Radia Source - Sistema de Informação para Gestão de Processos de uma Estação de Rádio

Daniel Enrique Zacarias Silva

Dissertação para obtenção do Grau de Mestre em Engenharia Informática e de Computadores

Júri

Presidente: Prof. José Manuel Nunes Salvador Tribolet
Orientador: Prof. José Luís Brinquete Borbinha
Vogais: Prof. António Manuel Ferreira Rito da Silva

Setembro de 2008
Acknowledgements

I would like to thank everyone that influenced this work and helped me get to the finish line, specially:

My family, for their constant support and unwavering love.

My friends, for their company and always keeping me sane.

My advisor, José Borbinha, for his suggestions and guidance in the shaping of this work.

Everyone at Rádio Zero and the Radia network, for their continuous help and influx of ideas.

This work is dedicated to everyone who strives for a different notion of radio: one where a community of people and their interests take the front seat, leaving computers, music playlists and impersonality to a secondary role.

(...) The boom was fantastic, in a sense, but it puzzled us. We had intended to establish a free radio station, not to transmit a one-way performance that disregarded listeners as most stations did. During the boom, most mini-FM stations were able to communicate to a handful of people only.

But many of these stations seemed to us to be naively copying professional radio studio work. On the contrary, we paid attention to constant and serious listeners and sought to provide a community of people with alternative information on politics and social change.

The radio station that my students and I had started on the campus re-established itself in the centre of Tokyo when the students finished school in 1983. The new station was called Radio Home Run. Every day, from 8 PM to midnight, one or two groups aired talk or music programs. Themes varied, but members always invited new guests who were involved in political or cultural activism. Repeating the station telephone number during each program was our basic policy.

Listeners who lived close to the station began to visit. Guests sometimes recorded cassette tapes of our programs and let their friends listen. Radio Home Run quickly became a meeting place for students, activists, artists, workers, owners of small shops, local politicians, men, women and the elderly. The function was centripetal rather than centrifugal: listeners tended to want to come to the station.

—Tetsuo Kogawa in From Mini-FM to Polymorphous Radio
Abstract

The operation of radio stations is defined by three main factors: content, production environment, and resources. The content of radio station programming schedules is mostly composed of automated music playlists or programs produced by their authors. Also, content contributions may come from a controlled environment or from different sources. Finally, the availability of human and financial resources also have a great impact on their operation.

Currently, there is no adequate application support for radio stations focused on author programming, that receive content from different sources and have low human and financial resources. This work proposes a solution for radio stations with the aforementioned characteristics. The proposed solution is based on an analysis of a station of this type, as well as requirements elicited from several radio stations with similar characteristics.

A design and implementation for an information system is presented, exploring the Representational State Transfer (REST) architectural style. The system supports the main processes associated with the broadcast in such stations, including the programming schedule definition, content contribution by authors and its usage for audio playback. The implemented prototype was submitted to user evaluation, and was considered to respond to their needs.

Keywords
Radio, Programming Schedule, Information System, REST, Resources
Resumo
A operação de estações de rádio é definida por três factores principais: conteúdo, ambiente de produção e recursos. O conteúdo das grelhas de programação das rádios é principalmente composto de listas de músicas automaticamente geradas ou por programas produzidos pelos seus autores. Adicionalmente, as contribuições desses conteúdos podem vir de um ambiente controlado ou de diferentes fontes. Finalmente, a disponibilidade de recursos humanos e financeiros também um grande impacto nas suas operações.

Atualmente não há suporte aplicacional adequado para estações de rádio focadas em programas de autor, que recebem conteúdos de diferentes fontes e que têm baixos recursos humanos e financeiros. Este trabalho propõe uma solução para estações de rádio com as características previamente referidas. A proposta de solução assenta na análise de uma estação deste tipo, assim como em requisitos levantados a partir de estações de rádio com características similares.

Um desenho e implementação para um sistema de informação é apresentado, explorando o estilo arquitetural Representational State Transfer (REST). O sistema suporta os principais processos associados com a emissão nessas estações, incluindo a definição da grelha de programação, a contribuição de conteúdos e a sua utilização na emissão do áudio. O protótipo implementado foi submetido a avaliação por utilizadores e foi considerado corresponder às suas necessidades.

Palavras-chave
Rádio, Grelha de Programação, Sistema de Informação, REST, Recursos
# Table of Contents

Acknowledgements .............................................. i
Abstract .......................................................... iii
Resumo .................................................................. v
List of Figures ......................................................... xii
List of Tables ........................................................ xiii
List of Acronyms ..................................................... xv

1 Introduction ......................................................... 1
   1.1 Context and Motivation ...................................... 1
   1.2 Dissertation Statement ...................................... 2
   1.3 Dissertation Structure ...................................... 3

2 Background and Related Work .................................. 5
   2.1 Introduction .................................................. 5
   2.2 Key Concepts in Radio Stations ............................ 5
   2.3 Description of a Programming Schedule .................. 6
   2.4 Typical Radio Station Applications ....................... 7
   2.5 Survey of Radio Station Applications ..................... 8
      2.5.1 Proprietary Software .................................. 9
      2.5.2 Open Source Software ............................... 11
   2.6 Systems Comparison ........................................ 16
   2.7 Metadata Schemas ......................................... 18
      2.7.1 TV-Anytime ........................................... 19
      2.7.2 Dublin Core ........................................... 19
      2.7.3 EBU Core Metadata Set for Radio Archives ...... 19
      2.7.4 Exchange Broadcast Binary and Metadata Format .. 20

3 Problem Statement ............................................... 21
   3.1 Introduction ................................................ 21
   3.2 Categorization of Radio Stations ........................... 21
   3.3 Organizational Description of Rádio Zero ............... 23
      3.3.1 Organizational Structure ............................. 23
      3.3.2 Management .......................................... 24
      3.3.3 Technology .......................................... 25
      3.3.4 Broadcast Business Processes ....................... 25
   3.4 Problem .................................................... 30
## List of Figures

2.1 Concept map of a radio station schedule. .......................................................... 6
3.1 Overall organization of Rádio Zero. ................................................................. 23
3.2 Business Process for the first transmission of pre-recorded broadcasts. ............ 26
3.3 “Delivery” activity for pre-recorded broadcasts. ............................................... 27
3.4 “Validation” activity for pre-recorded broadcasts. ........................................... 27
3.5 “Post-processing” activity for pre-recorded broadcasts. ................................... 28
3.6 Business process for repetitions of pre-recorded broadcasts. ........................... 29
3.7 Business process for the first transmission of live broadcasts. .......................... 29
3.8 “Author post-processing” activity for live broadcasts. ..................................... 30
3.9 Business process for repetitions of live broadcasts. ........................................ 30
3.10 Evolution of weekly first-time broadcasts at Rádio Zero. ................................. 31
3.11 General administration. .................................................................................. 33
3.12 Schedule management. ................................................................................... 34
3.13 Management options for programs. ................................................................. 35
3.14 Author management. ...................................................................................... 35
3.15 Author's area. ................................................................................................. 36
3.16 Editor management. ....................................................................................... 36
3.17 Functions accessible to unregistered users. ..................................................... 37
4.1 Domain Model. ................................................................................................. 39
4.2 Components and the interfaces that connect them. .......................................... 43
4.3 System actors. ................................................................................................ 43
4.4 Statuses of a broadcast in relation to its audio assets. ...................................... 45
4.5 Interaction overview of the schedule update scenario. .................................... 52
4.6 Load schedule from calendar. ......................................................................... 52
4.7 Resolve conflicts between existing broadcasts and those being created. .......... 53
4.8 Confirmation of schedule update results from user. ......................................... 53
4.9 Querying for audio asset availability. ............................................................... 54
4.10 Delivery of audio asset by an author. ............................................................... 55
4.11 Components for the Playout module. .............................................................. 55
4.12 Updating programming schedule information. .............................................. 58
4.13 Exporting schedule configuration from Scheduler format to Player format. ...... 59
4.14 Fetching audio assets for playout. ................................................................. 60
5.1 Directory structure of a Rails application. ....................................................... 62
5.2 Model-View-Controller as an architectural style for web applications. .......... 62
5.3 Deployment scheme. ....................................................................................... 65
5.4 Schedule timeline. .......................................................................................... 68
List of Tables

2.1 Supported functional areas for MAR4Win ................................................. 9
2.2 Supported functional areas for Jazler .................................................. 10
2.3 Supported functional areas for Raduga ............................................... 10
2.4 Supported functional areas for ZaraRadio ........................................... 11
2.5 Supported functional areas for Reboot.fm ........................................... 13
2.6 Supported functional areas for Campcaster ......................................... 14
2.7 Supported functional areas for Rivendell ............................................. 15
2.8 Supported functional areas for SomaSuite .......................................... 15
2.9 Supported functional areas for Savonet .............................................. 16
2.10 Feature list for proprietary schedulers .............................................. 17
2.11 Feature list for Open Source schedulers .......................................... 18

3.1 Alignment of Software Systems with Radio Station Types ......................... 31

4.1 Schedule resource description ......................................................... 47
4.2 Program resource description ........................................................... 47
4.3 Broadcast resource description ........................................................ 48
4.4 Gap resource description ................................................................. 48
4.5 Single resource description .............................................................. 49
4.6 Playlist resource description ............................................................ 49
4.7 Editor resource description ............................................................... 49
4.8 Author resource description ............................................................. 50
4.9 User resource description ................................................................. 50
4.10 User mailbox resource description .................................................... 51
4.11 User mailbox messages resource description ...................................... 51
4.12 Gap resource description (at Playout service) ..................................... 57
4.13 Single resource description (at Playout service) .................................. 57
4.14 Play logs resource description .......................................................... 58

5.1 CRUD operations and HTTP verbs as REST interface ............................. 63
5.2 Summary of implemented web resources in the Scheduler module ............... 66
5.3 Summary of implemented web resources in the Playout module ................. 73

6.1 Main editorial activities and average time to perform them ........................ 75
## List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AJAX</td>
<td>Asynchronous JavaScript and XML</td>
</tr>
<tr>
<td>AM</td>
<td>Amplitude Modulation</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>BPMN</td>
<td>Business Process Modeling Notation</td>
</tr>
<tr>
<td>CRUD</td>
<td>Create, Read, Update, Delete</td>
</tr>
<tr>
<td>DAM</td>
<td>Digital Asset Management</td>
</tr>
<tr>
<td>DCMI</td>
<td>Dublin Core Metadata Initiative</td>
</tr>
<tr>
<td>DCMES</td>
<td>Dublin Core Metadata Element Set</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name Service</td>
</tr>
<tr>
<td>EBU</td>
<td>European Broadcasting Union</td>
</tr>
<tr>
<td>EPG</td>
<td>Electronic Program Guide</td>
</tr>
<tr>
<td>FM</td>
<td>Frequency Modulation</td>
</tr>
<tr>
<td>FOSS</td>
<td>Free and Open Source Software</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>GNU</td>
<td>GNU's Not Unix</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IS</td>
<td>Information System(s)</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organizations for Standardization</td>
</tr>
<tr>
<td>MIME</td>
<td>Multipurpose Internet Mail Extensions</td>
</tr>
<tr>
<td>MVC</td>
<td>Model-View-Controller</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>SMB</td>
<td>Server Message Block</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Transfer Control Protocol / Internet Protocol</td>
</tr>
<tr>
<td>TV</td>
<td>Television</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>WWW</td>
<td>World Wide Web</td>
</tr>
<tr>
<td>XBMF</td>
<td>Exchange Broadcast Binary and Metadata Format</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XML-RPC</td>
<td>XML Remote Procedure Call</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Context and Motivation

The core activity for a radio station is to produce a broadcast signal based on a programming schedule. There are two main types of users in a radio station: editors and authors. Editors are responsible for the definition of the station’s programming and ensuring its proper execution. Authors are the content creators responsible for the production of a station’s broadcasts. The type of content in a programming schedule varies from station to station, but it can be generalized as follows:

- **Broadcasts**: the main elements in a programming schedule, corresponding to the instances in which radio programs occur. Broadcasts have a pre-defined beginning and ending dates and times. The production of their contents can be either *pre-recorded* or *live*.
- **Intermissions**: timeframes between broadcasts that feature the use of informational, promotional or advertisement spots\(^1\).

There are various areas supported by software in a modern radio stations. Applications to support a radio station can be divided according to the following functional groups:

- **Schedule**: control of the timing associated with the occurrence of broadcasts and definition of the audio content to be used.
- **Playout**: based on the programming schedule, these applications ensure its execution (i.e. the time-based *playback* of audio assets).
- **User Management**: definition of the different users (authors and editors) in the station in order to correctly control the permissions for them to execute their respective responsibilities.
- **Production**: there are tools that provide what is called “live assist” or “on-air support”, which are designed to support authors during the production of their audio content. There are also audio editing tools that may be used.
- **Digital Asset Management (DAM)**: refers to the tasks and decisions surrounding the cataloguing, storage and retrieval of digital media assets (in particular, audio assets) associated with the programming of the station.

Interviews with individuals responsible for 8 radio stations from 7 countries were conducted. The purpose of said interviews was to determine how each station works, the sort of technological applications they use and to identify where possible information system deficiencies may exist. As a result of this process, the following characterizing aspects were found to adequately define the stations’ operations, and consequently their information system needs:

- **Type of broadcast content**: there are two types of contents that are used by stations in their broadcasts: *music playlists* and *author programs*. **Playlists** are lists of audio assets that can be automatically generated or created by humans, but in the context of radio stations they are typically computer-generated according to pre-defined criteria. **Author programs** are produced by the station’s authors, and are characterized by a higher (or total) degree control of their creators over their audio elements.

\(^1\) Small audio pieces produced for their specific use in intermissions.
• **Production environment:** radio stations typically base their content production in a physical location—their studio. It is a *closed* environment in which it is easier to control and overview the status of the station's audio assets. Contrastingly, there are stations that do not have a single source for their content, as they operate in a more decentralized manner. Authors deliver content to the station either from the studio or external sources, in what can be called a *mixed* production environment. This situation entails a more difficult challenge from a management perspective.

• **Financial resources:** the level of financial resources available to the station shape many of its decisions and options, including the studio and IT equipment it can afford, as well as the software systems that it is able to deploy.

• **Human resources:** due to their nature and objectives, as well as financial resources, radio stations vary in the level of human resources they have available to achieve their activities. When they have *high* human resources, stations can have more individuals responsible for controlling and overviewing their programming schedule and performing other tasks. The opposite situation exists when stations have *low* human resources, making them inherently more dependent on the capabilities of their technological and software infrastructure.

There are many software systems to support the activity of radio stations. Most of them focus in the *schedule* and *playout* applications. Since that is also the focus of this work, those are the ones that were studied. A survey of these systems was made, including both Open Source and proprietary categories. They vary in the features they provide and each have their own advantages and disadvantages. Nonetheless, they share the main use case for these application: the definition of start/end times associated with audio assets to assure their stipulated playback at the correct moment.

The surveyed systems were evaluated in terms of functional characteristics but also as to their adequacy according to the needs of the aforementioned types of stations. What was found is that there is a misalignment between the features they provide and what is necessary for a specific group of radio stations: those that mostly focus on *author programs*, working within *mixed production environments* and afflicted by *low human and financial resources*. The need to address these shortcomings is what motivates the work presented in this document.

### 1.2 Dissertation Statement

The main objective of the present work was to prove that it is possible to develop a system that answers the needs of radio stations that emphasize *author programs*, operating within *mixed environments* with *low human and financial resources*.

To accomplish this, other main goals were established, such as the remote administration of a radio station schedule through a clear and easy to use interface; the ability to recover from human failures, assuring the continuity of the station's broadcasting signal.

Moreover, it is important to take into account to the particularities of different stations. Thus, it is necessary to create a modular and lightweight design, that allows interoperability with other applications and services, as well a flexibility of choice for the different underlying technological components.
Finally, due to the fact that the targeted type of radio stations are lacking in financial resources, this system cannot be based on a commercial model. To take advantage of the communities that surround such stations and their specific needs, the system should be released under an Open Source license. This decision will hopefully entail further developments in the future as well as facilitate the needs for customization that may arise.

1.3 Dissertation Structure

This document is organized in seven chapters, including the present one. Chapter 2 provides the basic context to understand a radio stations’ activity and its main concepts. A survey of the related work in the field of radio station software is presented, followed by a comparison of their features and limitations.

Next, the problem tackled by this work and the argument for its necessity is described in Chapter 3. The case for a system such as the one discussed in this document is established by first introducing a set of aspects that define how a radio station operates and what type of Information System needs it has. The existing software options are contrasted with the needs of a group of radio stations, reaching the conclusion that there is fundamental misalignment between what currently exists and what that type of stations need. Based on the identified needs and problems, this work’s objectives and system requirements are presented.

The proposed solution to achieve the established goals is described in Chapter 4. This description includes the most important aspects to the system, such as its architectural style, users and components. Each of the system’s components is explained in terms of code structure, dynamic processes and interfaces. In Chapter 5, the implementation details of the developed work are discussed. These details include an overview of the chosen technology, a deployment scheme, implemented web resources and user interface.

A qualitative evaluation of the implemented prototype was performed with a key group of users, and is presented in Chapter 6. Finally, Chapter 7 concludes this document by discussing the main contributions of this work and future developments for it.
2 Background and Related Work

2.1 Introduction

The intent of this Chapter is to introduce the operational context of radio stations, through the definition of key concepts in this business (Sections 2.2 and 2.3). Following these topics, the main types of applications used by radio stations are described in Section 2.4. Continuing this topic, an overview of the state-of-the-art of software applications for radio stations is presented in Section 2.5, namely those related with the management of the programming schedule and its broadcasting signal. A comparison and discussion of the features of said applications is available in Section 2.6. Finally, in order to support the work presented later in this document, some metadata schemas associated with media broadcasting are summarized in Section 2.7.

2.2 Key Concepts in Radio Stations

Before discussing the current state-of-the-art of radio station software applications, it is important to establish the meaning of the terms that will be used throughout this document. Also noteworthy is the fact that this is not a field that has been explored in typical scientific research; it has been tackled as a software engineering problem, which accounts for the lack of scientific citations in this chapter. This Section provides a general description of a radio station while also defining the main concepts that will be used throughout this document.

- **Radio Station**: Organization of individuals with the goal of creating a broadcast *signal* based on a programming *schedule*;
- **Schedule**: Set of rules that define the dates and times at which programs are to be broadcasted. The *schedule* is composed of *programs* which in turn have many *broadcasts*. In-between broadcasts there usually are *intermissions*, populated by advertisement and/or promotional *spots*;
- **Program**: Also called a *show*, a *program* is a group of broadcasts (defined below) that are usually authored by the same author(s) and have a common theme. A program may occur only one time (as a special broadcast) or many times (defined by time recurrence rules);
- **Broadcast**: This term usually has two uses. The first, is to represent the signal that a station produces. The second, is to represent a particular instance or occurrence of a radio program. To avoid confusion, this document will henceforth use the term *signal* for the first use (e.g., “a station’s signal”), and *broadcast* for the second (e.g., “a program’s broadcast”);
- **Author**: Refers to a person that is responsible for producing a radio program.
- **Editor**: Refers to a person that is responsible for the definition of the station schedule, configuration of necessary broadcast parameters, as well as ensuring the quality of the content to be broadcasted.
- **Intermission**: Time interval between two different broadcasts or within segments of the same broadcast.
- **Spot/Jingle**: Small sound piece used in intermissions for advertisement, promotional or informational purposes.
- **Playlist**: Group of sound pieces that must be played, either sequentially or at random.
2.3 Description of a Programming Schedule

As previously stated, the core activity for a radio station is to produce a broadcast signal based on a programming schedule. Exactly what type of content is in a programming schedule varies from station to station, but using broad definitions it is possible to specify its contents, accommodating the needs for most stations.

Thus, in order to better understand what constitutes a radio station schedule, a Concept map [Nov98, MMJ94] is presented in Figure 2.1.

![Concept map of a radio station schedule.](image-url)

- **Broadcasts**: main elements of the schedule. They have temporal rules that define their occurrences. These rules define the duration and recurrence of a program’s broadcasts (e.g. “Sundays from 9:00 PM to 10:30 PM” or “Every 3rd Wednesday of the month from 8 AM to 9 AM”).
  - Pre-recorded: broadcasts produced before their transmission time.
    - **Playlist**: programming time based on lists of songs that are either sequentially or randomly selected.
    - **Author**: programs created by authors in the radio station.
  - Live: these broadcasts, as their name indicates, are transmitted live, without prior production or recording.
    - **Internal**: live broadcasts that are produced by the station (those made in the studio or special broadcasts from other locations). They are recorded and stored for possible use in future broadcasts.
    - **External**: broadcasts whose content is provided by another station’s Broadcast signal.
  - **Repetitions**: broadcasts usually have *repetitions*, which, as the name implies, are instances where said broadcasts’ contents are replayed. In the case of pre-recorded broadcasts, a repetition is simply another playback of the original content, while for live shows a recording of the original broadcast is used.

- **Spots**:
  - **Fixed-time**: spots broadcasted at specific times and days of the week.
– **Variable-time**: spots that are broadcasted at certain timeframes instead of specific times (e.g., a date and hour range such as "weekdays from 6 AM to 3 PM").

### 2.4 Typical Radio Station Applications

Applications to support a radio station can be divided according to the following functional groups:

- **Schedule**: Control the times of the station’s programming. Must be able to accept time recurrence rules for programs that repeat over time, as well as one-time programs.

- **Playout**: Using the pre-defined schedule, the system must fetch and play the audio sources. It is also the responsibility of the *playout* system to handle the broadcast parameters (how the audio should processed and transmitted).

- **User Management**: Define which users are *editors*, i.e., those that can control the programming schedule and broadcast parameters. It is also necessary to add, remove and edit users, and their association to programs as *authors*.

- **Production**: There are tools that provide what is called “live assist” or “on-air support”. These are designed to help the show’s presenters during a live transmission (e.g., knowing which song is coming next, how much time is left, choosing a jingle/sound effect and similar operations). These functionalities will not be considered for comparison purposes, as they are not particularly important for all types of radio stations.

- **Digital Asset Management (DAM)**: Generally speaking, DAM refers to the tasks and decisions surrounding the cataloguing, storage and retrieval of digital media assets (e.g., photographs, audio, video, text). In the context of a radio station, it is necessary to have control over the audio content produced, indicating its author(s) and cataloguing it by program, production date, broadcast date and other metadata fields.

There are currently other features being used by radio stations which are based upon the information provided by the aforementioned components. Some examples include:

- **Electronic Program Guide (EPG)**: service that provides users with a guide to the station’s programming, usually allowing listeners to navigate, select and discover content by time, title, program, genre and similar criteria.

- **Podcasting**: distribution of the station’s content using syndication feeds such as RSS or Atom. Users subscribe to the feeds that interest them the most and automatically receive their contents as they are published.

- **Notifications**: in the context of traditional media broadcasting, content is transmitted at specific times. Many media providers provide services that allow users to subscribe to a given program and receive notifications to remind them of the scheduled broadcasting time or other relevant information regarding that program.
2.5 Survey of Radio Station Applications

This Section presents an overview of different software solutions currently used by radio stations, summarizing and comparing their features. Although there are packages that integrate all or some of the functional areas already presented, most of them focus on the schedule and playout areas, since these are the most important to radio stations. Those two areas will be given more emphasis in the survey, as they ultimately are the focus of this work. This sort of applications are usually called schedulers or radio automation systems; throughout this Section, these terms will be used interchangeably. In order to provide further insight, two radio stations, Rádio Universitária do Algarve\(^1\) (RUA) and Rádio Zero\(^2\), were asked to provide feedback on their use of some of these systems.

For each item, a general description is presented, as well as a table indicating which of the defined functional areas it supports. The features that will be taken into account in the study were chosen due to their relevance to ensure a typical radio broadcast signal, as well as their usefulness for the management of a station. They are:

- **Playlist definition**—the ability to define lists of songs (or other audio content) to be broadcasted;
- **Support for different playlist modes/types**—there is a need for music radio stations to be able to create various playlist types, according to time of the day, music genre, etc.;
- **Handling of jingles, spots and time announcements**—necessary for advertisements and other promotional material;
- **Live shows broadcast**—it is important for most stations to be able to broadcast live shows;
- **Scheduling of pre-recorded programs/events**—the ability to define the broadcast of audio at certain dates and times (either for a single occurrence or at recurrent occasions such as “every two weeks”, “monthly”, etc.);
- **Notifications in case of problems**—important for small radio stations where it is not possible to have someone always present to deal with potential problems;
- **Extensibility through APIs**—the possibility to develop plug-ins or add-ons that enhance the capabilities of the software to better suit the needs of the station;
- **Remote administration facilities**—needed for radios where there is not always someone present to handle its broadcast signal;
- **Silence detection**—audio broadcasts sometimes fail and silence detection technology is necessary to automatically recover from those situations;
- **Logging of broadcasted content**—regulations in many countries require the recording and registration of broadcasted content (this feature refers to text logging since it is trivial to record the audio of the broadcast);
- **Security measures**—in collaborative environments where there are various users in the station, it is important for the scheduling system to have access control mechanisms that prevent unauthorized changes to the broadcast.

\(^1\)http://rua.pt/

\(^2\)http://www.radiozero.pt/
2.5.1 Proprietary Software

There are many proprietary radio software packages available. The price range is very wide, going from free to well over a thousand Euro. Most of these packages are very feature-complete in terms of the needs for most radios. They are desktop software, meaning that they're installed on and managed within an Operating System (mainly Microsoft Windows\(^3\)). Some of the most widely used software systems of this kind are described in the present Section.

2.5.1.1 Mar4Win

Mar4Win is developed by AEQ\(^4\) and is deployed at many different professional radios, such as RNE in Spain and RDP and RUA in Portugal\(^5\). Most recently, AEQ introduced a new version of its automation software, named MAR4Suite Pro, which brings a renovated UI, uses Microsoft SQL Server as the database management system, as well as other new modules.

This package is structured in different modules, each responsible for a particular function: MAR4News (news editing); MAR4Remote (automatic voice recording through telephone lines); MAR4Edit, MAR4Multitrack (audio editing with support for multiple tracks); MAR4Schedule (playlist scheduling); MAR4Studio (live support for radio announcers, as they make their program); MAR4List (playlist generation) and many others.

Although it has a very powerful feature set, there were complaints at RUA related to the quality of the UI and some of the limitations imposed by the system, particularly in the playlist creation and generation area. For example, the number of tags that a song can have is limited, and so is the number of songs for a playlist. This means that users have to take the system into account when they're deciding what should be broadcasted (and many times affecting it).

Some of these problems might be solved in the most recent version, which was not available for analysis. AEQ was contacted for pricing information, but did not reply in time.

Table 2.1: Supported functional areas for Mar4Win

<table>
<thead>
<tr>
<th>Area</th>
<th>Available</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schedule</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Playout</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>DAM</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>User Management</td>
<td>No</td>
<td>It has a simple login system and does not associate users with programs</td>
</tr>
<tr>
<td>Production Support</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

\(^3\)http://www.microsoft.com/windows/
\(^4\)http://www.aeq.eu/
2.5.1.2 Jazler

This software is mainly targeted at professional radios. Jazler\(^6\) — much like the software described in the previous subsection (MAR4Win) — has existed for a long time, and as such, it has gradually developed many features. As a result of this, it is deployed in many stations around the world.

Beyond the feature list that has been defined, Jazler provides advanced (and varied) functions like automatic song database backups; automatic playlist creation; advanced spots/sweepers/jingles database; audio compressor for the main output; RDS support; "virtual" speakers to introduce songs; rebroadcast support from other sources and many others. The Jazler Radio packages are available in two versions: RadioStar (Euro 1200) and SimplePack (Euro 780).

<table>
<thead>
<tr>
<th>Area</th>
<th>Available</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schedule</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Playout</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>DAM</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>User Management</td>
<td>No</td>
<td>Similar to Mar4Win</td>
</tr>
<tr>
<td>Production Support</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

2.5.1.3 Raduga

Raduga comes in five different versions (all for the Windows OS), covering different feature needs and prices (from Euro 99 to Euro 1049 without VAT)\(^7\). For a radio station, the most adequate versions are the Full Automation, Universal Radio and Developer editions. Raduga is deployed in over 100 radios worldwide\(^8\).

Raduga’s interface is mainly targeted at playlist management, while support for programs and jingles is secondary (the UI is less accessible). According to the Programming Department Director of Rádio Zero, the user interface of this software was at times confusing and difficult to operate (causing the station to abandon the use of this software). There also were some stability issues that were related to how Raduga handles scheduled events (they are run as a separate thread from the one controlling the main playlist), which sometimes caused overlapping of audio files (two files being played at the same time).

<table>
<thead>
<tr>
<th>Area</th>
<th>Available</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schedule</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Playout</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>DAM</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>User Management</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Production Support</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

\(^6\)http://www.jazler.com/

\(^7\)http://www.wolosoft.com/en/raduga/

2.5.1.4  

ZaraRadio

ZaraRadio is a free software package with only one version (for Windows), although its source is not public. In terms of features, it is similar to Raduga, but it presents a simpler interface, also following the single-package paradigm. This means that all of the functions in the system are included and accessed in the same application.

The software is very capable and easy-to-use —those being the main reasons (besides its cost) cited for its adoption at Rádio Zero. It is used by many radios worldwide and is quite stable. The fact that it is desktop-based (only available for the Windows platform) and that it does not feature any remote administration facilities are its biggest downsides, from the perspective followed in this study.

Table 2.4: Supported functional areas for ZaraRadio

<table>
<thead>
<tr>
<th>Area</th>
<th>Available</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schedule</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Playout</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>DAM</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>User Management</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Production Support</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

2.5.2  Open Source Software

Although the documentation is not always updated or complete, all of these projects provide information on their architecture and main use cases, which makes them easier to describe than the proprietary ones. Most of these solutions, similar to the previously discussed proprietary software, also have some production support, in which the software is designed to assist the radio-maker during his/her show. In addition to this, some sort of automated playlist creation or selection is usually supported. The main features and architectures of these applications will be discussed in the present section.

2.5.2.1  Reboot.fm

This system was developed by Bootlab (Berlin, Germany) in order to fit the needs that arose when Reboot.fm radio station was operating. It is divided in three components: Playout, Scheduler and Archive.

Archive

The archive is the Open Meta Archive (OMA) developed by Thomax Kaulmann. It can be used to store different sorts of media like audio, text, images and video. Its permission system is simple: it has registered and unregistered users. Registered users can edit both media and its metadata.

There are three ways to import data into the archive: reading content from the local file system and using the ID of the radio program in order to read the metadata from the scheduler; a multi-platform application that

---

9 http://www.zararadio.com/
10 http://www.bootlab.org/
11 http://www.radia.fm/partners/reboot/
12 http://oma.sourceforge.net/
uploads the media and its metadata; and a simple 2-step process involving an HTTP upload and metadata input by the user. OMA supports XML exporting for the data based on templates. It also has an XML-RPC interface, and originally intended to support the OAI interface but this is currently not documented.

Scheduler

The Scheduler is a collection of packages that run on top of the Open Source web application framework OpenACS\(^\text{13}\). This module handles both the information about radio programs (time recurrences, metadata for the program and its broadcasts) and the public website for the radio.

Users are divided into different roles:

- **All users**: can post to the forums and have read access to all the public parts of the website.
- **Contributors**: these users are the one that actually make radio programs (they contribute to them). In addition to what all users are allowed to do, this type of users can edit the metadata for the shows to which they’ve been associated (previous and upcoming broadcasts, program description, etc.). They cannot however change the times for their program.
- **Editors**: these users define the schedule for the radio, within time “containers” to which they have been assigned. Containers are editorial spaces within which editors can operate (e.g. “Nightly Hours” that ranges from 8 PM to 11 PM). Editors are also responsible for associating contributors to their programs.
- **Meta Editors**: these users define the containers and attribute editors to them. They can also perform all of the editorial tasks in any container.

Playout

The Playout module is a collection of scripts whose purpose is to handle the broadcast of the station. It operates in two modes: for live transmissions and for scheduled shows. For live transmissions, there is an encoder at the studio and a decoder at the antenna or Internet streamer. For scheduled shows, it uses an iCalendar [DS98] file provided by the Scheduler, which gives the information on which programs must be played. This uses a customized version of Ices\(^\text{14}\) and Icecast\(^\text{15}\) that can interface with the Jack audio connection kit\(^\text{16}\).

The most differentiating aspect of this software compared to the ones presented until now is the fact that it is web-based, allowing for full control of the radio via a standard web browser.

Development for this software has stopped, and the OpenACS community seems to be lagging behind other web frameworks. Reboot.fm has also stopped its broadcast. Rádio Zero uses this platform as its web presence but uses neither the Archive nor the Playout modules.

2.5.2.2 Campcaster

Campcaster\(^\text{17}\) was originally developed through a grant from the Open Society Institute\(^\text{18}\). It has been developed by an international group of developers and designers led by the Media Development Loan Fund\(^\text{19}\) and

---

\(^{13}\)http://www.openacs.org/

\(^{14}\)http://www.icecast.org/ices.php

\(^{15}\)http://www.icecast.org/

\(^{16}\)http://www.jackaudio.org/

\(^{17}\)http://www.campware.org/en/camp/campcaster_news/

\(^{18}\)http://www.soros.org/

\(^{19}\)http://www.mdlf.org/
Table 2.5: Supported functional areas for Reboot.fm

<table>
<thead>
<tr>
<th>Area</th>
<th>Available</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schedule</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Playout</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>DAM</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>User Management</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Production Support</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

its Center for Advanced Media Prague (CAMP). According to data published on their website, 7 radio stations are using Campcaster (or parts of it) to manage the stations’ needs for automation.

The software is divided in different modules: Studio, Web, and Storage Server. It also provides various APIs via XML-RPC.

**Studio**

Campcaster Studio is a desktop application whose main focus is to assist live shows with basic support for automated scheduling (for which the web interface is more robust). The main tasks that this module covers are:

- Live shows: when in Live Mode, songs are found through the search facility or the scratchpad (window that shows the most recently used audio files). It is easy to then add the songs to the Live Mode window and to rearrange them at will.
- Creating playlists: similar to the Live Mode, but it is possible to save playlists for future use and to set effects such as fade-in and fade-out between songs.
- Playlist scheduling: it is possible to select a date and time in which the playlist is to be broadcasted.
- Importing audio files from disk.
- Editing audio metadata.

**Web**

Campcaster Web was designed to easily manage the automated playout for a radio station. Its interface features functions to accomplish this task, such as: a panel that shows the currently playing song and remaining time; a scratchpad area; user management; audio search (by keyword and browsing); content scheduling through day, week and month views. It is also possible to configure a Network Hub in which a group of radios can share content.

**Storage Server**

The Storage Server for Campcaster provides the necessary functions to handle the archive for the audio clips used by the system. It is possible to add, remove and edit the files stored in the archive. The content is accessible and searchable by users through the Studio and Web interfaces.

The focus on tracks and playlists that Campcaster presents can be limiting when scheduling pre-recorded shows, since the interface tools are not adequately designed for that task.
Table 2.6: Supported functional areas for Campcaster

<table>
<thead>
<tr>
<th>Area</th>
<th>Available</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schedule</td>
<td>Yes</td>
<td>Simple one-time scheduling of playlists</td>
</tr>
<tr>
<td>Playout</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>DAM</td>
<td>Yes</td>
<td>Only as individual songs</td>
</tr>
<tr>
<td>User Management</td>
<td>No</td>
<td>Simple login system without association to programs</td>
</tr>
<tr>
<td>Production Support</td>
<td>Yes</td>
<td>Basic support for some tasks</td>
</tr>
</tbody>
</table>

2.5.2.3 Rivendell

Rivendell\(^{20}\) is developed by Salem Radio Labs, a division of Salem Communications Corporation\(^{21}\) and offers a complete solution in terms of acquisition, management, scheduling and playout of content. Many online and AM/FM radios use this software. It is mostly for a GNU/Linux\(^{22}\) environment, but some applications are implemented for Windows.

As it is designed to be flexible in terms of the physical deployment of its components, communication between them is done via TCP/IP and an audio control protocol developed for this system, described in its website\(^{23}\).

It supports lossless formats (PCM 16\(^{24}\) and MPEG Layer 2\(^{25}\)) but only through certain sound cards from AudioScience Inc.\(^{26}\). Support for other audio sources is achieved through the Jack Audio Connection Kit.

The system components are RDAdmin, RDLogEdit, RDLibrary, RDAirPlay, RDCatch and RDLogManager.

- **RDAdmin**: configuration utility to manage users, groups, services, hosts and reports. It can also backup and restore the system DB.
- **RDLogEdit**: this module serves as a configuration tool for radio programs logging.
- **RDLibrary**: supports the acquisition (CD ripping, audio recording, manual uploading of files) and management the audio assets in the station.
- **RDAirPlay**: assists the playout during live or “on air” situations.
- **RDCatch**: automatic audio recorder and event scheduler designed to simultaneously capture multiple audio sources.
- **RDLogManager**: integrates the various music scheduling systems and other audio traffic configured for the system. Through this application is possible to configure clocks (one hour schedule), events and grids (weekly schedule). It is also possible generate logs and manage reports.

*Rivendell* is a powerful tool, as can be seen by its feature set. On the other hand, when taking into account the interoperability (as an important attribute defined at the outset of this survey), it is possible to conclude that

\(^{20}\)http://www.rivendellaudio.org/
\(^{21}\)http://www.salem.cc/
\(^{22}\)http://www.linux.org/
\(^{25}\)http://www.chiariglione.org/mpeg/standards/mpeg-2/mpeg-2.htm
\(^{26}\)http://www.audioscience.com/
by using its own protocol and data definition specifications, a studio-only and workstation-based architecture, makes interoperability with other systems very difficult to attain.

Table 2.7: Supported functional areas for Rivendell

<table>
<thead>
<tr>
<th>Area</th>
<th>Available</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schedule</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Playout</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>DAM</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>User Management</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Production Support</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

2.5.2.4 Soma Suite

Soma Suite\(^{27}\) is a radio automation tool mainly developed by Andrea Marchesini. It follows a client-server architecture in which the behavior of the server daemon (SomaD) is defined by two XML configuration files that are pretty flexible. One configuration file is for the regular programming for the radio and the other is for the spots/jingles.

Remote administration is achieved in Soma Suite thanks to its architecture and a C library (libsoma) that allows full control of the functions in this software.

Configuration can be done manually (by editing the text configuration files) or through a multi-platform desktop application named SomaX. This application also assists inside the studio, notifying the remaining time for the currently playing item and providing easy access to songs in the local file system or at some server.

Even though it is very capable software, one of the problems in Soma comes from the usability front. The graphical configuration tools and the terminology used are many times confusing. The documentation for the project is also not extensive, which makes it harder to learn how to properly use it. This isn’t a real problem in the long term, because despite of having a somehow steep learning curve, the stability and functions provided by the software are shown to be more than adequate by the number of stations that use it.

Table 2.8: Supported functional areas for SomaSuite

<table>
<thead>
<tr>
<th>Area</th>
<th>Available</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schedule</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Playout</td>
<td>Yes</td>
<td>Requires an external media player</td>
</tr>
<tr>
<td>DAM</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>User Management</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Production Support</td>
<td>Yes</td>
<td>Provides a very simple tool</td>
</tr>
</tbody>
</table>

\(^{27}\)http://www.somasuite.org/
2.5.2.5 Savonet

Savonet\(^{28}\) is a tool developed by Samuel Mimram and a group of French developers, and has a pretty unique approach to radio automation. It is written in OCaml\(^{29}\) and it defines its own scripting language (called Liquidsoap) that was specially conceived to describe audio workflows.

This software does not feature a GUI or many of the pre-defined operation modes, nor does it intend to (as stated in the project page). Its aim is to provide a reliable framework within which is possible to create many different solutions, customized for each radio. The system is essentially a tool to control audio sources, being able to support complex arrangements.

Liquidsoap has an extensive API that allows the handling of various playout situations (audio output to Icecast, Jack, sound card, etc.), fallbacks in case of problems and usage of different audio sources (audio input from the sound card, local files or remotely accessible via HTTP, FTP, SMB, etc.). There is even the possibility to use Last.fm\(^{30}\) streams as input sources.

Other interesting features in the API are related to its audio processing capabilities (audio normalization, wave generation, sound filters and many others). This sort of flexible audio handling is unique for this software in comparison to the ones previously described in this paper.

Because the main objective for this software is not to provide a final solution to most users, it is basically a command-line interface tool and a programming language, which hardly classifies it in the same category as the previous products studied.

As distributed, only technically savvy users (with computer programming know-how) would be able to use it. It is nonetheless very interesting as a possible resource to be reused and extended by other software packages.

<table>
<thead>
<tr>
<th>Area</th>
<th>Available</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schedule</td>
<td>Yes</td>
<td>Basic support for daily or weekly repetitions. Very difficult to change for most users.</td>
</tr>
<tr>
<td>Playout</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>DAM</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>User Management</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Production Support</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

2.6 Systems Comparison

There are many differences in the way each system implements its most basic features, as there are also some limitations and workarounds —how these comply to the particular requirements of a station can only be decided by those responsible for it. This comparison is intended as a first look into these systems. An

\(^{28}\)http://savonet.sourceforge.net/
\(^{29}\)http://ocaml.org/
\(^{30}\)http://www.last.fm/
extensive analysis would be too difficult for a completely generalist exercise due to the many types of needs for radio station automation.

With the exception of Campcaster and Reboot.fm, all of the software packages presented require the use of desktop software, and in most cases are tied to a particular platform. This situation makes the remote administration possibilities (when supported) more limited, as the user is required to install the software in the computer in which he wishes to work remotely (and it can be incompatible).

Table 2.10 shows a feature comparison between the proprietary solutions studied. Similarly, Table 2.11 compares Open Source solutions. It is important to note that even though a certain system does not provide a given feature, it is many times possible to work around it or extend the application, if the software supports it.

### Table 2.10: Feature list for proprietary schedulers

<table>
<thead>
<tr>
<th>Feature</th>
<th>Mar4Win</th>
<th>Jazler</th>
<th>Raduga</th>
<th>ZaraRadio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Playlist definition</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Different playlist modes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Jingles/Spots</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Time announcements</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Live shows broadcast</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Events/Programs scheduling</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Problem notifications</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Extensibility through APIs</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Remote administration</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Silence detection</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Broadcasted content logging</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Security measures</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

It is possible to see that proprietary packages support most of the features for the defined list. For the most basic features, ZaraRadio provides a good price/value ratio, since it is stable and free. Raduga supports some more features (including the possibility to develop plug-ins for it), but as its price (for the more complete versions of the software) is close to Jazler’s, the latter may be the stronger option.

Curiously, features that would be very important for radios that cannot afford a constant presence of someone at the studio or that need some flexibility, see their options limited in this regard. Remote administration is possible in the more expensive schedulers but only via desktop-based software. Notifications for users in case of problems is lacking in all but one of the alternatives presented.

All in all, these are very different software solutions but they provide (essentially) the same basic capabilities, so a choice between them will have to depend on the criteria and requirements defined by the radio station. The Open Source projects presented are varied and each obviously has its strengths and weaknesses.

Rivendell has a steep learning curve and not much documentation online, making it a difficult system to adopt at first, although there are many stations using it due to its feature-set (but some are dependent on having certain high-end audio cards).
Reboot.fm and Campcaster have the advantage of having most or all of the administrative functions as web applications, making them easy to remotely manage. Even though it is an interesting platform, Reboot.fm’s development has stopped. Similarly, Campcaster development has been slowing down as well as the growth of its community. Further negative aspects for Campcaster are its sluggish interface and limited scheduling features.

Soma Suite is a very strong product and has an active community, visible by the number of stations using the system and the participation level in its mailing lists. SomaX’s interface design is somewhat similar to Raduga and ZaraRadio —main focus on tracks, playlists, currently playing song, etc. The fact that it has a flexible configuration mechanism and ways to extend or to alter its behavior is very positive.

Savonet appeared as the most different option from the selection. Lacking any graphical interface and typical associated functions, it focuses on delivering a very capable API (and scripting language) to define the way the station’s broadcast is to be output. As previously noted, this software cannot be considered so much a solution but rather a tool —on top of which others can be built.

The main advantage of this group of schedulers over the proprietary ones stems from the fact that the former are Open Source: they can be adapted to particular needs. The features that are not available can be implemented either directly (within the source code itself) or through the APIs that these projects provide. The other big advantage is cost, which is very important for a small radio station.

### 2.7 Metadata Schemas

Metadata consists of data structures used to discuss other data structures. Metadata augments the values of information (or data) with additional properties that explain its meaning, organization, and other characteristics of interest in our models [GM05].

There are metadata schemes (both open and proprietary) to represent the media broadcasting business in the market. The purpose of this section is to present a brief summary of some of these formats, in order to provide additional support to the work presented later in this document.
2.7.1 TV-Anytime

The **TV-Anytime Forum**\(^\text{31}\) is an association of organizations that gathered to develop specifications to enable audio-visual and other services based on mass-market high volume digital storage in consumer platforms, also referred to as “local storage”. The specifications covered four main objectives: enabling applications to exploit local persistent storage in consumer electronic platforms; striving for network independency in content delivery; interoperability and integration of systems involved; creating security structures to defend the interests of stakeholders.

In terms of metadata, many aspects were covered: elements for EPGs, user preferences and habits, as well as content segmentation for editing and viewing purposes. These metadata specifications may be used in conjunction with existing industry standards such as MPEG-7\(^\text{32}\) in order to create working applications [PS00]. Nevertheless, there are not many implementations of the metadata specifications, and only the British Broadcasting Corporation\(^\text{33}\) (BBC) has released a Java Open Source library for EPG information creation\(^\text{34}\). **TV-Anytime** specifications are open, but they are also very large, which implies a large effort must be deployed for their implementation.

2.7.2 Dublin Core

The **Dublin Core Metadata Initiative**\(^\text{35}\) (DCMI) is an organization dedicated to promoting the widespread adoption of interoperable metadata standards and developing specialized metadata vocabularies for describing resources that enable more intelligent information discovery systems. The **Dublin Core Metadata Element Set** (DCMES) deals with cross-domain information resource description, and it is published as an ISO standard [ISO03].

DCMES is widely used as the basic format to catalogue digital media assets such as audio, video, and documents. It has a basic core of 15 elements such as **Title**, **Creator**, **Subject**, **Description**, **Contributor**, **Date**, **Type** and **Format**, among others.

2.7.3 EBU Core Metadata Set for Radio Archives

The **European Broadcasting Union**\(^\text{36}\) (EBU) is an association of radio and TV broadcasters from from 56 countries. Members usually are large publicly-owned broadcasters or private companies with public missions. The EBU has produced metadata schemas associated with the TV and Radio broadcasting businesses\(^\text{37}\).

This organization has many specifications, many of them quite large and complex. In one of its technical documents, EBU defines a set of metadata which is adapted for use in radio archives, but which is aligned both with the main metadata standards of the broadcasting industry and with DCMES [Uni01].

---

\(^\text{31}\)http://www.tv-anytime.org/
\(^\text{32}\)http://www.mpeg-industry.com/
\(^\text{33}\)http://www.bbc.co.uk/
\(^\text{34}\)http://www.bbc.co.uk/opensource/projects/tv_anytime_api/
\(^\text{35}\)http://dublincore.org/
\(^\text{36}\)http://www.ebu.ch/
\(^\text{37}\)http://tech.ebu.ch/publications
The Exchange Broadcast Binary and Metadata Format (XBMF) was created as part of the StreamOnThe-Fly project [ASMP+05, KMPS05]. Its purpose is to create an easily extensible file and metadata format for exchanging broadcasts between various broadcast content management systems. XMBF was designed to reflect the needs of smaller radio broadcasters, for which the industry metadata formats being proposed (e.g. TV-Anytime, MPEG-7) are too large and cumbersome [sot].

XBMF can be used for transfer and archival of radio programs coupled with metadata. This is a simple container format, in which metadata (in XML format), content files, and other associated files can be packaged together. Technically, an XBMF file is a compressed directory structure, where naming conventions and placement of files identify the data they convey. Implementation of software for handling XBMF files is straightforward, and existing Open Source software can be used. The metadata schema for XBMF can be seen as an application profile of Dublin Core (DC) for radios, constructed from DC elements and the StreamOnTheFly element set.
3 Problem Statement

3.1 Introduction

After reviewing the current landscape of radio software applications in Chapter 2, the present one will explain why certain types of radio stations are not properly served by those systems. The motivation for this work is that there is a fundamental misalignment between the existing systems and the characteristics of the aforementioned set of stations. This argument will be presented over the next sections.

A set of aspects that were found to identify the way radio stations operate are introduced in Section 3.2. Rádio Zero, the main case study for this work, is presented and described in Section 3.3. The problem addressed by this work is stated in Section 3.4, followed by its objectives (Section 3.5), and requirements (Section 3.7).

3.2 Categorization of Radio Stations

The topics discussed in this section are the result of semi-structured interviews with individuals in charge of several radio stations (the basic script used as a basis for that process is available in Appendix A). The purpose of the interviews was to have an overall understanding of how different stations work, what sort of software they use and what their needs are.

The following stations were interviewed during a meeting of the Radia Network\(^1\) in Berlin, Germany: Resonance FM\(^2\) (London, England); Tilos Radio\(^3\) (Budapest, Hungary); Radio Grenouille\(^4\) (Marseille, France); ORANGE 94.0\(^5\) (Vienna, Austria); Kanal 103\(^6\) (Skopje, Macedonia). Another member of Radia, free103point9\(^7\) (New York City, USA) was contacted via email. More extensive interviews were conducted at the facilities of Rádio Universitária do Algarve (RUA)\(^8\) (Faro, Portugal) and Rádio Zero\(^9\) (Lisbon, Portugal).

As a result of those interviews, it was possible to identify the main aspects in the operation of a station that lead to different information systems needs:

- **Type of broadcast content**

  In a broad sense, there are two types of contents that are used by stations in their broadcasts: *music playlists and author programs*. *Music playlists* are automatically generated by software systems containing databases of songs that use pre-defined criteria to select them. Under this approach, presenters have minimal to no control of what gets played. “Author programs” refer to a class of programs in which the people who make them (the “authors”), are in complete control of their production.

  As the programming depends more on author programs, the station’s management finds it necessary to be able to specify which people are authors, control if they fulfill their responsibilities and if their

\(^1\)http://www.radia.fm/
\(^2\)http://resonancefm.com/
\(^3\)http://tilos.hu/
\(^4\)http://www.grenouille888.org/
\(^5\)http://o94.at/
\(^6\)http://www.kanal103.com.mk/
\(^7\)http://www.free103point9.org/
\(^8\)http://www.rua.pt/
\(^9\)http://www.radiozero.pt/
work has appropriate quality. This means that as the focus of the station goes from songs and creating playlists to managing many programs and their authors, the underlying system must support a different range of functions.

• **Production environment**

Most stations use their studio as their sole production facility. Authors simply use the software systems at their disposal in the studio, and thus the station gets its content as soon as it is produced. To facilitate nomenclature, this environment will be called “closed”

On the other hand, some stations have a heterogenous production environment. The contents for their broadcasts can be completely produced inside the studio, partially or fully outside the studio. This environment will be referred to as “mixed”.

The difference here lies in how the station performs the acquisition of its content. If the station works within a “closed” environments, failures in the acquisition process are virtually non-existent. As a contrast, in an “mixed” environment, this process is prone to problems due to the use of different networks, communication delays as well as either technological or human failures. Thus, this case requires a system that provides a way for the station management to monitor, review and audit this process in order to know where problems exist or arose in the past.

• **Financial Resources**

The amount of financial resources at the disposal of the station mold its choices, and in particular, the software systems that it is able to deploy.

• **Human Resources**

Financial resources also influence the station at the human resources level, just as much as their objectives and organizational culture. Radio stations with sufficient budgets have paid staff and contributors whose responsibilities include dealing with and solving any potential problems that may affect the station’s signal. These staffers are present in the studio and can easily control the station management software when necessary.

Stations with lower financial resources are based in volunteer or part-time work, making them more susceptible to human and technical problems. Having low human resources means that it is not possible to respond to every possible issue. This context needs a system that supports some sort of automated recovery from errors and that is able to notify interested users in these cases. Furthermore, under these conditions, it is of utmost importance to have remote administration features so that those responsible for the management of the station can perform their duties without being physically present in the studio and more effectively respond to problems.

An additional problem that results from not having a reliable or sizable staff is that the station is unable to maintain the configuration, training and oversight efforts that some systems require.

Another aspect that may influence the choice of software for the station but does not necessarily characterize the type of station, is the use that intermissions are given. When stations use their signal as a revenue-generating activity, their intermissions are filled with advertisements and promotional spots. These spots have associated rules (combining time-based attributes and desired occurrence frequency) that define when they should be broadcasted (examples include: “Everyday between 8:00 AM and 11:00 AM”, “On week-
days between 7:00 PM and 9:00 PM: to be played a maximum of three times; On weekends at any hour: to be played a minimum of two times and a maximum of four times”). As a contrast, if stations do not have a sophisticated use of their intermissions, they are able to use systems that support simpler or no rules of this kind (which have the advantages of having simpler user interfaces and a much lower cost).

After taking these aspects into account, it is possible to understand how different combinations yield very different IS requirements.

### 3.3 Organizational Description of Rádio Zero

To fully comprehend Information Systems, it is necessary to understand the broader Organization, Management, and Information Technology dimensions of systems and their power to provide solutions to challenges and problems in the business environment [KJ04].

The Organization dimension of IS involves the organization's hierarchy, functional specialties, business processes and culture. The Management dimension involves leadership, strategy and management behavior. Finally, the Technology dimension relates to the computer hardware, software, data management and networking/telecommunications technology.

As it is the main case study in which the present work is based, Rádio Zero is described according to these three dimensions in this section.

#### 3.3.1 Organizational Structure

Rádio Zero is a non-commercial radio station focused on “author programs”. It has been broadcasting 24 hours per day since 2004, then named Rádio Interna do IST (RIIST). As of March 2006, it took its current name. Its broadcast is currently only available on the Internet, although the process to obtain an FM license in the Lisbon area has been initiated.

People within the station are organized in two categories: collaborators and members. Collaborators are the people involved in the various Departments or Administration of the station. Members are both authors and collaborators.

![Figure 3.1: Overall organization of Rádio Zero.](image)

The Administration of the station is composed of three members: President, Secretary and Treasurer.
Their responsibilities are mostly bureaucratic—dealing with bookkeeping, redacting official documents, repre-
senting Collaborators in official meetings with other entities, among other similar duties.

The main tasks for each Department are:

**Programming Department**

1. Receive and evaluate proposals for new programs;
2. Decide and set up the schedule for each broadcasting season;
3. Supervise content and quality of produced programs;
4. Receive, organize and prepare broadcasts delivered by authors to remote repositories;
5. Decide the studio occupation schedule based on the availability of authors and other temporal con-
   straints;
6. Handle exceptional requests to use (or free) the studio during the broadcasting season;
7. Register assiduousness of authors;
8. Responsibility for special programs whose authors are not members of the station (due to collaboration
   with international partners or one-time broadcasts);
9. Manage the record library of the station (cataloguing and organization);
10. Contact record labels to obtain copies of new albums and schedule interviews;
11. Contact event promoters to get press passes for members interested in attending an event;

**Technical and Logistics Department**

1. Manage the IT infrastructure of the station (servers, software and services);
2. Archive programs after they are no longer necessary for broadcast;
3. Set up—and tune when necessary—the studio equipment (mixing table, microphones, record players,
   amplifiers, sound compressors, cables, etc.);
4. Document equipment and IT configuration;
5. Handle requests from members to use station resources for external assignments (microphones and
   recorders for field interviews or other special coverage);

**Marketing**

1. Writing official documents, press releases and editorial section of the station's newsletter;
2. Promote programs in the station’s website and newsletter;
3. Contact external entities for partnerships and development of special projects;
4. Find and contact other organizations to develop relationships to ensure financing for projects;
5. Conceive and coordinate promotional campaigns for the station (e.g., reception of proposals for new
   programs, launch of a new broadcasting season, special events);

**3.3.2 Management**

Decisions concerning the strategy of the station are taken through votes from collaborators. Individual col-
aborators or departments may take initiatives and start projects, but ultimately, the decision to go ahead with
them lies with a majority of votes from all collaborators.
In order to increase the number of collaborators and fill positions that have been left vacant, this group is devising outreach strategies to members, listeners and outside elements.

3.3.3 Technology

This subsection describes the technology currently in use by Rádio Zero to achieve its operational and management objectives.

The station has three servers, two running on Debian GNU/Linux\(^{10}\) and one on Microsoft Windows. For its studio workstation, the station uses Microsoft Windows as its OS. There is an additional editing workstation that also runs Debian GNU/Linux. As part of its technological strategy, Rádio Zero intends to achieve the goal of having all of its IT infrastructure based on Free and Open Source Software.

The technology used for the operational standpoint will be described according to the categories previously defined in Section 2.4.

- **Schedule and Playout:** for these functions, ZaraRadio is used. Collaborators of the Programming Department are responsible for the initial configuration, updating the system’s settings and updating

- **Digital Asset Management (DAM):** there is no DAM solution currently being used by Rádio Zero. The most recent broadcasts made for each program are kept in one PC in the studio; when it is necessary to free some hard drive space, a member of the Technical and Logistics Department performs a backup of past broadcasts to DVDs.

- **User Management:** the list of collaborators and members is maintained in an internal wiki. The Programming Department is responsible for associating members to their respective programs and keeping this list updated.

- **Production:** broadcasts are produced using various tools, since authors have varying levels of proficiency in the area of audio production. Authors may use a combination of an audio player (e.g., WinAmp\(^{11}\)) and an audio recording utility (e.g., PowerMP3 Recorder\(^{12}\)). They also have available audio editing software, namely Adobe Audition\(^{13}\) and Audacity\(^{14}\).

On the management side, communication between collaborators is made via a mailing list. Since most decisions are taken through votes and prior communication, email has become a truly important tool in this organization. The aforementioned internal wiki is also the place where collaborative documentation and projects are developed.

3.3.4 Broadcast Business Processes

Another aspect of the organization are its processes. To further describe the work at Rádio Zero, the most important business processes associated with broadcasts are presented in this section. They are depicted using Business Process Modeling Notation (BPMN [Gro]) and accompanied by a textual description.

\(^{10}\)http://www.debian.org/
\(^{11}\)http://www.winamp.com/
\(^{12}\)http://www.coolsoft.com/mp3rec.htm
\(^{13}\)http://www.adobe.com/products/audition/
\(^{14}\)http://audacity.sourceforge.net/
Although the processes reflect Rádio Zero’s operation, interviews showed that other stations based in author programming, with similar organizational structures and constraints, share the same basic procedures.

### 3.3.4.1 Pre-recorded broadcasts

Pre-recorded broadcasts are those that are produced before their broadcast time. The process starts when a new broadcast is needed (because its broadcasting time is coming up), as can be seen in Figure 3.2.

![Business Process for the first transmission of pre-recorded broadcasts](image)

Figure 3.2: Business Process for the first transmission of pre-recorded broadcasts.

The authors produce the program using the tools they are most comfortable with. Production techniques used by authors are varied and are not standardized, so this part of the process will not be discussed.

After production comes the “delivery” activity (Figure 3.3). It is in this activity that the station acquires the content produced by its authors. Rádio Zero currently has two methods to receive produced broadcasts. When they are made inside the studio, there is a local repository in the filesystem of the workstation that runs ZaraRadio. Authors must name the files according to pre-established rules or otherwise the scheduler will not play them. When made partially of fully outside the studio, authors may access an FTP server where they upload their program as a single file. In this case, it is necessary to notify the editors that the needed content has been delivered, so they can move the file to the local repository (if they are at the studio, the operation is trivial; if they are outside of it, they must access the workstation through Windows Remote Desktop\(^{15}\)).

Once the broadcast is in place, editors must validate its contents (Figure 3.4). This is due to the fact that many authors are inexperienced or forget to comply with certain quality parameters. For instance, broadcasts should have the exact length that was made available for them in the schedule—avoiding cuts or blanks in the broadcast signal. Authors should also ensure that the audio file has been normalized\(^{16}\) and that sound glitches are not present. This type of audio processing must be done by humans, because no software is

---


\(^{16}\)Audio normalization is the process of increasing (or decreasing) the amplitude of an audio signal. It is a practice at Rádio Zero to set the highest peak in the waveform to -0.3 dB
currently capable of always accurately resolving these problems. Since many authors do not perform this type of checks, editors must routinely ensure this is done.

If there are no problems or if they are easily treatable, the broadcast is considered "valid" and the corrections that may be needed are applied. If the recording is in no condition to be broadcasted, a substitute is found, the problem is registered in the internal wiki for future reference and the author is contacted, to inform him/her of what went wrong.

Figure 3.4: “Validation” activity for pre-recorded broadcasts.
The “validation” phase (Figure 3.4), when performed, ensures that there is adequate content for transmission. If for some reason this phase is not executed, problems at the time of the broadcast may arise.

As the recording is now prepared, it is broadcasted (i.e., played and transmitted) during the “playout” activity.

![Diagram](image)

**Figure 3.5:** “Post-processing” activity for pre-recorded broadcasts.

Finally, after being transmitted, the broadcast enters the “post-processing” activity (Figure 3.5). Currently, Rádio Zero archives its past broadcasts to DVDs and to a hard drive in the station’s main server. This process is done manually whenever someone from the Technical Department is available to do so. It is in this phase that broadcasts remain until they are finally archived.

Repetitions of pre-recorded broadcasts follow a similar process (Figure 3.6) to the one that first-time broadcasts do (since they are both pre-recorded). The difference lies that at the outset of the process, a check is performed to verify if the first-time broadcast was successful. If it was, the process continues normally. If there were problems with the first-time broadcast, the “validation” activity is once again performed to ensure that there is valid content for broadcast. After this, the process continues.

### 3.3.4.2 Live broadcasts

Live broadcasts can be of two types (internal or external), as described in Section 2.3. Because only internal live broadcasts are produced by the radio station, those are the ones that will be addressed in this Section.
Production of live broadcasts is inherently different than pre-recorded ones (Figure 3.7). It starts at the same time of the broadcast’s transmission. Three activities are performed in parallel: “Studio production”, “Recording” and “Playout”. “Studio production” refers to the making of the program itself, at the studio or at some special event venue, and similarly to pre-recorded broadcasts, it involves authors using the tools they feel most comfortable with. Since it is a live show, “recording” is done at the same time of broadcast. “Playout” is done by software, taking the audio that is being created to the station’s signal.

After its transmission, the “author post-processing” activity is performed (Figure 3.8). At Rádio Zero, the
most common corrections to the recording of the broadcast is to normalize the audio signal and set the
program metadata. Other minor operations may be applied as the author desires. This activity leaves the
recording ready to be “delivered” to the station, following the same process as for pre-recorded shows.

![Figure 3.8: “Author post-processing” activity for live broadcasts.](image)

Repetitions of live shows are somewhat similar to first-time pre-recorded broadcasts with some minor
differences (Figure 3.9). The show is produced at the same time it is transmitted, and the “delivery” phase is
performed after the live broadcast ends (an operational requirement at this station). Afterwards, the broadcast
goes through the same “validation”, “playout” and “post-processing” that were previously described.

![Figure 3.9: Business process for repetitions of live broadcasts.](image)

3.4 Problem

Rádio Zero has been growing its programming almost constantly while maintaining a practically flat (and low)
number of people responsible for them (Figure 3.10).

The time required to the editors by their responsibilities (Section 3.3.1) increases with the number of pro-
grams. Many of these tasks are repetitive and could be carried out by automated processes. This statement
is further supported by considering the overall life-cycle and business processes for broadcasts presented in
Section 3.3.4.

Different software systems for radio stations were described and compared on a functional basis in Section
2.4. Table 3.1 presents a listing of those systems, now including the variables deemed to be important to
categorize different radio stations in Section 3.2. For each given variable, a classification that reflects how a
system corresponds to it is presented. This classification is not naturally binary; it is the result of a judgment
if a particular system is better suited for a given type of radio station.

The “Content” variable reflects if the system is mainly designed to manage songs and audio files or authors
contributing programs.

The “Environment” can be “closed” or “mixed”; if the system is able to acquire content from different
sources and to function without the need to run completely in the studio, it is considered as supporting a “mixed” environment, and a “closed” one otherwise.

Dividing systems by the amount of Human Resources (HR) they require for their operation is more problematic. All systems obviously need configuration and oversight from their users. For the purpose of this comparison, two values will be possible: “Sporadic” and “Regular”. A “Regular” need for HR is when the system requires, beyond the normal programming schedule definition, that some administrative user interacts with it for all (or most) new broadcasts produced by authors. A “Sporadic” need for HR is considered to be when the system only requires configuration and interaction at the scheduling level—users contributing new broadcasts are supported by the system without the need for some administrator to intervene. Some systems have workarounds to the need for constant configuration, but due to the importance of this process, this comparison will only consider those that truly support this.

In the “Financial Resources” category, the threshold for what is considered a low endowment will be a maximum of 1000 Euro. This value is not completely arbitrary, since there are many small radio stations that cannot afford more than this in software licenses, specially at the beginning of their project.

Table 3.1: Alignment of Software Systems with Radio Station Types

<table>
<thead>
<tr>
<th>System</th>
<th>Content</th>
<th>Environment</th>
<th>Human Resources</th>
<th>Financial Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Playlist</td>
<td>Author</td>
<td>Closed</td>
<td>Mixed</td>
</tr>
<tr>
<td>Mar4Win</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Jazer</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Raduga</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>ZaraRadio</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Reboot.fm</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Campcaster</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Rivendell</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>SomaSuite</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Savonet</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
Rádio Zero is, according to the parameters established for comparison purposes, a radio station focused on author programs, operating in a mixed environment, with low human and financial resources. At the present moment, it has 66 programs and 97 authors, corresponding to 74 weekly first-time broadcasts (which follow the processes described in the previous section). With only 3 volunteer individuals responsible for the management of the programming schedule, the sustainability of this situation can be considered fragile.

This is not a self-imposed problem by the station due to poor management choices. Inspecting Table 3.1, it is observable that for a station with the same characteristics as Rádio Zero, there is only one adequate system among those surveyed (Reboot.fm). Nevertheless, as previously noted in Sections 2.5.2 and 2.6, development for this system has stopped, it was built on a now inactive technological platform and there are many deficiencies in terms of features and user interface, rendering it inappropriate.

There is thus a void in the offer of information systems for this niche of the radio station market. This fault causes most of the described processes to be carried out manually, at the detriment of efficiency of the work time made available by its collaborators. Interviews also showed that this situation is not unique to Rádio Zero, thus making it a relevant problem to explore and attempt to solve.

3.5 Objectives

The main objectives of the present work are to deal with the problem stated in the previous section, proving that it is possible to create a system that answers the needs of collaborative radio stations focused on author programs, operating within mixed environments with low human and financial resources. Taking into account the referred goals for the system, the remaining objectives are derived:

- Remote administration of a radio station schedule—stations with low human resources are not able to have a constant presence of editorial staff to ensure the configuration and management of its applications, which is why they need easily accessible interfaces for their systems.
- Modular and lightweight design—not all stations require the same set of functional components, making it necessary to have a clear separation of concerns between the modules that are part of the design.
- Recover from human failures (by either editors or authors), and assure the continuity of the station’s signal—the combination of low human resources and mixed production environments causes operational errors for the station (e.g. authors not delivering their produced contents to the station, editors not specifying programming times, communication errors that prevent the retrieval of contents). As such, a solution for these types of stations must consider the aforementioned issues in order to guarantee that empty programming timeframes do not occur.
- Provide a way for editors to audit the station’s operation and authors’ performance—a mixed environment and low human resources also entail that there is not a constant oversight of all the activities performed by the system actors. As a consequence, editors need to be able to access the information recorded by the system in order to know how authors are performing their expected responsibilities.
- Clear and easy to use interface—individuals involved in radio stations usually are not technologically savvy, and need applications that do not present steep learning curves.
- Design for interoperability with other applications and services—there are many systems for different
functional areas of radio stations; the design of the solution must be guided by the need to share its information, facilitating its integration with external systems.

- Allow flexibility of choice for the different technological components—separating the informational and business aspects of the system from its implementation choices, assures the possibility of adopting different underlying technologies, depending of the requirements of the station.
- Develop an Open Source software system—radio stations with low financial resources benefit from freely accessible software, and are often willing to participate in its customization and further development.

### 3.6 Use Cases

A use case is a description of the possible sequences of interactions between the system under discussion and its external actors, related to a particular goal [Coc00]. This section shows the UML diagrams of the most important use cases, along with an overall description for each group of cases. It is important to note that use cases have a code before their names in order to identify them and map them to the functional requirements listed in Section 3.7.1.

Figure 3.11 shows a set of use cases associated with the general system administration. These are supporting tasks, to manage necessary information used in other parts of the system. Editors need to be able to define the sources for live broadcasts (which are Internet URIs), as well as the locations of audio asset repositories that authors may use to deliver their productions. Also, both authors and editors are able to create playlists with the audio assets that exist in the system.

![Figure 3.11: General administration.](image)

A very important function of the system is to handle the programming schedule for the station. This entails different activities (the most relevant ones are listed in Figure 3.12). It should be possible to create broadcasts
as a bulk or as single instances. Additionally, the system must be able to associate an audio structure to broadcasts, in order for them to be useful to the playout application. Said structure should be able to contain various audio assets (either as single files or grouped in a playlist), and also allow for broadcasts to have intermissions. Finally, a feature that is relevant for some of the targeted type of radio stations, is the handling of promotional audio spots, played between broadcasts. This should include the definition of temporal and frequency rules associated with audio spots, so the system can determine the contents of intermissions.

Programs are an important entity for this system. They have many broadcasts and one or more authors, and also constitute an entry point to the station’s schedule. Among the features necessary to manage programs are (Figure 3.13): their creation in the system (with a name and description), navigating through broadcasts in time, visualizing the status for each of their broadcasts (to know if their contents are available to the system or not), and communicating with the programs’ authors.

In line with the prior set of functions, editors need to manage the authors of the broadcasts in the schedule (Figure 3.14). This means that they need to associate existing users in the system to a particular program or broadcast, thus allowing them to contribute content either to all of the broadcasts of a program or only to specific instances.

Authors need to perform a set of actions to fulfill their responsibilities, such as the ones shown in Figure 3.15. These actions include delivering content for their broadcasts to the system. Authors are responsible for the contents of their programs and broadcasts, and thus need to notify the system of their location as they are produced, so the system is able to retrieve those audio assets and use them in the broadcast. Beyond these functions, authors also need to see the status for their broadcasts, communicate with other users (editors or authors) and request (or free) the use of studio time.
System administrators are responsible for some high level tasks, such as the initial setup of the system or the creation and removal of users. Another function they perform is the attribution of the “editor” role to other users in the system (Figure 3.16). Given the nature of the application and its features, having the “editor” role amounts to almost being an administrator, since these users will then be able to manage the schedule, users, programs and other previously described tasks.

Finally, unregistered users (which can also be called listeners), may use the system to access information about the station's programming (Figure 3.17). Thus, among functions they can perform are: listing all programs in the schedule; view their information (description, authors, upcoming broadcasts); navigating by date through the programming schedule and accessing particular broadcasts to view their synopsis and contents. These users may also be other systems, that use the programming schedule and associated data for informational purposes, notifications or other uses.
3.7 Requirements

Based upon the aforementioned objectives and problem definition, the list of main functional and non-functional requirements is presented in Sections 3.7.1 and 3.7.2. They were derived from the stipulated objectives for this work as well as additional specific needs established by the stakeholders of the system. Also, the functional requirements are supported by the use cases described in Section 3.6.
3.7.1 Functional Requirements

Functional requirements specify the software functionality that the developers must build into the product to enable users to accomplish their tasks, thereby satisfying the business requirements [Wie03]. The requirements listed in this section are not fully detailed but serve as a reference of the main features in the system. Each requirement also indicates the use case code that supports it and the figure where it is available.

1. Editors must be able to define the station’s programming schedule (Figure 3.12, UC34);
2. Add, edit and remove individual broadcasts (Figure 3.12: UC35, UC37);
3. Add, edit and remove programs (Figure 3.13: UC29, UC30);
4. Add, edit and remove program authors (Figure 3.14: UC9, UC10, UC11);
5. Add, edit and remove editors (Figure 3.16: UC15, UC16);
6. Authors must be able to add, edit and remove audio assets that may be associated with their programs (Figure 3.15: UC1);
7. Add, edit and remove playlists (Figure 3.11: UC22, UC23, UC24);
8. The system must provide communication features between users (Figure 3.15: UC8);
9. Add, edit and remove content repositories for broadcasts to be delivered and retrieved (Figure 3.11: UC19, UC20, UC21);
10. Authors and editors must be able to overview the overall status of the system: upcoming broadcasts, if their contents are available, when delivery of those contents occurred, etc. (Figure 3.13: UC31, UC32—Figure 3.15: UC2, UC3);
11. Review historical assiduousness of an author (Figure 3.13: UC31, UC32);
12. Review a program’s broadcasting history (Figure 3.17: UC25, UC26, UC27, UC28);
13. Provide a playout solution for the station’s programming schedule;
3.7.2 Non-Functional Requirements

Non-functional requirements include different types of demands on the system, such as performance goals, quality attributes (e.g. usability, portability, integrity), as well as design and implementation constraints [Wie03]. The following requirements were defined by the needs and strategies of the business in question.

1. The system must assure the continuity of the broadcast signal when editors do not specify the programming content or when authors fail to deliver it to the station;
2. The system must be able to run on low-end hardware;
3. The system must be built on an active technological platform;
4. The system must be released under an Open Source license;
5. The system must use standard and open technologies to avoid future lock-ins;
6. The technology must have a low barrier of entry to facilitate future development of the system;
4 Proposed Solution

4.1 Introduction

This chapter presents a system proposal to fulfill the objectives and requirements established in Chapter 3. The present section and its subsections provides an overview of issues that are important to the designed solution. First, a domain model for this business is presented in Section 4.1.1. Then, desirable attributes for the system and its software architectural style are presented in Section 4.1.2. The chosen style is further described in Section 4.1.3, while the reasons for its use are described in Section 4.1.4.

Following these initial topics, the presentation of the proposed solution begins by addressing the followed methodology (Section 4.2), overall system architecture (Section 4.3) and system components (Sections 4.4 and 4.5). Each component is documented by presenting the most important topics according to three aspects: code structure, interfaces provided and dynamic processes relevant to detail their operation.

4.1.1 Domain Model

A domain model is useful as a way to become more familiarized with the business concepts and their relationships. The domain model for the system is illustrated in Figure 4.1, and it is based on the requirements and analysis of the problem.

![Domain Model Diagram]

Figure 4.1: Domain Model.

4.1.2 Software Architectural Styles

An architectural style is a coordinated set of architectural constraints that restrict the roles/features of architectural elements and the allowed relationships among those elements within any architecture that conforms
to that style [Fie00a]. This term should not be confused with software architecture, defined as the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [BCK03].

The basic requirements for this system stipulate that it must be remotely accessible and configurable, which entails that it must be network-based. There are various software architectural styles for network applications, which are classified by their type (data-flow, replication, hierarchical, mobile code and peer-to-peer [Fie00b]). Amongst these styles there are many classical and well-known ones, such as Pipe and Filter (data-flow), Replicated Repository (replication), Client-Server (hierarchical), Code on Demand (mobile code) and Distributed Objects (peer-to-peer). Each has benefits and disadvantages for different types of applications.

Based on the objectives and requirements established for this work, the desirable attributes for the system and the style guiding its architecture were defined:

- **Encourage loosely-coupled components**: for a modular design with a clear separation of concerns.
- **Support for a service architecture**: to provide integration capabilities with third-party systems and to make available the system’s information to the outside.
- **Scalability**: so the service architecture is able to grow if demand for it increases.
- **Simplicity**: present a low barrier of entry to developers, expediting the processes of customization and future developments of the system.
- **Extensibility**: in order to be able to include additional functions if business requirements change.
- **Support for standards**: to avoid technological lock-ins and to also make easier the integration with other applications and services.

### 4.1.3 Representational State Transfer (REST)

Representational State Transfer (or REST) is a software architectural style for distributed hypermedia systems [Fie00a]. The largest and most well-known implementation of REST is the World Wide Web [FT02].

This style is characterized by its division of application state and functionality into resources. Each resource is uniquely addressable using a universal syntax, to be used in hypermedia links. A constrained set of well-defined operations and content types form a uniform interface for resources to transfer state between server and client. The protocol used by REST is client-server, stateless, cacheable and layered.

REST ignores the details of component implementation and protocol syntax, instead focusing on the roles and constraints upon components, connectors and data that define the basis of the web architecture. The following list summarizes the concepts in REST, divided by architectural elements, as described in [Fie00a].

- **Data**
  - **Resource**: the key abstraction in REST. It represents any information that can be named: a document or image, a temporal service (e.g. “today’s weather”), a collection of other resources, a non-virtual object (e.g. a car or person), and so on.
  - **Resource Identifier**: necessary to identify a particular resource involved in an interaction between components.
- **Representation**: REST components perform actions on a resource by using a representation to capture the current or intended state of that resource and transferring that representation between components. Thus, in broad terms, a *representation* is a sequence of bytes, plus *representation metadata* to describe those bytes. The data format of a representation is called a *media type* [FB96].

- **Resource metadata**: information about the resource that is not specific to the supplied representation.

- **Control data**: defines the purpose of a message between components, such as the action being requested or the meaning of a response. It can also be used to parameterize requests and override the default behavior of some connecting elements (e.g. the cache’s behavior).

- **Connectors**: these elements encapsulate the activities of accessing resources and transferring resource representations through an abstract interface for communication between components.
  - **Client**: initiates the communication by making a request.
  - **Server**: listens to connections and replies to requests in order to supply access to its services.
  - **Cache**: can be located on the interface to a client or server connector in order to save cacheable responses to current interactions so that they can be reused for later requested interactions.
  - **Resolver**: translates partial or complete resource identifiers into the network address information needed to establish an inter-component connection (e.g. an URI that includes a DNS domain that is resolved into an IP address).
  - **Tunnel**: relays communication across a connection boundary, such as a firewall or lower-level network gateway.

- **Components**
  - **User Agent**: uses a client connector to initiate a request and becomes the ultimate recipient of the response (e.g. a Web browser).
  - **Origin Server**: uses a server connector to govern the namespace for a requested resource. It is the definitive source for representations of its resources and must be the ultimate recipient of any request that intends to modify the value of its resources.
  - **Proxy**: intermediary selected by a client to provide interface encapsulation of other services, data translation, performance enhancement or security protection.
  - **Gateway**: performs the same actions of a proxy, but this intermediary is imposed by the network or origin server and thus the client does not have the choice to use it or not (unlike a proxy).

### 4.1.4 Architectural Style

The coordinated set of architectural constraints that REST lays out attempt to minimize latency and network communication, while at the same time maximizing the independence and scalability of component implementations. Its most popular implementation (the WWW), uses the HTTP protocol, which is simple, lightweight and widespread. Hence, REST matches the overall desirable attributes for the system and architectural style established in Section 4.1.2, making it the style to be explored by this work.
By adopting this architectural style, the most important informational entities for this business can be exposed as web resources and the HTTP protocol provides the necessary verbs to manipulate them. This approach is unique for radio software applications, since it opens the business domain to interaction with the outside world via standard WWW technologies. Dealing with aligned systems (e.g. the DAM application proposed in [Cos08]) in this manner, makes integration more easily achievable, thus providing a modular support to the different radio station applicational needs.

4.2 Methodology

This section presents a brief overview of the overall methodology used during the development of the proposed solution. It was based on standard requirements elicitation and software development approaches used for small groups of stakeholders that are available for regular contact to clarify business aspects and functional needs [KS98, Wie03, BA04].

Work towards this solution was developed in an iterative and incremental manner. Initial informal interviews were conducted, with email follow-ups (as referred in Section 3.2). The visits and interviews performed at RUA and Rádio Zero allowed to have a better understanding of the current radio station software panorama, as well as realizing the problems faced by these stations.

Design and implementation were also developed over time, incorporating some aspects of agile programming methodologies. Modeling and coding problems were consulted with the dissertation advisor. Features were shown to Rádio Zero’s management as they were developed. From the gathered feedback, corrections were continually made. Finally, a small qualitative evaluation process was performed, in order to capture what users perceived to be the most positive and negative aspects resulting from this system.

4.3 Architecture

4.3.1 Components

The established requirements can be divided into two categories: those related with the schedule and its information; and those that are related with the execution of that schedule.

Thus, to accomplish the objectives for this work, two modules were devised: Scheduler and Playout. Because the system is an exploration of the REST architectural style, these modules communicate using standard HTTP interfaces, as shown in Figure 4.2.

It makes sense to separate these concerns into different modules, because the use of both may not be necessary to all stations—e.g., scheduling data may be used just for informational purposes (without directly controlling the broadcast signal); to assist in the management of the station (keeping a record of the programming contents and times, registering when an author delivers content to the station).

The characteristics of REST and this separation ensures that the system is based on loosely coupled components, a design choice that carries important benefits for current and future development. These benefits include a greater flexibility in deployment strategies; more technological options to implement low-level features; and the possibility of modifying or enhancing certain features without major changes or disruptions.
to other systems.

The Scheduler and Playout modules will be respectively detailed in Sections 4.4 and 4.5. Those sections document both modules by describing the most important topics according to three aspects: code structure, interfaces provided (as web resources) and dynamic processes that help to understand their operation.

Figure 4.2: Components and the interfaces that connect them.

4.3.2 Actors

The necessary actor classes are not complex. As the requirements stipulate, they can be divided in four levels (as shown in Figure 4.3), listed here in increasing levels of privileges:

- **User**: corresponds to an user that is not registered in the system or one that is registered but without any privileges attributed to him/her;
- **Author**: a registered user that is the author of a program or some broadcast(s).
- **Editor**: user responsible for setting up the programming schedule, adding/removing authors, supervising the execution of the schedule, etc.;
- **Administrator**: super-user that is able to control all functionalities in the system.

Figure 4.3: System actors.

4.4 Scheduler

4.4.1 Description

The Scheduler module is concerned will all the information surrounding the radio station schedule. It holds the most important entities for the station's business: programs, authors, broadcasts, intermissions, etc. It is through this module’s services that users will interact with the system, making it the most important in this design.
4.4.2 Structure

Audio contents are made available by authors at varying times before broadcasting time, and sometimes do not even become available by that time. It is a requirement that this system supports and properly records a broadcast’s life-cycle to inform the station’s management and to ensure that programming voids do not occur.

Consequently, from the perspective of this component, it is necessary to record important attributes for a broadcast, such as its start and end time, the program to which it belongs, its authors, the audio assets it needs—how they are organized in the broadcast and where they are located. These and other aspects relevant to the code structure of this component are described over the next sections.

To provide its features, this module is designed as a web application that provides the interface for users, as well as services for the service’s clients. At a higher-level perspective, this corresponds to a single component. Model classes to support the business logic for this module are available in Figure B.1. The class model for this application is designed to achieve the main objectives defined in Chapter 3. As presented in Section 2.7, there are various metadata formats that could be integrated into the domain of this application; this is particularly true for the data elements in XBMF, which are geared for radio stations working within simpler contexts, just as the ones focused in this work. However, in order to facilitate its implementation, and to create a working proof-of-concept application, it was decided that this solution should not be focused on immediate compatibility with the referred formats and instead focus on its main objectives.

4.4.2.1 Audio Assets

Audio assets refer to the station’s media contents. They represent any content that may be broadcasted. The design defines three types: singles, spots, and playlists. Singles and Spots both represent individual assets, the former are used as content for broadcasts, the latter are used for intermissions. Playlists are groups of ordered audio assets (individual) or other playlists.

Additionally, singles can be associated with live sources, thus representing an audio asset that is first broadcasted live and is to be input to the system afterwards.

Since these assets are only necessary for the broadcast playout, they are not directly used by the Scheduler module. These mechanisms will be addressed in Sections 4.4.4.2 and 4.4.4.3.

Assets are associated with their authors, independently if they are used in a broadcast or not. However, this module is not responsible for permissions, proper cataloguing and other operations on audio assets: those are the activities performed within a Digital Audio Management (DAM) system and thus outside the scope of this work. Nonetheless, by keeping the record of which users are the authors of a given asset, whether and when it was used in broadcasts, the Scheduler module has the basic elements to support integration with DAM applications or services.

4.4.2.2 Broadcast Authorship

The Authorship association links the User, Program and Broadcast classes. An user may be related to a broadcast as an author in two ways: through a program, indicating when is the user responsible for the broadcasts (always or only on certain days of the week); or directly, to particular broadcasts.
With these options, it is possible to easily support the most common types of authorship (always or associated to days of the week) while also covering other cases that cannot be so simply defined (by indicating individual broadcasts). By maintaining references to their associated authorships, it is simple to access all programs and broadcasts associated with an author, as well as authors for given programs and broadcasts.

4.4.2.3 Broadcast Status

As established, it is necessary for management (and other systems) to know the status of a given broadcast. A broadcast's status depends on the status of its audio assets. By design, an audio asset is considered to be “delivered” when an author inputs to the system the URI where it can be retrieved. It is considered “available” when the asset is present at the Playout module. If a URI has not been given, the asset is considered to be “pending”.

These states translate to the status given to broadcasts (as illustrated in Figure 4.4). When a broadcast is created, if its contents are new (i.e. broadcast does not use previously existent content in the station), its state is “Pending”. If all necessary audio assets for a broadcast are delivered, then the status for the broadcast is also “delivered”. Broadcasts are considered “partially delivered” if only some assets have been input to the system. When all assets are at the Playout module, the broadcast is considered to be “available”.

At any time, a broadcast may be cancelled by the station’s editors, leading to its deletion from the system.

4.4.2.4 Broadcast Structure

There are several ways to describe the arrangement of audio elements that play sequentially\(^1\), commonly referred as playlist. Because playlists are the basis for radio broadcasts, a similar structure was used to internally represent their contents.

\(^{1}\)http://gonze.com/playlists/playlist-format-survey.html

---

Figure 4.4: Statuses of a broadcast in relation to its audio assets.
Broadcasts in this system possess a Structure, which is composed of Segments. A Segment has various attributes, such as its length, position within the broadcast and the audio asset it uses. If it refers to a playlist, a Segment may also indicate if the playlist is to be randomly or sequentially played, or the number of items that should be used. These options allow flexibility when defining the structure of broadcasts.

4.4.2.5 Broadcast Occurrence

There are patterns to record time-based information [CEF98] but the real problem associated with the programming schedule is the use of recurrence rules to define when a program’s broadcasts are transmitted. This sort of rules can be simple (e.g. “every Sunday at 8 AM”) or complex (e.g. “every third Wednesday of the month”), depending on the needs of the station.

Consequently, the approaches to the inclusion of time recurrence rules in different radio software systems have been varied. Some systems do not support them and require that all broadcasts be created individually while others support rules that allow recurrences at the week or month level. Nonetheless, there are richer specifications for these types of rules in calendaring systems, such as iCalendar [DS98], Google Calendar\(^2\) or Microsoft SharePoint\(^3\).

Due to this reason, this solution proposal does not model classes to define time-recurrence rules. It must import occurrence information from existing libraries that manipulate the aforementioned (or other) calendar formats. Thus, all broadcasts are created in the system by specifying their start/end dates and times.

4.4.2.6 Gaps

Programming gaps refer to periods of time where there are no scheduled programs. This may occur due to a fault in the editors time allotment for the programming or simply because there is nothing to be broadcasted. Since gaps directly affect the signal the station produces, they should be addressed in the domain of the system. In a given timeframe, gaps may occur between broadcasts, before the first scheduled broadcast, after the last scheduled broadcast or span the entire timeframe. Because gaps can be potentially infinite (if there are no broadcasts), they must be calculated in the context of a starting and ending date and time, not existing as persistent entities.

4.4.3 Resources

This section presents a list of web resources available at the Scheduler service. For each resource, a table is presented, containing:

- **URIs**: the URIs under which the resource is available. Depending on each resource, it may me accessible as a collection (list of resources) or as a member of that collection (single resource). The URIs may contain variables, using the following notation: /path/to/resource/{variable}.

- **Elements**: data elements that are part of representations for a single resource (member). Their data types are listed alongside, and, if they refer to resources within the system, the indication “Resource

---

\(^2\)http://code.google.com/apis/calendar/
\(^3\)http://www.microsoft.com/sharepoint/
URI” is used.

- **Parameters:** if the resource expects parameters as part of the request to access it, they are listed under this table entry.

Only data and generic information about resources is discussed. Other topics such as authentication and authorization mechanisms are addressed as implementation details, in Chapter 5.

### 4.4.3.1 Schedule

Represents the programming schedule. Lists all broadcasts and gaps for the requested timeframe. If some or none of the parameters is given, a 24-hour timeframe is assumed, starting at the time at which the request was received. This resource is unique (or singleton) and thus it does not need an ID to be accessed.

It serves as the main mechanism to access the station’s programming information. For each broadcast entry, the full resource entry is rendered, reducing the number of queries that a client would need to retrieve the same information. The Schedule resource is summarized in Table 4.1.

<table>
<thead>
<tr>
<th>URIs</th>
<th>/schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>Broadcasts</td>
</tr>
<tr>
<td></td>
<td>Gaps</td>
</tr>
<tr>
<td>Parameters</td>
<td>start (DateTime, optional)</td>
</tr>
<tr>
<td></td>
<td>end (DateTime, optional)</td>
</tr>
</tbody>
</table>

### 4.4.3.2 Programs

Represents a radio program. It shows basic information about the program, such as its name and description. There are other resources that are nested within programs, namely Broadcasts (Section 4.4.3.3) and Authors (Section 4.4.3.9). This resource is summarized in Table 4.2.

| URIs                   | /programs (collection) |
|                       | /programs/{id} (member) |
| Elements               | Name (String)          |
|                        | Description (String)   |

### 4.4.3.3 Broadcasts

Broadcasts are one of the main resources in the system. They can be accessed in three ways:

- Globally, by date (nested under Schedule resource)
- Globally, by ID
- By program and date (nested under Program resource)
Besides the main attributes (such as start/end date and time, description and program), these resources include their structure (listing segments and audio assets), as summarized in Table 4.3.

Table 4.3: Broadcast resource description

<table>
<thead>
<tr>
<th>URIs</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>/schedule/broadcasts (collection)</td>
<td>Start ( \textbf{\text{DateTime}} )</td>
</tr>
<tr>
<td>/schedule/broadcasts/{year}/{month}/{day} (collection)</td>
<td>End ( \textbf{\text{DateTime}} )</td>
</tr>
<tr>
<td>/schedule/broadcasts/{year}/{month}/{day}/{id} (member)</td>
<td>Description ( \textbf{\text{String}} )</td>
</tr>
<tr>
<td>/programs/{program-id}/broadcasts (collection)</td>
<td>Structure (including segments, audio assets and related information)</td>
</tr>
<tr>
<td>/programs/{program-id}/broadcasts/{year}/{month}/{day} (collection)</td>
<td>Program (Resource URI)</td>
</tr>
<tr>
<td>/programs/{program-id}/broadcasts/{year}/{month}/{day}/{id} (member)</td>
<td>Authors (Resource URIs)</td>
</tr>
<tr>
<td>/broadcasts (collection)</td>
<td></td>
</tr>
<tr>
<td>/broadcasts/{id} (member)</td>
<td></td>
</tr>
</tbody>
</table>

4.4.3.4 Gaps

This resource represents gaps in the programming schedule (summarized in Table 4.4). Due to the nature of gaps, they are always calculated and do not possess an identifier. They are only accessible by date, as a collection. Thus, the minimum access granularity is achieved by querying for gaps on a given day. Because gaps do not have any prescheduled content, they do not have a structure to list—that should be decided by the Playout module.

Table 4.4: Gap resource description

<table>
<thead>
<tr>
<th>URIs</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>/schedule/gaps (collection)</td>
<td>Start (optional ( \textbf{\text{DateTime}} ))</td>
</tr>
<tr>
<td>/schedule/gaps/{year}/{month}/{day} (collection)</td>
<td>End (optional ( \textbf{\text{DateTime}} ))</td>
</tr>
</tbody>
</table>

4.4.3.5 Singles

Single audio assets are nested under the /audio namespace. They each list their title, a description and its authors (corresponding to system users); if its authors are not system users (i.e., the single is an external song or audio file), the artist name is listed. In the case that a single corresponds to a live source, its URI is also listed (recall that live programming segments are treated as Internet streams). This resource also has a special method (at /audio/singles/unavailable) that lists all singles that have been recently added to the system. The Single resource is summarized in Table 4.5.

48
Table 4.5: Single resource description

<table>
<thead>
<tr>
<th>URIs</th>
<th>/audio/singles (collection)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>/audio/singles/{id} (member)</td>
</tr>
<tr>
<td></td>
<td>/audio/singles/unavailable (collection)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Elements</th>
<th>Title (String)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Artist (String)</td>
</tr>
<tr>
<td></td>
<td>Description (String)</td>
</tr>
<tr>
<td></td>
<td>Live URI (String, if available)</td>
</tr>
<tr>
<td></td>
<td>Authors (Resource URIs)</td>
</tr>
</tbody>
</table>

4.4.3.6 Spots

The Spots resource is analogous to the Singles resource, but is available at the following URIs:

- /audio/spots (collection);
- /audio/spots/{id} (member);
- /audio/spots/unavailable (collection);

4.4.3.7 Playlists

Represents a collection of single or spot audio assets defined by some user. Assets are ordered within the playlist, but this information is only considered depending on the segment that is using the playlist (it may play elements sequentially or at random). The Playlist resource is summarized in Table 4.6.

Table 4.6: Playlist resource description

<table>
<thead>
<tr>
<th>URIs</th>
<th>/audio/playlists (collection)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>/audio/playlists/{id} (member)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Elements</th>
<th>Name (String)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Singles and/or Spots (Resource URIs)</td>
</tr>
</tbody>
</table>

4.4.3.8 Editors

Represents users that are responsible for the management of the programming schedule. This resource is particularly useful to achieve better integration with other application or services that share all or some of the user classes that are available in this module. This resource is summarized in Table 4.7.

Table 4.7: Editor resource description

<table>
<thead>
<tr>
<th>URIs</th>
<th>/editors (collection)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>/editors/{id} (member)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Elements</th>
<th>Name (String)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Description (String)</td>
</tr>
<tr>
<td></td>
<td>Authors (Resource URIs)</td>
</tr>
</tbody>
</table>
4.4.3.9 Authors

Represents system users that author some of the station’s broadcasts. This resource lists the programs and broadcasts for which the given is author is responsible. It also includes the user’s resource URI, in order to be able to access its information as a user.

This resource is available under two URIs. One is global to the system, and the other is nested by program, as summarized in Table 4.8.

Table 4.8: Author resource description

<table>
<thead>
<tr>
<th>URIs</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>/authors (collection)</td>
<td>User (Resource URI)</td>
</tr>
<tr>
<td>/authors/{id} (member)</td>
<td>Programs (Resource URIs)</td>
</tr>
<tr>
<td>/programs/{program-id}/authors (collection)</td>
<td>Broadcasts (Resource URIs, listed under the corresponding program)</td>
</tr>
<tr>
<td>/programs/{program-id}/authors/{author-id}</td>
<td></td>
</tr>
</tbody>
</table>

4.4.3.10 Users

All of the system’s users are also resources, each with its own URI. Similarly to the Editors and Authors resources, availability of this information makes it easier for this system to integrate with other services, due to the use of a unique URI to identify an user. This resource is summarized in Table 4.9.

Table 4.9: User resource description

<table>
<thead>
<tr>
<th>URIs</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>/users (collection)</td>
<td>System ID (String)</td>
</tr>
<tr>
<td>/users/{id} (member)</td>
<td>Name (String)</td>
</tr>
</tbody>
</table>

4.4.3.11 User Mailboxes

Part of the problem that this system is tackling is the improvement of communications between radio station members. It is most efficient for users if they are able to use the same interface where they perform their duties to communicate with each other.

Consequently, a simple messaging scheme was included as part of this system. In order to enable future services such as alerts, message reception in different devices and formats, the messaging system is also available as a web resource. Users have “mailboxes” under which messages they receive are organized (similar to most messaging systems). Mailboxes are resources (summarized in Table 4.10), as well as their messages (Section 4.4.3.12).

There are three standard mailboxes: inbox, sentbox and trash (these names are used as IDs for the URI). This simple resource only lists the name of the mailbox and the number of messages in it. This resource does
not need to be read-write enabled (read-only should suffice), because the provided mailboxes are enough to fulfill the system requirements without introducing unnecessary complexity. The mailbox resource is only available under this design to provide information about mailboxes without having to access the messages resource (which may contain more data to download). Thus, this resource can be considered a “summary” of the mailbox messages resource, and can be extended in the future to provide other fields.

Table 4.10: User mailbox resource description

<table>
<thead>
<tr>
<th>URIs</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>/user/mailboxes (collection)</td>
<td>Name (String)</td>
</tr>
<tr>
<td>/user/mailboxes/{id} (member)</td>
<td>Number of messages (Integer)</td>
</tr>
</tbody>
</table>

4.4.3.12 User Mailbox Messages

As already described, messages are organized under mailboxes. This resource lists the standard fields associated with messages in most systems, such as subject, message body, sender, recipient(s), etc. Being a simple messaging mechanism to be used between system users, senders and recipients are referenced by their user URIs. This resource is summarized in Table 4.11.

Table 4.11: User mailbox messages resource description

<table>
<thead>
<tr>
<th>URIs</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>/user/{user-id}/mailboxes/{mailbox-id}/messages (collection)</td>
<td>Subject (String)</td>
</tr>
<tr>
<td>/user/{user-id}/mailboxes/{mailbox-id}/messages/{id} (member)</td>
<td>Sent date (DateTime)</td>
</tr>
<tr>
<td></td>
<td>Read status (Boolean)</td>
</tr>
<tr>
<td></td>
<td>Sender user (Resource URI)</td>
</tr>
<tr>
<td></td>
<td>Recipient user(s) (Resource URI(s))</td>
</tr>
<tr>
<td></td>
<td>Message (String)</td>
</tr>
</tbody>
</table>

4.4.4 Dynamic

4.4.4.1 Schedule Update

Editors are responsible the station’s programming schedule. Interviews showed that this activity is not performed often, usually only at the start of a new broadcasting season, followed by small updates through time. Defining a broadcast schedule is a time-consuming activity due to the editorial decisions involved and the need to align authors’ availability and the studio’s occupation; for these reasons, when the schedule is updated, it is already a coherent calendar. Thus, this system supports two schedule update mechanisms: individual and multiple creation of broadcasts through a calendar object. This section describes how the creation of broadcasts from a calendar is performed, since it is more interesting than the single broadcast case.
Figure 4.5 presents an interaction overview diagram that shows the schedule update scenario using a calendar. The entire activity is composed of three interactions. First, the calendar object is loaded into the system. If there are conflicts between the broadcasts that are already registered in the system and those that are to be created according to the calendar, the editor must resolve those conflicts. When there are no conflicts remaining, the system is able to accept the batch of broadcasts to create.

To set the schedule from a calendar (Figure 4.6), the editor user starts by selecting this option in the system, which shows the LoadScheduleScreen. The user selects the calendar file, the final date that should be considered from the calendar and the structure that the broadcasts being created will have (it can be chosen from a list of template structures). The ScheduleController passes the incoming contents to the
instance of the Schedule class\textsuperscript{4}, in order to compare the existing broadcasts with those that are to be created. The result is a list of broadcasts that can be created and those that present conflicts with existing ones, shown in PreviewScheduleChangeScreen.

A “conflict” occurs when there is a broadcast in the system whose contents have changed (e.g. a synopsis was added, its content was delivered, etc.) and the incoming calendar indicates a different broadcast at the same (or at an intersecting) time period. When presented with the PreviewScheduleChangeScreen (Figure 4.7), the editor must select a conflict from the list and choose which broadcast to use (by canceling the existing one or choosing the new one). This option is sent to ScheduleController, and the list is updated. After all conflicts have been resolved, the broadcast creation process can resume.

![Figure 4.7: Resolve conflicts between existing broadcasts and those being created.](image)

When the user submits the final list of broadcasts to create (Figure 4.8), the data is passed on to the ScheduleController and then on to the Schedule instance to finally perform the update on its broadcasts. The ScheduleUpdateResultScreen is then rendered to the user, showing the results of the operation.

![Figure 4.8: Confirmation of schedule update results from user.](image)

### 4.4.4.2 Audio Asset Availability

Section 4.4.2.3 described how the status of a broadcast is associated with the status of its audio assets. To determine the status of an audio asset, the system follows a simple two-step mechanism. Figure 4.9 shows an scenario in which an user (an editor in this case), wants to know a selected asset’s status and later its playing history.

The user queries the SinglesController at the Scheduler module, through the audio asset’s screen (ShowAudioAssetScreen). In this case, the asset is already registered in the system and has been delivered by its author. Because of this, the Scheduler queries the Singles resource in the Playout service to determine if

\textsuperscript{4}This class is a singleton
the asset is available to it or not. This result is then shown to the user (the possible replies are “available” and “unavailable”). If the asset had not been delivered, the query to the Playout service would not have been necessary to provide an answer to the user (the asset’s status would have been “pending”).

Finally, to show an asset’s playing history through time, the Scheduler service needs to query the PlayLogs resource in the Playout module, which is handled by PlayLogsController. Because this information is only available in the Playout service, the view element can query directly PlayLogsController, thus bypassing the SinglesController at the Scheduler.

4.4.4.3 Audio Asset Delivery

There are two ways in which an user may deliver audio assets to the system: associated with a broadcast or not associated. The followed steps are the same, and Figure 4.10 shows the latter scenario.

The author accesses the new audio asset screen, indicating the asset’s title, retrieval URI and authors. These data is sent to SinglesController, and a new instance of Single (or Spot) is created. From this moment on, the new single is listed in Scheduler’s web resources, for Playout to retrieve when necessary.

4.5 Playout

4.5.1 Description

The Playout module is concerned with the execution of the radio station schedule. It takes the programming schedule provided by the Scheduler module, interpreting and assuring that its content is broadcasted.

To fulfill these responsibilities, this module checks which audio assets are needed for the time period it requested and download those assets that it does not already possess and whose retrieval URIs are available at the Scheduler service.

Another aspect to this module’s work is that to provide the playback of audio assets according to a given schedule, it must interface with some audio player or one of the systems studied in Chapter 2. To achieve
this, it is necessary to convert the format provided by the Scheduler to one that the player system accepts.

4.5.2 Structure

More varied responsibilities lead to a different configuration of components in this module, as opposed to the single-component structure of the Scheduler. Besides the web application to communicate with outside systems, this module needs a constantly running daemon that triggers schedule updates and checks for audio asset availability. Additionally, an audio player is required, that receives its configuration through the daemon. This configuration of components is illustrated in Figure 4.11.

Sections 4.5.2.1 and 4.5.2.2 discuss some options taken, related to this module's web application code structure.

4.5.2.1 Consuming Scheduler Resources

The Playout module depends heavily on the data it obtains from the Scheduler service. To wrap the resources it consumes from that service, some classes are necessary, shown in Figure B.3. The structure reflects the one found in the Scheduler.

These classes represent the result coming from queries to the Scheduler service, and are used by local model classes to maintain consistency or retrieve information from the referred service.
4.5.2.2 Model Classes

Once again, local model classes also follow the same structure used by the Scheduler models and based on the business domain model (Figure 4.1). The model classes for the web application are available in Figure B.2.

In order to access updated information that is available at the Scheduler service, some classes, such as RadioStation and Single, reference resource classes.

Despite these similarities, there are some differences worth noting. The most glaring one is related to the logging of each asset's play time (achieved by the PlayLog class). This is necessary to fulfill audit and legal requirements in some countries, besides being useful for management purposes. The reason to keep this record in the Playout module is that even though the Scheduler defines the programming schedule, there may be variations in what actually gets played. The divergence appears due to the use of randomness when selecting audio assets by the Player, or when choosing content to fill programming gaps and intermissions.

Another difference between this module and the Scheduler is that it does not keep local records of playlists as assets, since they are only collections of singles, spots and/or other playlists and may change regularly. By accessing playlists as resources when they are referenced in the schedule, this module obtains the most updated version of the playlist and the audio assets it uses.

Because of the existence of programming gaps or content that was not made available by its author(s), this module, after obtaining the updated schedule from the Scheduler, knows when programming gaps occur. After this, it must decide which content is to be used to fill these empty spaces, according to which audio assets are available. It is for this reason that decision over which assets to use is performed at the Playout module—for increased speed and efficiency determining the availability of assets, since it is only a matter of a local database query instead of one over the web.

4.5.3 Resources

The present Section follows the same notation used in Section 4.4.3. It describes the web resources associated with this module, and presents a summary in the form of a table.

4.5.3.1 Gaps

Programming gaps are filled by this module. Consequently, it is at this service where the programming contents are known, making the Gap resource different from the one at the Scheduler. Another difference associated with gaps in this module is that they are persistent and identifiable objects; they are each individually accessible by their URI and cease to exist whenever the schedule is updated at the Scheduler, to fill a gap. This resource is summarized in Table 4.12.

4.5.3.2 Singles

This resource represents the same resources that are listed by the Scheduler. Their IDs are attributed by that module (although they should be managed by a DAM service that fulfills that role). Singles are available from
4.5.3.3 Spots

This resource is analogous to the Singles resource, but refers to spot audio assets. It is accessible at these URIs:

- /audio/spots (collection)
- /audio/spots/{id} (member)

4.5.3.4 Play Logs

Play Logs represent each instance in which an audio asset has been played. This resource is available in two ways: nested by audio asset or globally accessible at the URI top-level. Logs are accessible either as a full list or indexed by date (in the same way that broadcasts are). This resource is summarized in Table 4.14.

4.5.4 Dynamic

4.5.4.1 Programming Schedule Update

The daemon component is responsible for the update of the programming schedule (Figure 4.12 illustrates a possible scenario). It periodically sends the schedule_changed? message to the RadioStation instance. To reply, it is necessary to query the Scheduler service, obtaining the latest Schedule resource.
Table 4.14: Play logs resource description

<table>
<thead>
<tr>
<th>URIs</th>
<th>/audio/singles/{single-id}/logs/{year}/{month}/{day} (collection)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>/audio/singles/{single-id}/logs/{year}/{month}/{day}/{id} (member)</td>
</tr>
<tr>
<td></td>
<td>/audio/spots/{single-id}/logs/{year}/{month}/{day} (collection)</td>
</tr>
<tr>
<td></td>
<td>/audio/spots/{single-id}/logs/{year}/{month}/{day}/{id} (member)</td>
</tr>
<tr>
<td></td>
<td>/logs/{year}/{month}/{day} (collection)</td>
</tr>
<tr>
<td></td>
<td>/logs/{year}/{month}/{day}/{id} (member)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Elements</th>
<th>Audio Asset (Resource URI)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Play date (DateTime)</td>
</tr>
</tbody>
</table>

![Figure 4.12: Updating programming schedule information.](image)

After this step, the two latest schedules are compared. If there are differences (as in this scenario), programming gaps are updated (and filled) to reflect the most recent programming data, keeping gap information reasonably updated.

A reply message is then sent to the daemon, indicating that there are differences with the previous schedule. Consequently, it needs to obtain the most recent configuration (method described in Section 4.5.4.2) and feed it to the Player component.

Under this design, there may be time intervals in which the Gap resource does not correspond to the latest version of the programming schedule. This is not important, since that information is always updated at the Scheduler service—only the audio assets that are to be used are not known at that service, which does not pose a problem to the system’s core functions.

Another delay that is introduced by this schedule update mechanism is associated with the query interval used to the Schedule resource and other implementation details, such as network availability, response times, or the use of caching. According to the system’s stakeholders, a 20 to 30 second delay is completely acceptable, since it is very uncommon to alter the currently playing content (i.e., the case in which such a delay would apply).
4.5.4.2 Export configuration

To export the configuration format from the one that the Scheduler service provides to the one needed by the Player, this module’s class structure is used (Figure 4.13). The Schedule resource is retrieved from the Scheduler. The to_broadcast method call is issued to this object (indicating the output format) and then to its related objects (Broadcast, Structure, Gap, etc.), resulting in a text string return.

Figure 4.13: Exporting schedule configuration from Scheduler format to Player format.

This design assumes that the Player component has a textual configuration file and some mechanism to alert it that the configuration has changed, so it can read its updated information. The software system used by the Player may be implemented using a simple playlist-based music player or some of the FOSS systems studied in Chapter 2. Since it simplifies the needed software, a music scheduler that takes as input a list of audio assets and the times at which they are to be played, is the best option to implement the Player component.

4.5.4.3 Fetching Audio Assets

This module needs audio assets in order to function. Assets must be downloaded to the same machine running this module, to avoid network access or latency problems if attempting to play them directly from their source. This system provides two options for the Playout service to know which assets it must use and where to retrieve them. The first is through the Schedule resource query; the second is through a special method for the Singles resource at the Scheduler service. Each has its advantages and disadvantages.

Using the list of assets present in the Schedule resource gives information of when they are needed but the time to download them may be insufficient in certain cases. The other option, using the unavailable method, lists all new assets that have been recently added to the system; this allows the system to retrieve assets as they are added. This scenario is shown in Figure 4.14.

The update daemon calls a method at the RadioStation instance, which causes a subsequent query to the unavailable method in the Singles resource, returning a list of assets (each with their retrieval URI) that
need to be downloaded. After this step, all assets in that list that are not already locally available, are listed in the *Playout Singles* resource with status “downloading”.

The list of assets is given back to the update *daemon* which will then attempt to download those assets and when it finishes, inform the *RadioStation* instance of which are now locally available. After this final step, the assets that were successfully downloaded are listed at the *Playout* service as “available”, while the others are listed as “unavailable”.

![Figure 4.14: Fetching audio assets for playout.](image-url)
5 Implementation

5.1 Introduction

This Chapter describes an implementation of the solution proposed in Chapter 4. Section 5.2 introduces the technological choices and why they were adopted. Section 5.4 presents an overall introduction to the work that was developed, while Sections 5.5 and 5.6 respectively discuss the main issues related to the implementation of the Scheduler and Playout modules. Finally, a deployment scheme is explained in Section 5.3.

5.2 Technology

This section presents an overview of the technologies chosen as the basic infrastructure for the system. One of the main non-functional requirements for this system is the use of Free and Open Source Software, which constrains the number and type of available technological options. Over the next four sections, the main technologies that support this project are described and arguments for their use are presented.

5.2.1 Ruby

Ruby was created by Yuhiro Matsumoto and publicly released in 1995\(^1\). It is a genuine object-oriented language—everything that is manipulated is an object and the result of those manipulations are themselves objects [TFH05]. The language is characterized for being dynamic, reflective and allowing multiple programming paradigms (like object-oriented, functional and imperative). Additionally, it has a dynamic type system and automatic memory management. This is an interpreted language, i.e. the source code is compiled by an interpreter at the point of execution, instead of the traditional approach of pre-compiling into a binary format that targets a specific processor (as done in C or C++); it is part of a group of interpreted languages that have recently become more popular due to their advantages [Ous98].

5.2.2 Ruby on Rails

Ruby on Rails\(^2\) (often referred simply as Rails) is a web application framework that uses the Ruby programming language. It was originally developed by David Heinemeier Hansson as an extraction of his personal work developing web applications [THB+07].

The design of Rails was driven by a couple of key concepts: DRY and Convention over Configuration. DRY stands for Don’t Repeat Yourself\(^3\) —every piece of knowledge in a system should be expressed in just one place. The Convention over Configuration\(^4\) (CoC) principle intends to decrease the need for many of the configuration files common in various frameworks by defining a set of “rules” that, if followed, allow the programmer to work at a higher level of abstraction. Some of these “rules” include the specific locations in which source code and other files are to be placed, and naming conventions that files, classes and database

\(^{1}\)http://www.ruby-lang.org/
\(^{2}\)http://www.rubyonrails.com/
\(^{3}\)http://c2.com/cgi/wiki?DonRepeaYourself
\(^{4}\)http://softwareengineering.vazexqi.com/files/pattern.html
Tables should follow and thus ease development. Part of this principle is reflected in the directory structure that is generated when a new Rails application is created (Figure 5.1).

![Directory structure of a Rails application.](image)

**Figure 5.1:** Directory structure of a Rails application.

Rails imposes one big constraint on the applications developed using this platform: they must follow the Model-View-Controller (MVC) design pattern. This design pattern is useful for web applications [LR01], helping to separate the main concerns that such applications have to address. Figure 5.2 (adapted from [THB’07]) shows the control sequence followed when a request is posted to the application. A web page represents the interaction mechanism between users and the system. The user performs some action within the page (e.g. clicks on a link or fills and submits a form) and a request is sent the web server which forwards it to the application. A controller handles the request, and calls the appropriate method on a model object (containing the business logic to be executed). The operations may be persisted to the database and when the model is done with its duties, the controller resumes control. Using the results of the operation and other necessary information, the view is invoked, rendering a new web page to the user.

![Model-View-Controller as an architectural style for web applications.](image)

**Figure 5.2:** Model-View-Controller as an architectural style for web applications.

This framework makes use of the MVC pattern to support REST interfaces. In Rails, web resources are associated with specific controller classes. Each controller has actions, i.e. methods corresponding to an operation on the resource. By implementing a set of actions that match the Create, Read, Update and Delete (CRUD) operations, the framework can map them to HTTP verbs and thus provide a consistent REST
Another advantage of using this pattern to provide REST interfaces is that it facilitates the implementation of multiple resource representations. If controllers and their methods (actions) correspond to resources and the interfaces to manipulate them, the view part of MVC can be considered their representation. REST stipulates that a single resource may have multiple representations (e.g. a generic “person” resource may be viewed as a web page, XML or vCard [DH98] document). Rails provides a mechanism that allows it to use the same business logic to perform some operation on a resource and then respond with the requested representation format (which is part of the URI) as shown in Listing 5.1. Due to the way that web resources are supported in Ruby on Rails, the URIs to access them indicate their representation format. As an example, the same “person” resource may be accessed at /person/john for its HTML representation or at /person/john.xml for its XML representation and so on.

Listing 5.1: Multiple representations for a single resource in Rails

```ruby
# GET /programs/:id
# GET /programs/:id.:format
def show
  @program = Program.find(params[:id])
  respond_to do |format|
    format.html # show.html.erb
    format.xml { render :xml => @program.to_xml }
  end
end
```

Rails promotes the development of custom solutions to each project, based on the framework’s ease of learning and use. This is particularly reflected in the user interface part of applications, since there are many plugins and extensions to reuse at the code-level. By emphasizing custom development of certain aspects typically needed in many applications (e.g. messaging systems, authorization mechanisms), there is a trade-off between better control of the system’s code and speed of development of those aspects. On the other hand, the many benefits presented in this Section, are more than sufficient to chose the Ruby on Rails framework to implement the web applications that form this system. It also features a low barrier of entry to new developers and an active community, enhancing this system’s chances of future development as an Open Source project.
5.2.3 Soma Suite

Requirement 13 (Section 3.7.1) stipulates that this system must support a playout solution for the programming schedule; Soma Suite (introduced in Section 2.5.2) was chosen to handle part of this function. There are various advantages that influenced this decision:

- Its design facilitates the use of Soma within a networked environment.
- It has a mechanism to update its configuration from other processes.
- It is stable and used by many stations.
- It is a time-based audio player (or “scheduler” in radio terms), which avoids the need for more complexity in the Playout module.
- It supports the use of spots at specified time intervals, separate from the main programming schedule; this is enough for the promotional needs of the targeted radio stations.
- It is FOSS and its development is still active.

5.2.4 iCalendar

iCalendar (also known as iCal) is a standard format defined in RFC 2445 [DS98] for the exchange of calendar information. It was originally developed as a response to the various group scheduling and Personal Information Managers (PIM) products that communicated over the Internet but used proprietary formats. The basis for iCalendar is the earlier work developed for the vCalendar specification. It uses the MIME type “text/calendar”, providing a format for capturing calendar event, free-busy time, to-do and journal entry information. An Event calendar component represents a scheduled amount of time on a calendar. It supports recurring and one-time events. Recurring events can be defined using complex rules such as “on the third Sunday every two months” or “every two weeks on Tuesdays and Thursdays”.

Due to its flexibility to define event occurrences and widespread use as an Internet standard, the iCalendar format chosen to represent the programming schedule. There are many clients that support it (both desktop and web-based) with proper interfaces and necessary functionalities to deal with calendaring information. As such, this system assumes that the calendar construction activity is performed in one of those applications and later imported as the programming schedule. Nonetheless, it is important to note that the system also supports the creation and editing of single-time broadcasts, but to create many broadcasts at the same time, the calendar object is used (as described in Chapter 4).

5.3 Deployment

This section presents a possible deployment scheme for this system (Figure 5.3). The two modules can be deployed to two different servers with access to the Internet. The servers need to have a web server, a database system and the Ruby on Rails environment. Each application requires a unique URI, such as http://scheduler.example.com and http://playout.example.com. It is of course possible to deploy everything to the same machine, as long as there is an URI for each application. The server supporting the Playout module additionally needs Soma Suite and the necessary audio playback capabilities. The techno-
logical stack is not resource-intensive, thus fulfilling the requirement that the system is able to run on low-end hardware.

![Deployment Diagram](image)

Figure 5.3: Deployment scheme.

### 5.4 Implementation Overview

The aforementioned technologies were used to develop a working implementation for the proposed design. As presented in Chapter 4, there are two modules that compose this system. They provide two sets of web resources (through RESTful interfaces), associated with their respective responsibilities. Using those resources, the web interface (part of the Scheduler module) is the interaction mechanism between users and the application.

A majority of features defined by the system’s requirements were implemented, particularly those needed to create a working proof-of-concept application. Some, such as an interface to review an author’s historical assiduousness, were not developed but the underlying code and records exist, leaving their implementation as a matter of making some small adjustments in the code base and creating appropriate HTML interfaces.

Another limitation of this implementation is in the authentication and authorization at the API level. Currently, all write operations are only protected by HTTP basic access authentication. Under this approach, HTTPS must be used if connections need to be secured. There have been recent advancements to create open protocols to handle API authentication and authorization, such as OpenID[^5] and OAuth[^6], which could be supported in the future to advance this system’s use of open standards and enable other sites to reuse the data that is made available.

### 5.5 Scheduler

The Scheduler module implementation corresponds to 22 classes for controllers and 26 for models. The application code is organized in the Rails structure (Figure 5.1). Controllers, models and views are located in

[^5]: http://openid.net/
[^6]: http://oauth.net/
their directories under the app folder at the root of the project. Throughout development, unit and integration
tests were created, respectively targeting model and controller classes.

Another difference between the system’s requirements and its implementation is related to intermissions. Due to timing constraints, it was not possible to implement this feature as initially desired. The functionality was simplified and left for future work. Intermissions occurring within broadcasts are not supported and intermissions between different broadcasts are considered gaps that the Playout module must fill using spots. Thus, spots do not have the time-based rules that define when they should be played and for now are simply randomly chosen. Although this feature was initially stated as a system requirement, in practice, it does not affect Rádio Zero’s programming, because its current needs for intermissions can be fulfilled as the feature was implemented.

5.5.1 Resources

This Section summarizes the implementation of the web resources already described in Section 4.4.3. Table 5.2 lists for each resource, if it was implemented, which HTTP verbs are supported and which representation formats are available. Some verbs are not fully supported in some resource representations, but in those cases the HTML format is implemented, to allow the necessary interactions with users.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Implemented?</th>
<th>Verbs</th>
<th>Representations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schedule</td>
<td>Yes</td>
<td>GET</td>
<td>HTML, XML, iCalendar</td>
</tr>
<tr>
<td>Programs</td>
<td>Yes</td>
<td>GET, POST, PUT, DELETE</td>
<td>HTML, XML</td>
</tr>
<tr>
<td>Broadcasts</td>
<td>Yes</td>
<td>GET, POST, PUT, DELETE</td>
<td>HTML, XML, iCalendar</td>
</tr>
<tr>
<td>Gaps</td>
<td>Yes</td>
<td>GET</td>
<td>HTML, XML, iCalendar</td>
</tr>
<tr>
<td>Singles</td>
<td>Yes</td>
<td>GET, POST, PUT, DELETE</td>
<td>HTML, XML</td>
</tr>
<tr>
<td>Spots</td>
<td>Yes</td>
<td>GET, POST, PUT, DELETE</td>
<td>HTML, XML</td>
</tr>
<tr>
<td>Playlists</td>
<td>Yes</td>
<td>GET, POST, PUT, DELETE</td>
<td>HTML, XML</td>
</tr>
<tr>
<td>Editors</td>
<td>Yes</td>
<td>GET, POST, DELETE</td>
<td>HTML, XML</td>
</tr>
<tr>
<td>Authors</td>
<td>Yes</td>
<td>GET, POST, PUT, DELETE</td>
<td>HTML, XML</td>
</tr>
<tr>
<td>Users</td>
<td>Yes</td>
<td>GET, POST, PUT, DELETE</td>
<td>HTML, XML</td>
</tr>
<tr>
<td>User Mailboxes</td>
<td>Yes</td>
<td>GET</td>
<td>HTML, XML</td>
</tr>
<tr>
<td>User Mailbox Messages</td>
<td>Yes</td>
<td>GET, POST, PUT, DELETE</td>
<td>HTML, XML</td>
</tr>
</tbody>
</table>

An interesting consequence of using RESTful interfaces is that having multiple representations for the same resource is clearly useful. As an example, the Schedule resource may be used in HTML through the web interface. The XML representation is used by the Playout module. The iCalendar representation can be used by different users as a way to subscribe to the station’s programming information (adding that URI to their personal calendar application). All of this can be achieved by the same business logic code to generate the necessary information and then rendering it in different formats.

The Schedule resource is used by the Playout module to inform itself on the programming for a given timeframe. Listing 5.2 shows an excerpt of the XML representation for this resource. In this particular example, it is composed by an array of broadcast resources: an emission (another name for a first-time broadcast) and
a gap. The composing elements of these resources (introduced in Sections 4.4.3.3 and 4.4.3.4) can be observed in this example.

Listing 5.2: Excerpt of XML representation of Schedule resource

```xml
<?xml version="1.0" encoding="UTF-8"?>
<schedule>
  <broadcasts type="array">
    <broadcast type="emission">
      <id type="integer">1536</id>
      <program id type="string">gravidade-zero</program>
      <datetime start type="datetime">Fri Aug 22 19:03:00 +0100 2008</datetime>
      <datetime end type="datetime">Fri Aug 22 20:00:00 +0100 2008</datetime>
      <description type="string">Excerpt of XML representation of Schedule resource</description>
    </broadcast>
    <broadcast type="gap">
      <datetime start type="datetime">Fri Aug 22 20:00:00 +0100 2008</datetime>
      <datetime end type="datetime">Fri Aug 22 20:30:00 +0100 2008</datetime>
      <description type="string">Gap</description>
    </broadcast>
  </broadcasts>
</schedule>
```

5.5.2 Web Interface

The web interface is a very important part of the application, since it is where users visualize and control the system's functions. It was developed using standard web technologies: HTML, CSS, *JavaScript* and AJAX\(^7\). To render dynamic content, ERB\(^8\), the standard Rails templating system was used. ERB features the use of *Ruby* code embedded in text documents (in this case, HTML) for the purposes of generating document information details and/or flow control.

One of the limiting factors to the further advancement of this implementation were various issues related to the web interface. The need to develop from scratch the user interface and form processing, rendering differences and problems across different browsers and personal inexperience in the use of web technologies caused slower than desired progress. Nonetheless, many features and screens were developed, some of which are described in the following Sections.

5.5.2.1 Schedule timeline

One of the main screens in the system (Figure 5.4) shows the schedule for the next 24 hours. It is a list of the complete broadcasting (first-time broadcasts, repetitions and gaps), indicating their start/end date and time. Each entry links to the broadcast page and to the program page (when not a gap). In the sidebar, there are options to create a new broadcast, import programming from a calendar file and list all registered broadcasts.

---


\(^8\) [http://ruby-doc.org/stdlib/libdoc/erb/rdoc/classes/ERB.html](http://ruby-doc.org/stdlib/libdoc/erb/rdoc/classes/ERB.html)
and gaps in the system. Additional listing options could be added (such as direct links to broadcast listings by month, year or programs), although there already is a mini-calendar that allows for this time-based navigation.

5.5.2.2 Listing broadcasts

Broadcasts are listed in one of two ways: globally, as part of the schedule; or indexed by program. These options are important when reviewing the station’s overall programming schedule or only a single program’s. This can be seen by the navigation bars in Figure 5.5 and Figure 5.6. As a contrast to the schedule timeline, when listing broadcasts, gaps are excluded because the notion of a programming gap only makes sense within a schedule.

Figure 5.4: Schedule timeline.

Figure 5.5: Navigation bar to list all broadcasts by date.

Figure 5.6: Navigation bar to list all broadcasts by program and date.
5.5.2.3 Program list

The application lists all registered programs to facilitate access to editors and authors. They are grouped alphabetically and are easily accessible, as show in Figure 5.7. For faster access to common options, when the mouse pointer hovers a program's name, links to edit or delete it from the system appear beside it.

![Figure 5.7: List of all programs.](image)

5.5.2.4 Program homepage

A program's homepage (Figure 5.8) lists its most important attributes and management options. It shows the program's name, description and upcoming first-time broadcasts. The sidebar has links needed by authors and editors: editing program properties (namely its name and description), sending a message to its authors, or (for authors) to send a message the station's editors. The mini-calendar is a variation of the one used in the schedule page where, besides the time-based navigation of broadcasts and repetitions it allows, the days of the month where they occur are highlighted for improved usability. Authors are also listed in the sidebar. When an author's link is accessed, that user's page relative to that program is shown. When the option to add an author is clicked, the user is taken to a new screen in which he can select a user from a list and indicate when that user is an author of this program.

5.5.2.5 Broadcast structure definition

To define the contents of a broadcast, this system uses the concept of a structure (c.f. Section 4.4.2.4). It is a grouping of segments, each referring to audio assets. Figure 5.9 shows a screen segment with the available
components to define a *structure*. A user may choose from a single, a live source, playlist or an intermission, although the latter is not yet implemented. Singles refer to a single file that is to be produced by an author or that already exists in the system. A live source corresponds to an URI (from which the transmission originates) and can be associated with a *single* asset to be delivered after the broadcast, if a repetition of that content is necessary. Finally, a playlist refers to one of the existing playlists in the system.

By using these basic components for a broadcast, it is possible to satisfy the needs for the targeted radio stations. As a “shortcut” when defining broadcast structures, editors may create “structure templates”. They are generic structures that can be referenced when creating broadcasts. Users give them a name and identifying color (as seen in Figure 5.4, this information appears as colored tags beside broadcasts in the lists).

Figure 5.9: Basic components that may be used to define a broadcast structure.

Figure 5.10 shows the available fields when using a single audio asset, serving as an example of the options that appear after choosing one of the aforementioned components. The user must select if the content is to be produced by an author (and it that case either choose if it has the same length as the broadcast or a specific duration) or if it already exists in the system, specify which one, from a list. These different options are rendered using a combination of *JavaScript* and *AJAX*, to provide a more fluid interaction.
5.5.2.6 Audio Asset Delivery

One of the main objectives for this system is to have an integrated solution in which the programming schedule is defined and its contents are delivered by authors when they produce said content (or audio assets). Because the sources for audio assets can be different, this has to be taken into account. The present Section describes this mechanism. Figure 5.11 shows a broadcast screen, showing the date and time for the broadcast and its structure (under the Segments section). In this example, this broadcast is composed of a single audio asset, with an expected length of 30 minutes. At the sidebar, the author can click on the “deliver content” option, taking him to the screen shown in Figure 5.12.

When in the second screen, the user must select one of the registered audio asset sources in the sidebar, introduce a password if necessary and connect. A list of files that exist in that repository is presented, and the user must select the one that corresponds to the expected audio asset. After this process is concluded, the Playout module will retrieve the audio asset from the repository as described in Section 4.5.4.3.

5.5.2.7 Editorial Dashboard

A very simple editorial dashboard was created to provide an overview of the system's status. The information presented in this screen (Figure 5.13) serves as a proof-of-concept, since it is only a small part of what could be aggregated in this page. It shows the next 5 broadcasts and their content status, as well as the most recently delivered audio assets.

Figures 5.14 and 5.15 illustrate the cases described in Section 4.4.4.2. The first shows the broadcast status as “delivered”, since the assets are not yet available at the Playout module, and the second one is
listed as “available” because all of the broadcast’s assets are present at that module and thus ready for their broadcast.

5.6 Playout

The Playout module implementation corresponds to 5 controller classes and 4 model classes for the web application. Wrapper classes to issue queries to the Scheduler service were also created. Development for this module was much more limited, when compared with the Scheduler. Once again, the criterium was to implement the necessary features to produce a working proof-of-concept application. There are still many aspects left for future improvement.

As described in Section 4.5, this module is composed of two main parts: a web application and a daemon. The web application makes available a set of web resources that are related to the entities that exist at the Scheduler service, displaying information associated with the locally performed activities.

5.6.1 Resources

As a contrast to the Scheduler module, resources at this service are read-only. This is because resource creation and management is performed at the Scheduler, and local resources simply show different information. As with the overall functionalities, implementation of resources in this service was less extensive, in comparison to the Scheduler.

The gaps and play logs resources were not implemented because they are not critical to the application’s core responsibilities. They are necessary to fulfill requirements related to the station’s historical records and accountability.

Only one representation format (XML) was used in the two resources that were implemented: singles and spots. These resources are needed to identify the status of audio assets within this service, which is information for which the Scheduler service queries to show its users if a given asset is ready for broadcast. Table 5.3 summarizes the implementation information of resources for this module.
Figure 5.13: Simple editorial dashboard.

Figure 5.14: Broadcast status after asset delivery and not yet available at Playout module.

Table 5.3: Summary of implemented web resources in the Playout module

<table>
<thead>
<tr>
<th>Resource</th>
<th>Implemented?</th>
<th>Verbs</th>
<th>Representations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gaps</td>
<td>No</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Singles</td>
<td>Yes</td>
<td>GET</td>
<td>XML</td>
</tr>
<tr>
<td>Spots</td>
<td>Yes</td>
<td>GET</td>
<td>XML</td>
</tr>
<tr>
<td>Play Logs</td>
<td>No</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

5.6.2 Daemon

This small component is responsible for continually issuing queries to the web application and thus keep local data reasonably updated. Rails does not have built-in support for long-running tasks, so an alternative

Figure 5.15: Broadcast status after audio asset is available at Playout module.
solution had to be used. The *Ruby Daemons*\(^9\) project was used to make the script run as a *daemon*, i.e. a background process not controlled by an user. To handle the periodic execution of method calls, the *Rufus Scheduler*\(^10\) is used.

The script periodically calls update methods implemented as a small library within the web application. These methods will then query the *Scheduler* module for the updated programming schedule or needed audio assets (c.f. Section 4.5.4.1 and 4.5.4.3). The interval is set at 30 seconds, but it is configurable. When there is an updated schedule, the web application outputs it in the XML format used by *Soma Suite*\(^11\), which the daemon script then uses to update the running *Soma* instance.

\(^9\)http://raa.ruby-lang.org/project/daemons/
\(^10\)http://rufus.rubyforge.org/rufus-scheduler/files/README_txt.html
\(^11\)http://www.somasuite.org/demo/man-palinsesto-cfg
6 Evaluation

6.1 Introduction

The system presented in this document primarily benefits the tasks associated with the management of the radio station’s schedule (c.f. Section 3.3.1). The objectives that guide the design of this system are geared towards creating an Information System that links various activities to the core of a radio station—its programming schedule.

This chapter first lists the tasks that the Programming Department of Rádio Zero performs and that were addressed in the implementation of the system. The time that these activities take to be carried out and issues related to them are also presented in Section 6.2. Due to the small number of editors, it was possible to directly ask them to assess how the implemented prototype corresponded to their expectations and determine which areas should be enhanced and developed in the future. It is a qualitative evaluation whose result is presented in Section 6.3.

6.2 Activities Overview

As already noted, Section 3.3.1 lists the main activities performed by the Programming Department of Rádio Zero. The system’s objectives, design and implementation focus on those tasks that are related to the programming schedule, which is the essence of any radio station. As a reminder, these broad responsibilities are listed below:

- Decide and set up the schedule for each broadcasting season.
- Receive, organize and prepare broadcasts delivered by authors to remote repositories.
- Control assiduousness of authors.
- Responsibility for special programs whose authors are not members of the station.

Table 6.1 shows a list of activities and the average time that it takes a person to fulfill them (based in the answer given by Rádio Zero’s editors). However, the time spent by editors performing their duties is not limited to the ones listed in the table. It is important to note that in an average total of 74 new broadcasts per week, authors fail to deliver content approximately 8 times. That is a significant figure that, as a consequence, makes editors waste many hours per week checking their email (to check if content has arrived), contacting authors (to try to make them deliver content or to find out what happened) and placing alternative content to replace the missing one.

Table 6.1: Main editorial activities and average time to perform them

<table>
<thead>
<tr>
<th>Activities</th>
<th>Average time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defining schedule</td>
<td>4 hours (per new schedule)</td>
</tr>
<tr>
<td>Putting delivered broadcasts in place</td>
<td>1-2 hours/week</td>
</tr>
<tr>
<td>Recording author assiduousness</td>
<td>30 minutes/week</td>
</tr>
<tr>
<td>Send message to an author</td>
<td>3-5 minutes</td>
</tr>
</tbody>
</table>
The other system stakeholders—the station’s authors—have a more limited set of tasks to perform, besides the production of their programs. Once they create content, they have to deliver it to the station and notify the editors. Authors may also want to make their programs available in other ways, such as a Podcast\(^1\), which involves filling the necessary metadata fields and publish the feed, using the ListGarden feed generator program\(^2\) (installed at a Rádio Zero server). When using the website to input information about a broadcast, users have to duplicate this information, since it is a separate system. All of these small tasks take time and can be improved, albeit not as much as for editors’ activities.

### 6.3 Assessment

Editor users were shown the implemented prototype and asked to comment how it fitted their expectations and needs for the system. Overall, they were satisfied with the results. Positive and negative aspects were identified, which will be addressed in the present Section.

- **Schedule definition**
  
  In terms of the activity of the schedule definition, there is a small improvement over the current time spent. The activity is complex, involving many editorial criteria and authors’ time-availability, and thus continues to be a time-consuming process. Currently, editors create the schedule using a Google Calendar at the start of a new broadcasting season, and then re-create that information in ZaraRadio (which is also a long activity). The calendar is only created for visualization purposes and is not kept up to date during the season. Under the new approach, editors only need to create the calendar and then input it to the system. By avoiding the need to define the schedule twice, in two different systems, there is a gain of approximately 1.5 hours. The bigger advantage is that because editors must now keep the calendar updated, they have an appropriate visualization of the schedule at all times. Editors prefer the calendar view of the schedule, because it is easier to identify empty time-slots and locate broadcasts.

- **Content delivery (according to editors)**
  
  Content delivery is another aspect focused by the implementation. Editors were extremely satisfied by this feature because it frees them from that task. They no longer need to wait for authors to notify them or visit the station’s FTP server (or some other location) to download and place a broadcast’s content for its playback. Additionally, if an author fails to deliver content, the Playout module will automatically choose a replacement. This is a big advancement over the current situation. On the other hand, some limitations of the implementation were identified as negative aspects: the fact that replacement content is randomly chosen and that there are no notifications when these problems occur. They were pleased with the general approach but feel that there is still work to be done before the implementation can be considered to be at production-quality.

- **Content delivery (according to authors)**
  
  Besides the editorial aspects, there are features for authors, as mentioned in Section 6.2. Authors from 6 programs were asked to comment on the content delivery feature and overall usability of the system.

---


In terms of the time spent performing that task, there were not any gains: the process is somewhat equivalent in that aspect. All authors were satisfied with the general organization of the UI. As to the content delivery process, they said that the process was not obvious, due to the location of some links and steps to be followed. This suggests that a better UI design and evaluation should be made.

- **Audit capabilities**
  The system currently records when authors deliver content to the station, but it does not yet have a visualization interface for this information. This is important to editors, because it frees from another task. After the improvements needed in the Playout module, this feature was identified as the next that should be implemented and enhanced. From the gathered data, editors would like to be able to see in the forms of graphs, the assiduousness of a given author through time and thus determine his/her reliability. Another statistic that they would like to be kept and be able to visualize is the number of listeners of the web stream through time, for given broadcasts and overall for a particular program. Although planned, but not implemented, was the record of the use of audio assets, that should be available for editors to see.

- **Communication**
  The simple messaging tool between users was considered adequate. Currently, when editors need to talk to authors, they need to visit the station's wiki to know which members are responsible for the program to with which they want to communicate and get their email address. By creating a direct link at each program's and author's page, the communication needs can be fulfilled within the system. As long as users are logged in to the system, they can see if they have messages and reply to them if they need to. A feature that was requested as future enhancement is that users should be able to receive email notifications when they get messages inside the system, in order to have a better reaction time.

- **Single information location**
  As a general note, editors were very pleased with the fact that this design allows them to keep the most important information for the station's operation in a single system. Users, programs, broadcasts, authorship, content and communication are managed within the same system, thus ensuring a coherent and updated set of information.

This basic assessment process served as a way to validate the success of the system implementation in achieving its main objectives. Users were very satisfied with the general approach and features, as they reflect their activities and needs well. Various suggestions were made (as referred above) for new features and enhancements to the ones that were presented to them, but it is possible to conclude through this evaluation that the system achieved its main objectives.
7 Conclusions and Future Work

7.1 Main Contributions

This document describes the current landscape of radio station software, sorting it according to applicational areas. After conducting interviews with various individuals responsible for radio stations, it was possible to identify a set of aspects that are useful to group radio stations in categories. By evaluating the existing systems according to those referred aspects, it was possible to conclude that there was a fundamental misalignment between what they offered and a specific group of radio stations. The design that was presented is focused in the schedule and playout applications that radios need, specifically targeting stations based on author programs, with mixed production environments, and low human and financial resources.

Because it was designed to tackle a different set of problems from the ones that current systems do, this work presents several main contributions, which will be discussed in this Section. There are still many possibilities for future enhancements and modifications in the future, which are addressed in Section 7.2.

Relative to the business side of the problem, there are multiple contributions, focusing on each radio station characterizing aspect defined in Section 3.2. Content can be delivered to the station from many different sources, thus supporting the mixed environment in which these stations operate. The system includes the notion of authorship associated to programs and broadcasts, making it easier to manage a station that emphasizes author programs. Having low human resources means that there is not always someone present to resolve problems as they arise. To avoid that human-related problems do not affect the station's broadcast signal, the system is prepared to fill empty programming hours or find alternative content when the original is not available. Since many stations have low financial resources, the implementation for this system will be released under an Open Source license.

This design does not need to have a studio in order for it to be used. Due to its targeting of mixed production environments, the system's applications need only to run on web servers. They can be accessed easily by physically distant users and thus support a “virtual radio station”. Contrastingly, many radio station applications follow the opposite approach, requiring a physical location where they must operate.

Other contributions stem from the use of standard web technologies and the exploration of the REST architectural style. This approach is very different from current applications, which are “closed” systems, not making available their data to the outside world. Making full use of the web and its technologies brings advantages. The basic information entities of the business are available with CRUD access to other applications and services as web resources. This architectural feature and the possibility of having many resource representations facilitates the integration of these applications with others and the development of complementary systems and services.

Moreover, because the system architecture is based on a set of services (in particular, REST interfaces), it is possible to decouple the various radio station application components by responsibilities. This facilitates the adoption of different technological solutions, as long as they respect the interfaces (i.e. web resources). As a consequence, the proposed solution is not dependent on the technological choices made for its current implementation.

There are limitations, as the proposed solution is not adequate for all types of radio stations—its design
targets stations that correspond to the specific attributes that have been discussed. Furthermore, there are improvements and new developments to be made in order for the implementation to achieve its full potential. Nevertheless, based on the aforementioned contributions and the positive evaluation presented in Chapter 6, it is possible to state that this system has achieved its main objectives. It presents a new approach to radio software applications and it is successful in addressing the problem that it set out to solve.

7.2 Future Work

There are several aspects in the current system implementation that can be improved in the future. This section discusses some recommendations given by users and possible future developments for the system.

The most important area of improvement is the Playout module. The algorithm that selects audio assets to fill programming gaps or undelivered content is currently random. It should be more elaborate, using different criteria such as “previous broadcasts with the smaller play count” or “a broadcast by the same author”. Also, audio assets are now downloaded to this module as soon as they appear as delivered at the Scheduler; this behavior could be optimized to only download the necessary assets according to their expected use in the schedule.

Another point for improvement is related to the web application’s UI. Some actions need to be simplified and important features should have a greater focus. These redesigns should be based on a more careful evaluation, involving more users and gathering more quantitative data to validate not only the UI but also the broader implementation. There are other visualization opportunities that should be developed. The concept of “Real-Time Enterprises” has been popular recently. Real-time is about getting the right data about the right processes to the right people at the right cost and at the right time to create and sustain competitive advantage [Tab07]. The editorial dashboard, the records of authors’ assiduousness, number of listeners and usage of audio assets, are all interesting information visualization challenges that could be addressed as a way to support the station’s management under this “real-time” paradigm.

Also, it is important to comply with metadata standards, in order to provide a better user experience and take advantage of developments in the industry. There are no large metadata formats that are an adequate fit for this system and its operational context. However, there are smaller initiatives such as XBMF that should be possible to integrate through the creation of other resource representations and the definition of the necessary metadata entities.

Finally, the availability of the station’s data could be used to develop complementary services and applications, as mentioned in the previous section. Some examples include: a podcasting service that uses broadcast metadata, the URI of assets and the time of broadcast to automatically publish podcast feeds after a show has been broadcasted; a notification service that listeners subscribe to receive alerts when selected programs were about to go on the air; desktop widgets that use the programming information of the station to list broadcasts by favorite programs or authors (i.e. EPG applications).

As can be gathered by the topics addressed in this section, there are many possibilities for development and enhancement of the system and its implementation. This demonstrates its essential validity, usefulness and interestingness for the community, hopefully ensuring its future as an Open Source project.
Bibliography


Appendix
Informal interviews were conducted with individuals responsible for their stations. These interviews followed a script (listed below) with questions with varying levels of detail. Beyond these questions, interviewees were also allowed to give their insights into their particular problems and needs. The focus was mostly on qualitative aspects, since the purpose of this activity was to gather information that allowed a better understanding of issues associated with the stations’ operation.

- Describe your radio station and its overall goals.
- How would you characterize the effect that human and financial resources have on your activity?
- How many persons are involved in administrative responsibilities in your station?
- How many authors are there in your station?
- How many programs does your programming schedule have?
- Which Operating System(s) are you using in your stations?
- Which scheduler or playout software do you use for your schedule? Are you satisfied with it?
- How do you update your programming schedule?
- Do you have an website showing your schedule, programs, authors, etc? Is it integrated with your current scheduling/automation application or do you have to do it by hand?
- How do you manage the contributions from the programs’ authors? That is, when someone makes a show, how is it delivered to the station? (Is it always done inside the studio? Do you use some server? USB drive?).
- What types of broadcasts do you have? (e.g. based on computer-generated playlists, author programs – live and pre-recorded, relaying other station’s streams, etc.).
- For each type of program, what’s the structure of the broadcast? (for instance, you may have pre-recorded shows that are one single file, or composed of different ones).
- Do you have intermissions between broadcasts? If so, what type of spots/jingles do you use? (advertisement, time-announcements, promos, etc).
- Do spots have fixed lengths or are they variable? What kind of schedule do you use for intermissions? Is it variable (like “play this stuff in intermissions at these particular hours and days”) or is it fixed (that is, you have to exactly specify what’s in every intermission)?
B Design Diagrams

Due to space limitations, the size of Figure B.1 may be inappropriate for reading a printed version of this document. The image is available online for better legibility\(^1\).

\(^1\)http://web.ist.utl.pt/daniel.zacarias/thesis/scheduler_models.png
Figure B.1: Class diagram of the Scheduler module.
Figure B.2: Class diagram of the Playout module.
Figure B.3: Class diagram of the *Playout* module (classes to wrap resources from the *Scheduler* service).