requirements compilation

Finally, table 6.1 shows a tests compilation. All of these tests have been performed over the system. However only some of them have been detailed in this section. Other tests or functionalities, not explained in this section but explained in other chapters, have been added to the table to show that all identified requirements have been fulfilled.

Moreover, this table also relates these tests to the identified requirement that fulfill, and shows a briefly description about requirements. A more extensive description of these requirement and system functionalities can be found in both chapters: project requirements and implementation.

Table 6.1: System tests

Identifier	Description	Requirement identifier
TEST1	Analysis of 1000 songs consecutively	REQ7 System must be stable REQ8 Server must be available 24 hours per day
TEST2	Interoperability: GNU/Linux and Windows	REQ2 Final users use Windows O.S. REQ3 Songs stored in Windows Server REQ4 AudioTool library compiled for GNU/Linux
TEST3	Concurrency in C++: between both threads	REQ7 System must be stable REQ11 New Songs should be automatically identified
TEST4	Integration with the Web Application	REQ7 System must be stable REQ8 The system should be rapid REQ10 The system should easily configurable
TEST5	Server running 3 days	REQ7 System must be stable REQ8 Server must be available 24 hours per day
TEST6	Corrupted files added to the system	REQ7 System must be stable REQ8 Server must be available 24 hours per day
TEST7	Stability in Web Application	REQ7 System must be stable
TEST8	Concurrency in Web Application	REQ15 Several users connected to the Web
TEST9	Processes duration test	REQ9 The system should be rapid
FUN1	System completely independent	REQ1 System should be completely independent
FUN2	System encodes songs to any format	REQ5 Songs format is MPEG-1 (Audio Layer 2) REQ6 Required format of songs is WAV
FUN3	System identifies automatically new songs	REQ11 New Songs should be automatically identified
FUN4	Web Application presents data	REQ12 Data should be presented to final users
FUN5	Web Application allows validating data	REQ13 Data should be able to be validated
FUN6	Web Application allows listening to songs	REQ14 Users should be able to listen to songs
FUN7	Two database implemented: songs and users	REQ16 Song data should be independent to user data.

Chapter 7

Conclusions

All the objectives, that were set at the beginning of the project, have been achieved. The three modules: the C++ server, the database and the Web Application, have been designed, implemented and documented. They have been integrated in a single system and they perform the functions revealed in the user requirements evaluation. A good integration design and implementation is essential for achieving a successful project. The system must be stable and rapid in order to be useful. However, the project will only be a success project if the audio processing module extracts good results. This way users would find the whole system functional as it would add value to their current workflow.
One important conclusion, that may be deducted of this project, is that when a project is planned, it is advisable to add some more days than expected in the planning to resolve unforeseen problems. Specifically, if the project makes use of some new technology (unknown to developers) or if the project is deployed in a new environment.

During the implementation and the design phase of the project some problems and setbacks have emerged. Thus, several implementation and design decisions have been made to meet the user requirements. The C++ server implementation has entailed more work and time than the first work plan supposed. Two main setbacks have been found: songs encoding and events notification. Both drawbacks have meant finding new solutions and, even, including new features to the system. Songs codification was more complex than expected. However, in compensation, the final way to encode the songs allows controlling the errors and encoding the songs to any type of format in an easy way. The implementation of event notification has meant implementing three different classes for the same action and with little compensation. The system is set to notify events in different run-time environment. However, the work has been higher than expected, since various solutions have been tested to achieve the good solution. The other functionalities of the C++ application have been achieved without major problems deserving to be mentioned. Regarding the database issue, two databases have been designed and implemented. The first one is shared by the C++ application and by the Web Application and contains the data about songs, as well as their analysis. The second one

is only used by the Web Application and contains the data about users. The design and implementation of this module does not involve any important delay in the project.
My knowledge about Web Application implementation and browser-supported languages was very little and the learning curve entailed spending more time than expected in this implementation phase. However, this module has been implemented without important problems deserving to be mentioned.

All the project goals have been achieved and a music management system has been designed and implemented. However, since it consists of a production scope system of the media sector, it should be kept in constant development in order to improve and adapt its functionalities. Thus, several improvements and changes appear constantly and they should be considered as future work.

This project has also let me learn several skills related to the degree and to my actual job. Some of these skills are directly linked with some degree subjects.

In the next points these two topics are explained:

- **Future work.** It explains what improvements have been identified during the project implementation and currently which the most important areas of future work are.
- **Learned skills.** It details what the main learned skills are and the relation with degree subjects.

7.1 Future work

The most important areas of immediate future work for this application are three:

1. To adapt the C++ server to specific user requirements and to new functionalities of new modules:
    - To finish defining specific user requirements in order to apply them to specific cases of the application. For instance, resolving specific cases such as, what to do when a song is inserted to the system twice or more: reprocessing the song or ignoring the song; or what to do if a user try to validate a song that has been validated before.
    - To adapt the C++ application for including new parameters that new modules will extract. To include new methods to extract these parameters and new methods to save them in the database.
    - To prepare the C++ application to be able to integrate the artificial intelligence module. This module should set the validate song parameters and recalculate its algorithm. In order to do this,

the system should indicate the validation date to set the correct validation and not repeat the retraining with the same validations.

2. To insert new tables in the database to save new data:
   - One table for each new parameters that the module will extract.
   - One table to save validations dates that users have been performed.
3. To improve the Web Application and to include new features:
   - To improve the player of the application. The current version downloads songs using an URL with a dynamic JSP . It is not the optimal solution, since all the song should download before starting to play the song. The API javax.sound provides the feature of downloading audios with real time protocol and applets example to perform this task in an optimal way.
   - To improve the charts that shows the Sflatness data. Currently, in order to perform this task the chart google API is used. However, this API only can be used if the server is connect to Internet. Other API's exists, more flexible and stable, to perform this task in a better way.
   - Other useful features would be to let users upload songs to the Server from the Web Application. The server would process the song and the results would be shown in the Web Application.
   - To save the user trace. It would allow to control who has perform a validation, when this validation has been done, what searches are more frequent, etc.

This point explains the immediate future work identified during the project implementation. However, it is pretty sure that this system will evolve and several improvement will emerge. The user test will reveal other requirements not expected before and new feature will be assigned as future work.

7.2 Learned skills and technologies

The design and implementation of this project has allowed me to deepen my knowledge of technical skills related to some subjects of the degree and to some actual trend technologies. Specifically, I have studied in depth topics about object-oriented programming (OOP). I have learnt the programming paradigm that uses "objects", data structures consisting of data fields and methods together with their interactions , to design applications. I have used programming features such as information hiding, data abstraction, encapsulation, modularity, polymorphism, and inheritance. One book [44] has been followed as reference, focused on C++ programming. However, several concepts can be applied to any object-oriented programming language (such as Java, also used in this project). Also Unified Modeling Language (UML) has been used to specify, visualize, modify and document the artifacts of the object-oriented software in the design phase of the project. Several subjects of the degree deals with programming languages as well logical programming, such as: *Introducció als Ordinadors* or *Arquitectura de Computadors i Sistemes Operatius 1.* However, only a slightest notion of object-oriented programming is explained in the subject: *Programació Concurrent* , related to Java programming. Some important skills applied in this project are outlined in subjects of the degree. Next, the main topics and the related subjects are explained. Both subjects, *Programació Concurrent* and *Arquitectura de Computadors i Sistemes Operatius 2*, deal with two important project topics: multi-threading programming and concurrent programming. The application that we have developed in C++ is multi-threading. These threads share resource and generate concurrency problem in the application. Mechanisms for supporting mutual exclusion in concurrent programs have been applied. Both subjects are also focused on using GNU/Linux operating system. The run-time environment of the developed system is also GNU/Linux. And the application should access network storage and works with different file systems (GNU/Linux and Windows file system). It has meant testing and running the application over different environment and finding different solutions for each environment. How file systems work in different operating system and amongst them is also outlined in the subject: *Arquitectura de Computadors i Sistemes Operatius 2.*

Actually, other important project skills are not related to the degree, such as databases and Web Application, or at least, not to the core degree subjects. I usually use these skills at work and I had a slightest notion about them. Moreover, this project allows me to better understand both topics. I have learnt to install and manage a database, as well as use different drivers or APIs to access to databases, such as JDBC or ODBC. I have learnt browser-supported language such as HTML, CSS styles, Javascript/Ajax, applets deployment and implementation, J2EE, or, specifically Struts2 framework, and how the different web browsers render Web applications. I have also manage Web Application deployment in a Server, as well as the configuration of Web Application servers.

To sum up, during the project, I have had the opportunity to learn several skills and better understand skills that I had a slightest notion. Some of these skills are very useful for my daily work and other skills are considered trend technologies and, thus, this knowledge can help me in my professional career.

Appendices

Appendix A

References

[1] Google chart API Web Page. http://code.google.com/apis/chart/, 2000.

[2] InnoDB Web Page. http://www.innodb.com/, 2000.

[3] OGNL Web Page. http://www.opensymphony.com/ognl/, 2000.

[4] Guice an Struts2 article on-line. http://www.tzavellas.com/techblog/2007/07/03/using-dependency-injection-in-struts2-for-stateless-ejbs-part-1/, 2007.

[5] DirectX Web Page. http://www.microsoft.com/games/en-US/aboutGFW/pages/directx.aspx, 2009.

[6] Dnotify Web Page. http://linux.die.net/man/1/dnotify, 2009.

[7] Dojo Toolkit Web Page. http://www.dojotoolkit.org/, 2009.

[8] FAM Web Page. http://oss.sgi.com/projects/fam/, 2009.

[9] FFMpeg Web Page. http://ffmpeg.org/, 2009.

[10] Google guice Web Page. http://code.google.com/p/google-guice/, 2009.

[11] IEEE Web Page. http://www.ieee.org/, 2009.

[12] Informix Web Page. http://www-01.ibm.com/software/data/informix/, 2009.

[13] Inotify Web Page. http://inotify.aiken.cz/, 2009.

[14] Introduction to CORBA. http://java.sun.com/developer/onlineTraining/corba/corba.html/, 2009.

[15] iODBC Web Page. http://www.iodbc.org, 2009.

[16] ISMIR Web Page. http://www.ismir.net/, 2009.

[17] Java Applet Web Page. http://java.sun.com/applets/, 2009.

[18] Java JDBC Web Page. http://java.sun.com/docs/books/tutorial/jdbc/index.html, 2009.

[19] JavaEE Web Page. http://java.sun.com/javaee/, 2009.

[20] JavaZoom Web Page. http://www.javazoom.net/, 2009.

[21] JBoss EJB3 Web Page. http://www.jboss.org/ejb3/, 2009.

[22] JBoss Web Page. http://www.jboss.org, 2009.

[23] JSON Web Page. http://json.org/, 2009.

[24] Linux ODBC Web Page. http://www.easysoft.com/developer/interfaces/odbc/linux.html, 2009.

[25] MySQL Web Page. http://www.mysql.com/, 2009.

[26] OMG Web Page. http://www.omg.org/, 2009.

[27] OmniORB Web Page. http://omniorb.sourceforge.net/docs.html/, 2009.

[28] OmniOrb4 development Status Web Page. http://www.omniorb-support.com/omniwiki/OmniOrb4DevelopmentStatus, 2009.

[29] Oracle Web Page. http://www.oracle.com/, 2009.

[30] PostgreSQL Web Page. http://www.postgresql.org/, 2009.

[31] SAMBA Web Page. http://www.samba.org/, 2009.

[32] Struts2 Web Page. http://struts.apache.org/2.x/, 2009.

[33] Valgrind Web Page. http://valgrind.org/, 2009.

[34] W3C Web Page. http://www.w3.org, 2009.

[35] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley Pub Col, 1997.

[36] Boehm B. A Spiral Model of Software Development and Enhancement. 1988.

[37] Bohme M. Using libavformat and libavcodec. http://www.inb.uni-luebeck.de/b̃oehme/using_libavcodec.html, 2004.

[38] C. Laurier and P. Herrera. Mood Cloud : A Real-Time Music Mood Visualization Tool. *CMMR, Computer Music Modeling and Retrieval, Copenaghen*, 2008.

[39] D. Turnbull, L. Barrington, D. Torres and G. Lanckriet. Semantic Annotation and Retrieval of Music and Sound Effects. *IEEE Transactions on Audio, Speech, and Language Processing*, 2008.

[40] E. Gamma, R. Helm, R. Johnson and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Pub Col, 1994.

[41] Ian Roughley. *Starting Struts2*. InfoQ, 2006.

[42] M. Henning and S. Vinoski. *Advanced CORBA Programming with C++*. Addison-Wesley Professional, 1999.

[43] P. Herrera, X. Amatriain, E. Batlle and X. Serra . Towards Instrument Segmentation for Music Content Description: A Critical Review of Instrument Classification Techniques. *Proc. of International Symposium on Music Information Retrieval (ISMIR 2000)*, 2000.

[44] S. Meyers. *Effective C++: 55 Specific Ways to Improve Your Programs and Designs*. Addison-Wesley Pub Col, 1997.

[45] T. Teorey, S. Lightstone, T. Nadeau. *Database Modeling and Design: Logical Design, 4th edition.* Morgan Kaufmann Press, 2005.

[46] Z. Duan, L. Lu and C. Zhang. Collective Annotation of Music from Multiple Semantic Categories. *ISMIR 2008 Session 2c Knowledge Representation, Tags, Metadata*, 2008.

Appendix B

Guia d'administrador

B.1 Base de dades

Instał.lar la base de dades postgreSQL. Es pot fer des de Synaptic package manager, però es recomana fer-ho de manera manual, seguint aquests passos:

- Descarregar postgres i executar les següents comandes:
  - `./configure without-readline`
  - `make`
  - `sudo make install`
  - `sudo adduser postgres` //definim un user per la BBDD ens demanarà pasword i dades
  - `sudo mkdir /usr/local/pgsql/data`
  - `sudo chwon postgres /usr/local/pgsql/data` //dones permisos a l'usuari postgres
  - `/usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data` // iniciem la BBDD
  - `/usr/local/pgsql/bin/postgres -D /usr/local/pgsql/data`
  - `/usr/local/pgsql/bin/createdb test`
  - `/usr/local/pgsql/bin/psql test`
- Des de Synaptic instał.lar:
  - ODBC driver per PostgreSQL (odbc-postgresql)
  - C++ ODBC database API (libsimpledb2)
- Configurar els fitxers **/etc/odbc.ini** i **/etc/odbcinst.ini**. La figura B.1 mostra el fitxer **/etc/odbcinst.ini** i la figura B.2 mostra el fitxer **/etc/odbc.ini** de configuració:

```
[postgre]
Description = postgre
Driver = postgre
Trace = Yes
TraceFile =
Database = audiotool
Servername = localhost
Username = postgres
Password = postgres
Port = 5432
Protocol = 6.4
ReadOnly = No
RowVersioning = Yes
ShowSystemTables = No
ShowOidColumn = No
FakeOidIndex = No
ConnSettings =
```

Figure B.1: Fitxer odbc.ini

```
[postgre]
Description = driver odbc for postgre
Driver = /usr/lib/odbc/psqlodbca.so
UsageCount = 1
CPTimeout = 1000
CPReuse = 100
```

Figure B.2: Fitxer odbcinst.ini

- Instał.lar pgadmin3 per gestionar la base de dades. Si la base de dades es troba en local al camp host escriure: /tmp i el port per defecte:
  - Crear una base de dades amb el nom **audiotool**. Si es vol canviar el nom de la base de dades, recordar canviar també al fitxer de configuració **/etc/odbc.ini** el tag Database.

  - Executar el fitxer AudioToolBBDD.sql en aquesta base de dades.
  - Inserir a la taula Server, els servidors que la aplicació ha de monitoritzar i el seu identificador. El nom del servidor ha de contenir una barra davant (p.ex. si es tracta del server ssvsa301, inserir-ho com: **/ssvsa301**).
  - Inserir els directoris (a la taula directory) dels servidors que s'han de monitoritzar amb el seu respectiu identificador i el identificador del servidor al que pertany. Ficar el camp booleà monitor a **true**. També inserir el path amb barres (p.ex. si es tracta del directori CatRadio/AUDIOICATFM, inserir-ho com: **/CatRadio/AUDIOICATFM**).
  - Crear una base de dades amb el nom **users**.
  - Executar el fitxer Users.sql en aquesta base de dades.
  - Inserir els usuarios, els passwords i els noms dels usuaris que es desitgin.

B.2 Servidor

- Editar el fitxer de configuració de l'aplicació audiotool (audiotool.inp). S'ha d'indicar el nom de la base de dades, l'usuari i la contrasenya. També s'ha d'indicar la IP de la màquina on està corrent l'aplicació per la que la connexió CORBA sortirà i el port (que ha de ser major a 1025). Per últim indicar si l'aplicació monitoritzarà directoris Windows o no. La figura B.3 mostrà aquest fitxer de configuració:

```
    # configuration file audiotool.inp
windows share = true # if is connected to a Windows server
bbdd name = postgre  # name bdbb
bbdd user = postgres   # user bbdd
bbdd password = postgres # password bbdd
IP Corba = 192.168.163.17 #IP Corba
Port Corba = 1025 #Port Corba
```

Figure B.3: Fitxer audiotool.inp

- Obrir el port escollit a la màquina on s'executarà aquesta aplicació.
- Les dependències de l'aplicació AudioTool són les següents llibreries:
  - omniORB4

  - libc
  - libz
  - avformat
  - avcodec
  - avutil
  - audiotools
  - mad
  - pthread
  - asound
  - iobdc

  Si es vol executar en una màquina que no sigui GNU/Linux, també s'haurà d'incloure la llibreria fam. Totes aquestes llibreries es poden descarregar des de Synaptic excepte la llibreria AudioTool que s'ha de copiar a **/$HOME/usr/lib**

- Si es vol utilitzar servidors en Windows connectats a la xarxa local, s'haurà d'utilitzar SAMBA. Per això editar el fitxer **/etc/fstab**, i incloure una entrada del tipus: **//192.168.163.12/CatRadio /opt/CatRadio/ cifs iocharset=utf8,rw,auto,username=postgres,password=postgres 0 0** On sindica el path del servidor, el punt de montatge, el tipus de fitxer, la codificació, els permisos, l'usuari i el nom del servidor. El punt de muntatge és el que s'ha dindicar a la base de dades (a les taules server i directory).
- Ja pots executar l'aplicació AudioTool. Quan una nova cançó entri en un dels directoris monitoritzats serà processada.

B.3 Aplicació Web

L'aplicació Web corre sobre un servidor JBoos. Per fer servir aquest servidor:

- Descarregueu l'última versió de la pàgina Web: http://www.jboss.org/jbossas/downloads/
- Copieu l'arxiu WAR, generat per aquest projecte, a la carpeta: **jboss-5.1.0.GA/server/default/deploy**
- Engegueu el servidor: entreu a la carpeta **jboss-5.1.0.GA/bin**, i executeu la comandada: **./sh run.sh -b 0.0.0.0.** (l'opció -b és per engegar el servidor en mode no local).

- Si voleu canviar el port on s'executarà JBoos (per defecte 8080):
  - Obriu l'arxiu: **jboss-5.1.0.GA/server/default/deploy/jbossweb.sar/server.xml**
  - Modifiqueu el tag: **Connector protocol="HTTP/1.1" port="8080" address="$jboss.bin.address" connectionTimeout="20000" redirectPort="8443"** pel port desitjat. Això és vàlid per la versió JBoos 5.1, per altres versions s'hauria de mirar la documentació de la versió en concret.

Appendix C

Guia d'usuari

Per utilitzar aquesta aplicació, primer de tot s'ha de tenir un usuari i una contrasenya, que s'haurà de demanar a l'administrador del sistema.

- Quan l'aplicació comenci demanarà aquestes dades per poder entrar. La figura C.1 mostrà la pantalla per registrar-se:

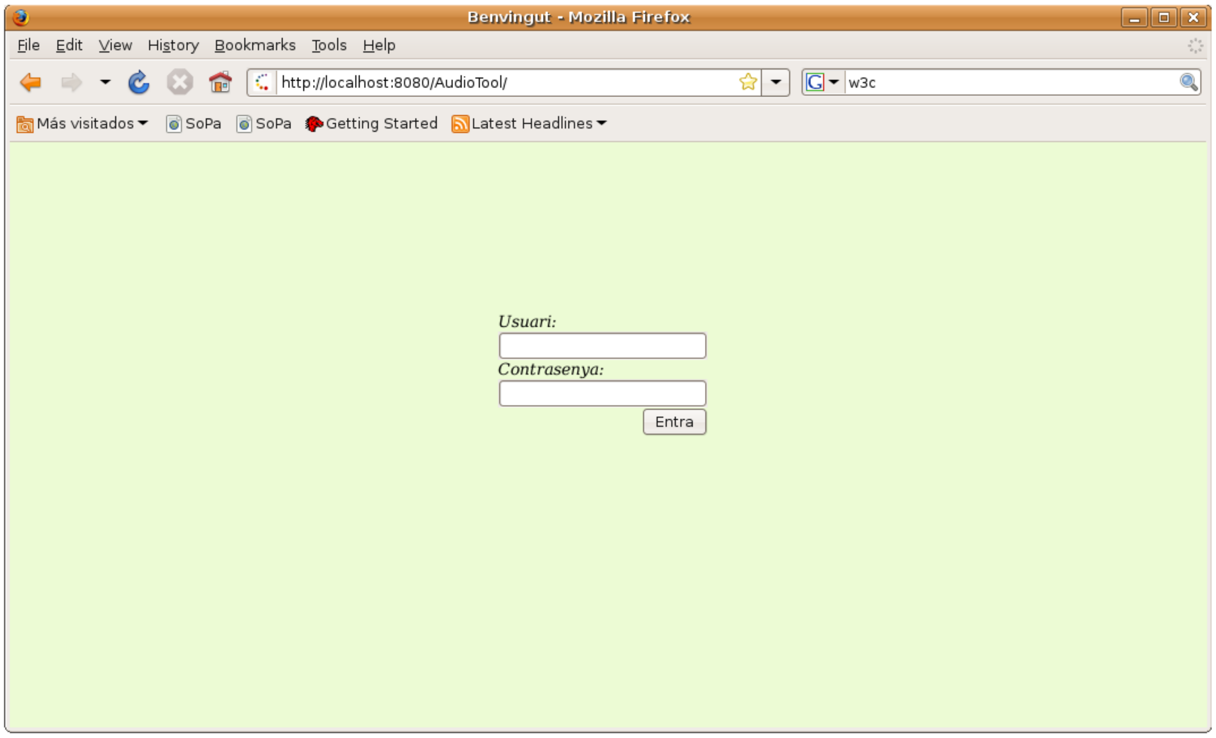
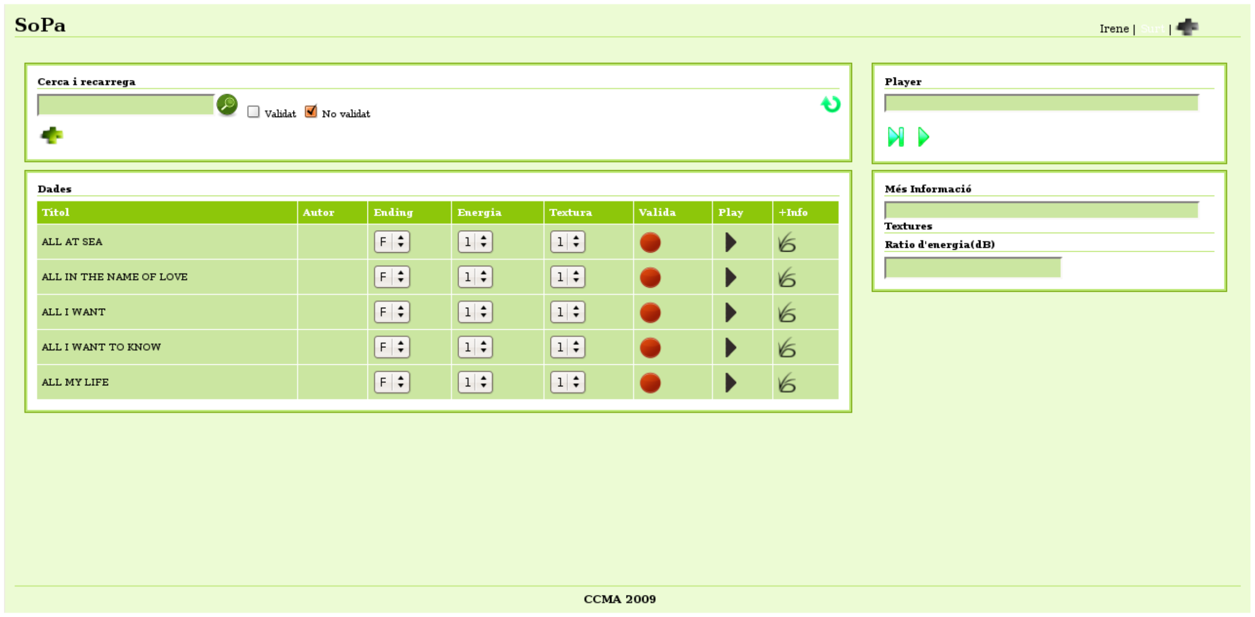

Benvingut - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/AudioTool/
w3c
Más visitados SoPa SoPa Getting Started Latest Headlines
Usuari:
Contrasenya:
Entra

Figure C.1: Pantalla per registrar-se

- Després l'aplicació mostrarà en una taula les últimes 5 cançons, amb les dades corresponent, que han entrat al sistema i encara no s'han validat. La figura C.2 mostra la pantalla inicial:



SoPa

Irene | Surt |

Cerca i recarrega

☐ Validat ☑ No validat

Player

Més Informació

Textures

Ratio d'energia(dB)

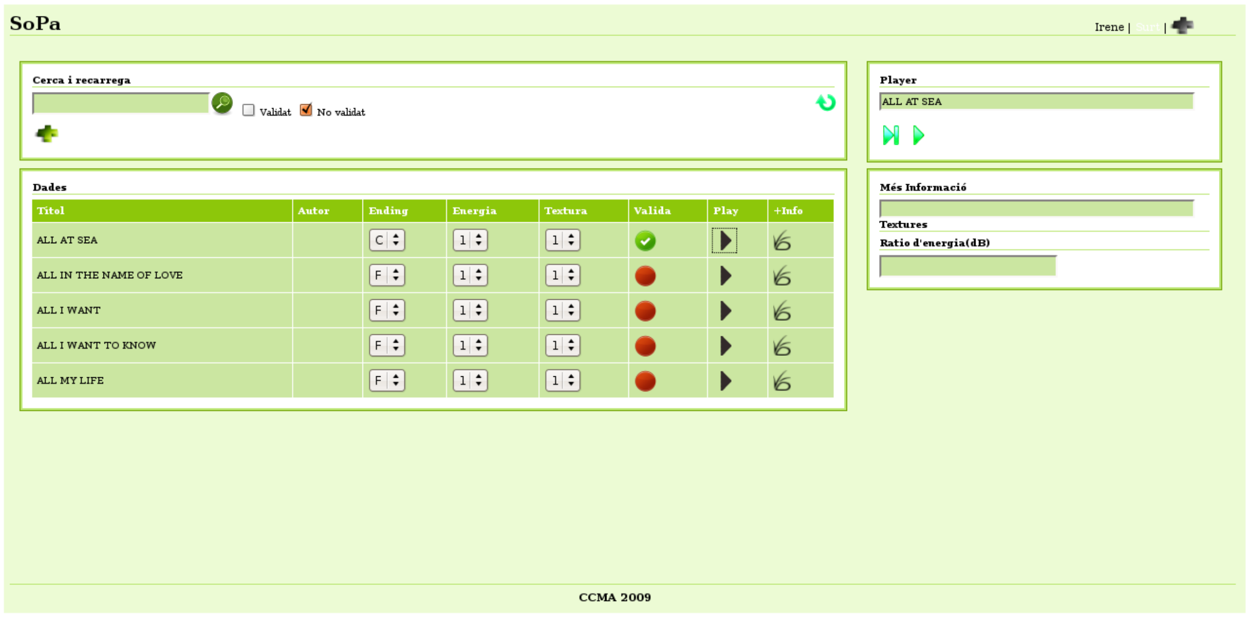
Dades

Titol	Autor	Ending	Energia	Textura	Valida	Play	+Info
ALL AT SEA		F	1	1	●	▶	
ALL IN THE NAME OF LOVE		F	1	1	●	▶	
ALL I WANT		F	1	1	●	▶	
ALL I WANT TO KNOW		F	1	1	●	▶	
ALL MY LIFE		F	1	1	●	▶	

CCMA 2009

Figure C.2: Pantalla inicial

- Aquesta taula mostra si una cançó està validada o no, amb un botó vermell o verd. Mostra els paràmetres d'anotació proposats pel sistema en diferents menús desplegables. L'usuari pot canviar aquests valors, segons el seu criteri, i per últim validar la cançó. Per fer aquest últim pas, només s'ha de pitjar el botó vermell. Si no hi ha hagut cap error, el botó es convertirà en verd. Una cançó ja validada no es pot tornar a validar (un pop-up apareixerà per indicar que no es pot realitzar aquesta validació). La figura C.3 mostra una cançó validada:


SoPa

Irene | |

Cerca i recarrega

☐ Validat ☑ No validat

Player

ALL AT SEA

Dades

Titol	Autor	Ending	Energia	Textura	Valida	Play	+Info
ALL AT SEA		C	1	1	✓	▶	
ALL IN THE NAME OF LOVE		F	1	1	●	▶	
ALL I WANT		F	1	1	●	▶	
ALL I WANT TO KNOW		F	1	1	●	▶	
ALL MY LIFE		F	1	1	●	▶	

Més Informació

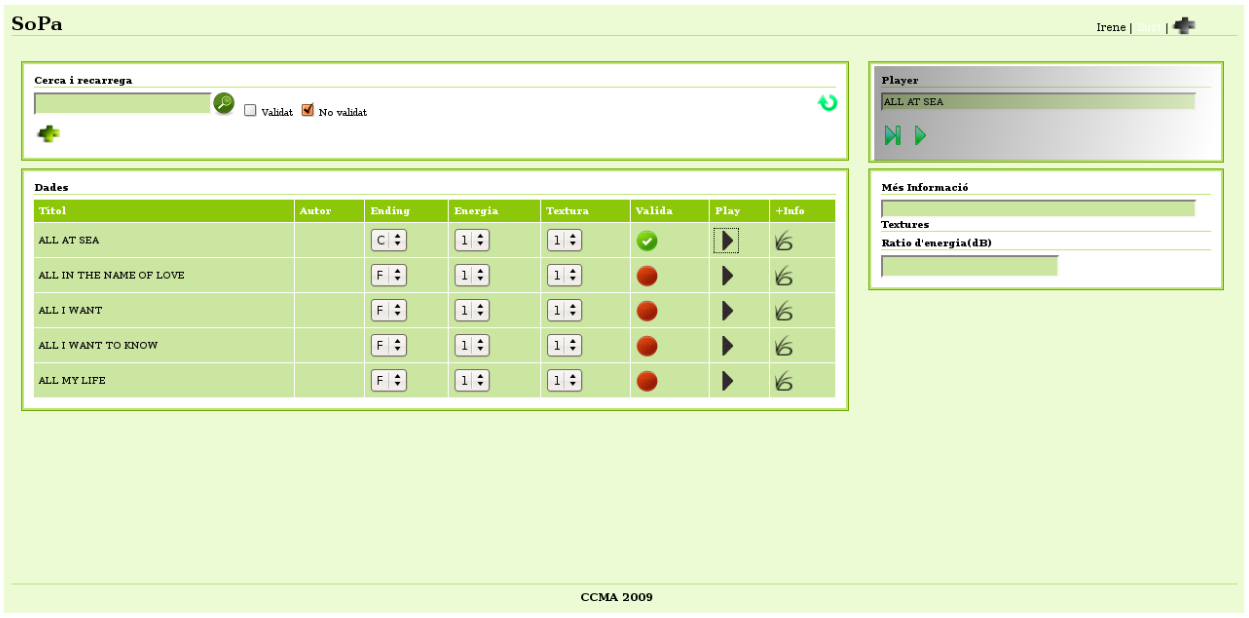
Textures

Ratio d'energia(dB)

CCMA 2009

Figure C.3: Cançó validada

- El sistema també permet escoltar les cançons que s'estan validant. Cada fila de cada cançó conté un botó de play per escoltar la cançó en qüestió. El nom de la cançó que està sonant sortirà a l'àrea de Player. Des de aquesta àrea també es pot fer una pausa de la cançó i tornar a fer un play. La figura C.4 mostra el player de l'aplicació:

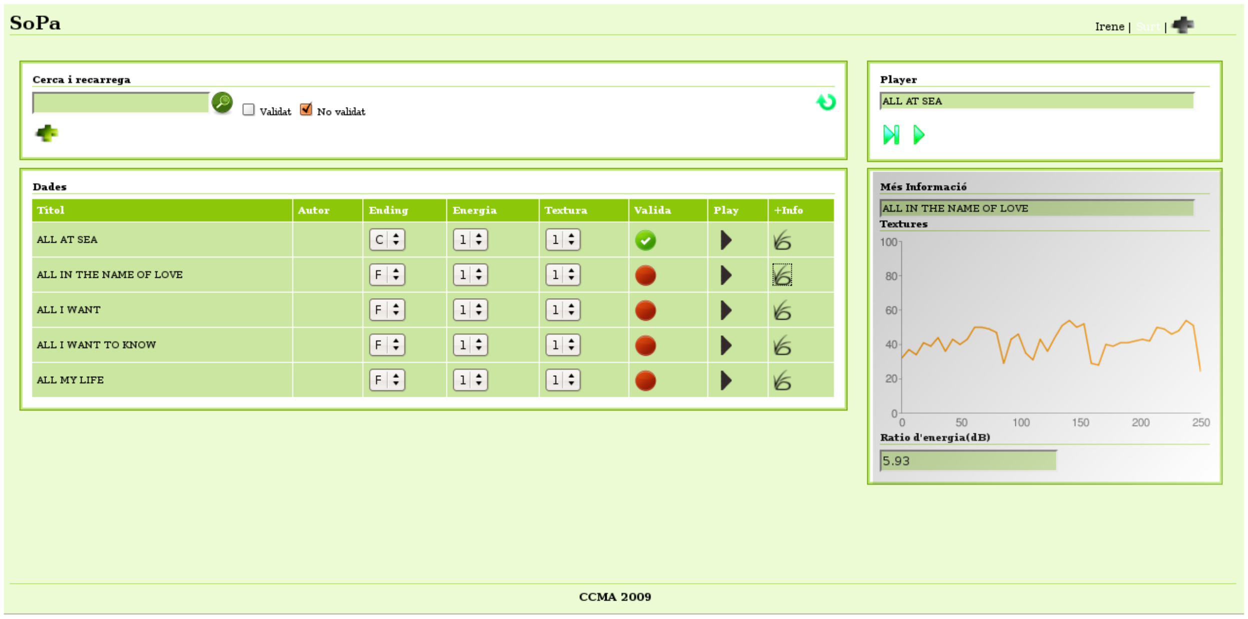


Dades

Titol	Autor	Ending	Energia	Textura	Valida	Play	+Info
ALL AT SEA		C	1	1	✓	▶	
ALL IN THE NAME OF LOVE		F	1	1	●	▶	
ALL I WANT		F	1	1	●	▶	
ALL I WANT TO KNOW		F	1	1	●	▶	
ALL MY LIFE		F	1	1	●	▶	

Figure C.4: Player de l'aplicació

- L'aplicació també permet veure les dades complementàries de les cançons. Per això hi ha un botó en forma d'ull a l'última columna de la taula. Pitjant aquest botó apareixerà la gràfica de les textures de la cançó en qüestió i la dada de ràtio d'energia. La figura C.5 mostra les dades complementàries:



SoPa

Irene |

Cerca i recarrega

Validat [✓ No validat]

Dades

Títol	Autor	Ending	Energia	Textura	Valida	Play	+Info
ALL AT SEA		C	1	1	✓	▶	
ALL IN THE NAME OF LOVE		F	1	1	●	▶	
ALL I WANT		F	1	1	●	▶	
ALL I WANT TO KNOW		F	1	1	●	▶	
ALL MY LIFE		F	1	1	●	▶	

Player

ALL AT SEA

Més Informació

ALL IN THE NAME OF LOVE

Textures

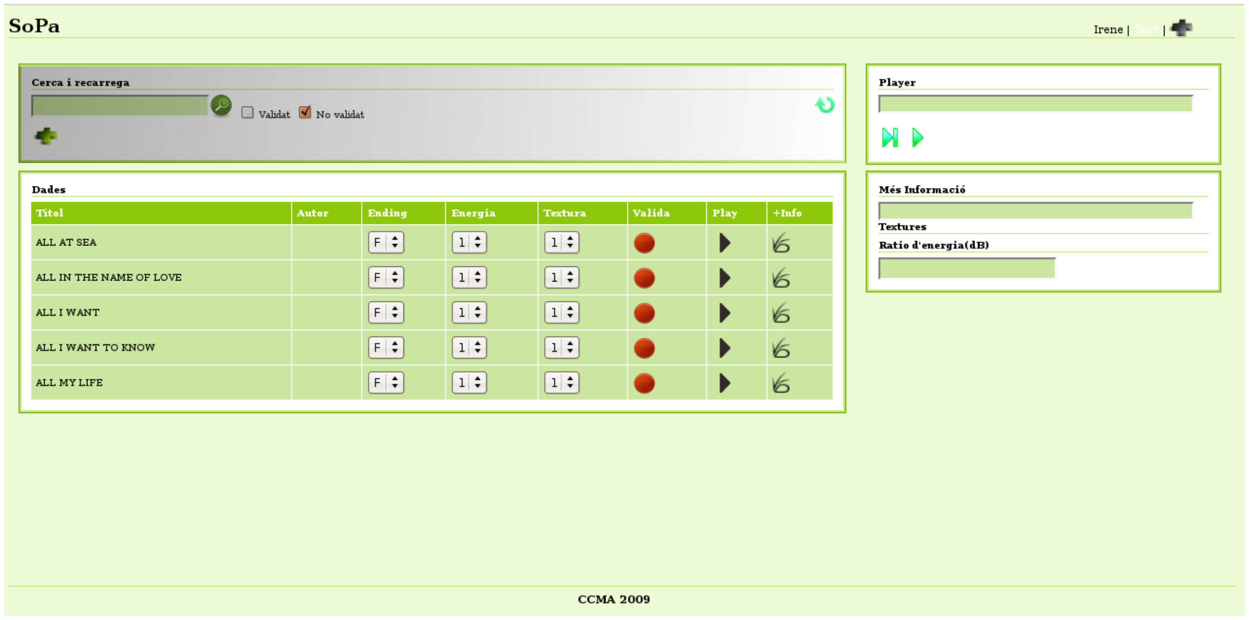
Ratio d'energia(dB)

5.93

CCMA 2009

Figure C.5: Dades complementàries

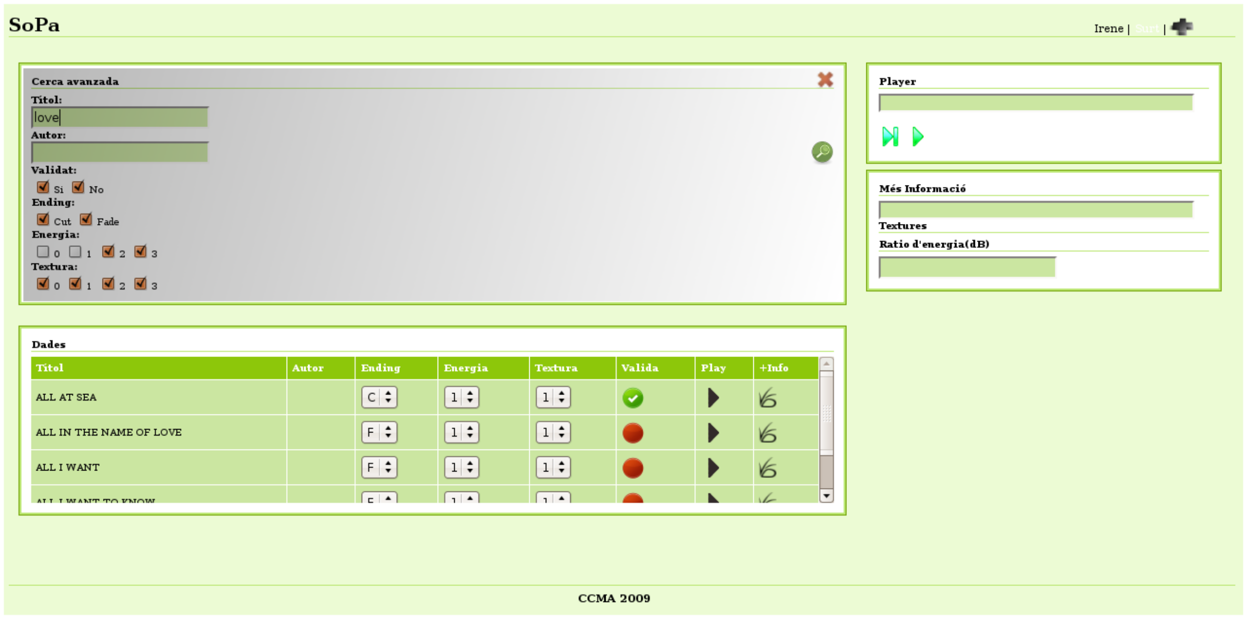
- El sistema permet fer cerques simples i avançades a la base de dades. La cerca simple surt per defecte a la part superior de les dades. Aquesta cerca permet realitzar cerques sobre el nom i l'autor de les cançons i també marcar si els resultats han d'estar validats o no. La figura C.6 mostra la cerca simple:



Titol	Autor	Ending	Energia	Textura	Valida	Play	+Info
ALL AT SEA		F	1	1	●	▶	
ALL IN THE NAME OF LOVE		F	1	1	●	▶	
ALL I WANT		F	1	1	●	▶	
ALL I WANT TO KNOW		F	1	1	●	▶	
ALL MY LIFE		F	1	1	●	▶	

Figure C.6: Cerca simple

- Dintre de la cerca simple també hi ha un botó de color verd (amb forma de fletxa recargolada) que permet recarregar l'aplicació amb noves cançons no validades, quan l'usuari acabi de validar les que inicialment apareixen.
- Per entrar a la cerca avançada s'ha de pitjar la creu verda que està a la cerca simple. Noves opcions apareixen a aquesta zona de la interfície. La cerca avançada permet escollir sobre tots els paràmetres. Per tornar a la cerca simple, pitjar la creu vermella de la part superior dreta. La figura C.7 mostra la cerca avançada:


SoPa

Irene | |

Cerca avanzada

Titol: love

Autor:

Validat: [✓ Si] [✓ No]

Ending: [✓ Cut] [✓ Fade]

Energia: 0 1 [✓ 2] [✓ 3]

Textura: [✓ 0] [✓ 1] [✓ 2] [✓ 3]

Player

Més Informació

Textures

Ratio d'energia(dB)

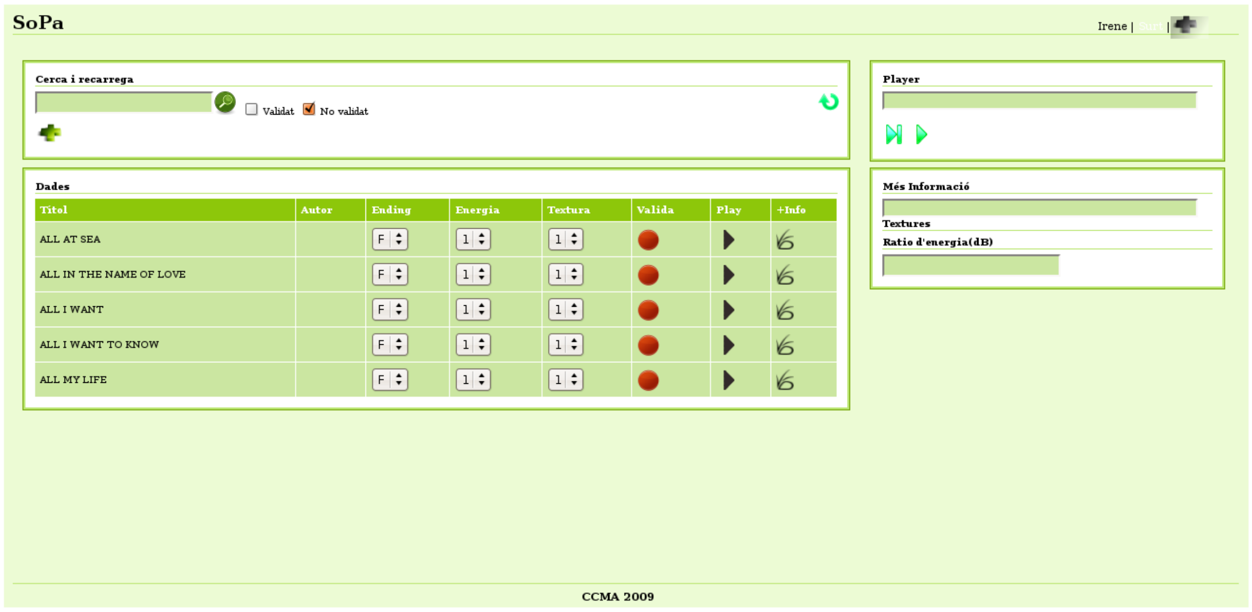
Dades

Titol	Autor	Ending	Energia	Textura	Valida	Play	+Info
ALL AT SEA		C	1	1	✓	▶	
ALL IN THE NAME OF LOVE		F	1	1	●	▶	
ALL I WANT		F	1	1	●	▶	
ALL I WANT TO KNOW		F	1	1	●	▶	

CCMA 2009

Figure C.7: Cerca avançada

- Per entrar a l'administració del sistema s'ha de pitjar la creu grisa que es troba a la part superior dreta de l'aplicació. La figura C.8 mostra el botó administració:


SoPa

Irene | |

Cerca i recarrega

Validat [✓ No validat]

Player

Més Informació

Textures

Ratio d'energia(dB)

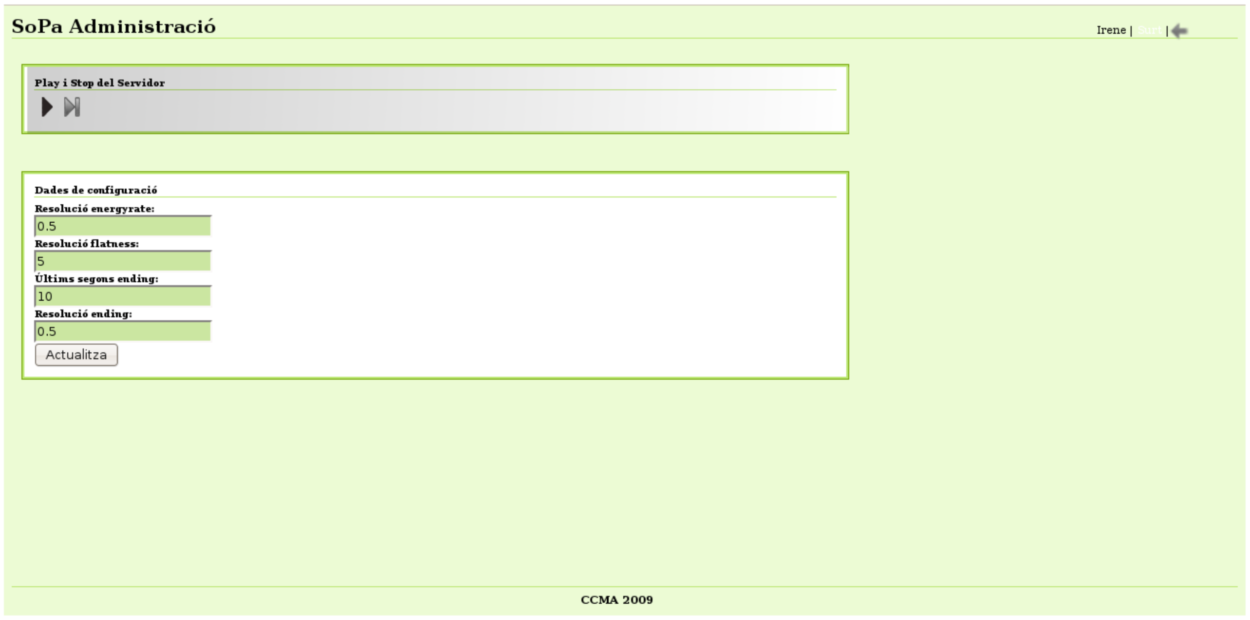
Dades

Titol	Autor	Ending	Energia	Textura	Valida	Play	+Info
ALL AT SEA		F	1	1	●	▶	
ALL IN THE NAME OF LOVE		F	1	1	●	▶	
ALL I WANT		F	1	1	●	▶	
ALL I WANT TO KNOW		F	1	1	●	▶	
ALL MY LIFE		F	1	1	●	▶	

CCMA 2009

Figure C.8: Botó administració

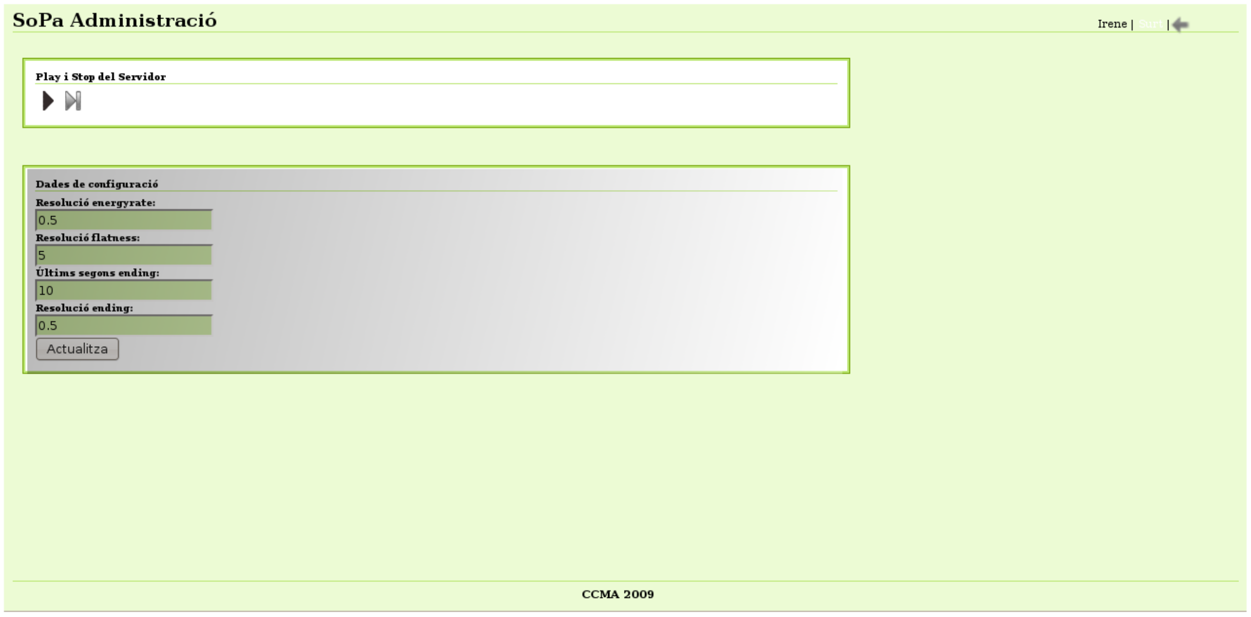
- Una nova pantalla apareixerà amb les opcions avançades. La part superior conté una àrea on es troben dos botons per engegar i parar el servidor en C++. Si el servidor està apagat un pop-up indicarà que no es pot connectar amb el servidor. Si el servidor esta engegat i s'intentà tornar a engegar, un pop-up ho indicarà. De la mateixa manera si s'intentà parar i ja està parat, també s'indicarà. La figura C.9 mostra les opcions avançades que permeten parar i engegar el servidor:



Resolució energyrate: 0.5
Resolució flatness: 5
Últims segons ending: 10
Resolució ending: 0.5

Figure C.9: Parar i engegar el servidor

- L'aplicació també permet recarregar els parmetres de l'anàlisi de cançons del servidor. Per això una nova àrea amb els actuals paràmetres es mostra, i permet canviar-los i actualitzar el servidor. La figura C.10 mostra aquesta opció:



SoPa Administració

Irene | Surt |

Play i Stop del Servidor

Dades de configuració

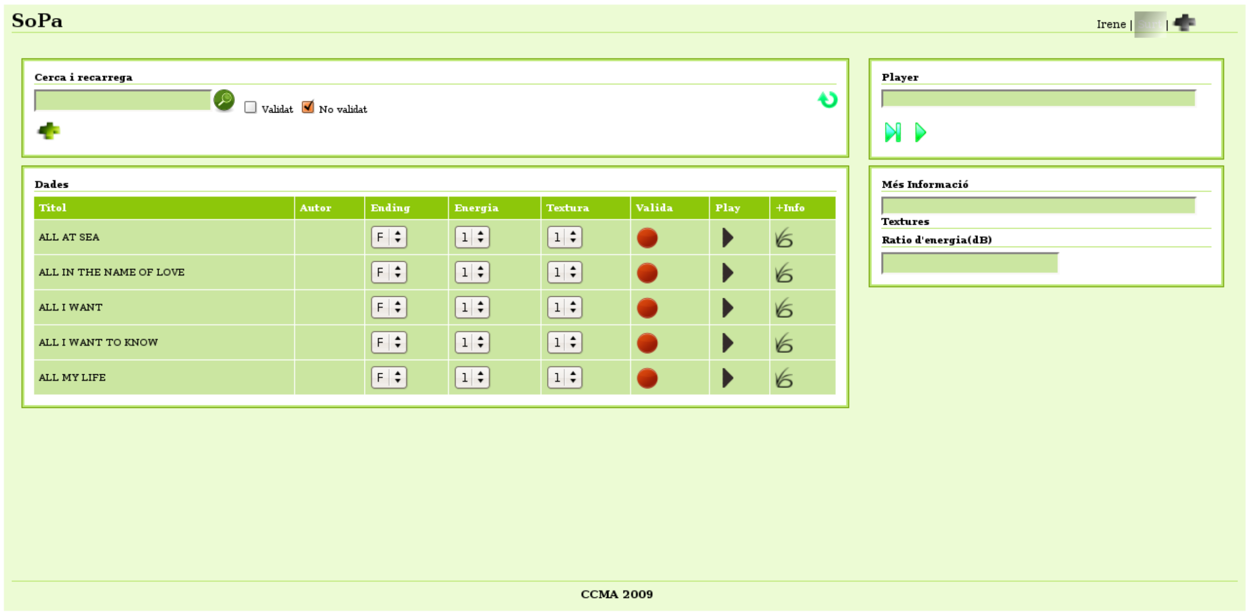
Resolució energyrate:	0.5
Resolució flatness:	5
Últims segons ending:	10
Resolució ending:	0.5

Actualitza

CCMA 2009

Figure C.10: Recarrega de paràmetres

- Per tornar a la pantalla anterior, una fletxa grisa es troba a la part dreta superior de la pantalla. Per sortir de l'aplicació, la pantalla conté un link amb la paraula surt (també a la part superior dreta), que et porta a la pantalla de registre d'usuari inicial. La figura C.11 mostra el link de sortida:



SoPa

Irene | |

Cerca i recarrega

☐ Validat [✓ No validat]

Player

Dades

Titol	Autor	Ending	Energia	Textura	Valida	Play	+Info
ALL AT SEA		F	1	1	●	▶	
ALL IN THE NAME OF LOVE		F	1	1	●	▶	
ALL I WANT		F	1	1	●	▶	
ALL I WANT TO KNOW		F	1	1	●	▶	
ALL MY LIFE		F	1	1	●	▶	

Més Informació

Textures

Ratio d'energia(dB)

CCMA 2009

Figure C.11: Sortida de l'aplicació