Radia Source - An Information System for the Management of Processes in a Radio Station

[Dissertation Extended Summary]

Daniel Enrique Zacarias Silva
Instituto Superior Técnico
Av. Rovisco Pais, 1049-001
Lisboa, Portugal
daniel.zacarias@ist.utl.pt

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 with 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.

General Terms
Management, Design

Keywords
Radio, Programming Schedule, Information System, REST, Resources

1. INTRODUCTION
There are many types of radio stations, but they share the same objective: to have a broadcasting signal that is the result of scheduled programming of the contents they produce. Applicational support for radio stations is diverse, focusing on the different aspects related to the operation of a station: its schedule, safe-guarding its audio assets and broadcasting them as scheduled, among others. These systems are usually designed with the most common case of station in mind: based on music playlists, some live shows and always produced and managed within the same studio. On the other hand, there is a smaller number of stations that do not fit within these molds. The present work addresses these issues, first contextualizing and analyzing the problem in order to later propose a solution and implementation prototype.

This document is organized in seven sections. Section 2 provides the basic context to understand a radio stations’ activity and its main concepts. Also, 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 Section 3. The proposed solution to achieve the established goals is described in Section 4, followed by the implementation details of the developed work in Section 5. A qualitative evaluation of the implemented prototype was performed with a key group of users, and is presented in Section 6. Finally, Section 7 concludes this document by discussing the main contributions of this work and future developments for it. The contents of this paper follow a similar structure to the full dissertation that this document summarizes [10], in order to facilitate further consultation and detail when needed.

2. BACKGROUND AND RELATED WORK
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 & 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.

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.

### 2.1 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* [8, 7] is presented in Figure 1 and explained below.

- **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 broadcast (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.

1Small audio pieces produced for their specific use in intermissions.

#### Figure 1: Concept map of a radio station schedule.

- **Live**: programs created by authors in the radio station.
- **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”).

- **Spots**: small sound pieces used in intermissions for advertisement, promotional or informational purposes.

#### 2.2 Survey of Radio Station Applications

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*. The features 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. The final comparisons are available in Tables 1 and 2.
2.2.1 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). Some of the most widely used software systems of this kind are the following: (1) Mar4Win\(^2\); (2) Mar4Win\(^3\); (3) Raduga\(^4\); (4) ZaraRadio\(^5\).

2.2.2 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 following systems were analyzed: (1) Reboot.fm; (2) Campcaster\(^6\); (3) Riven-dell; (4) Soma Suite\(^7\); (5) Savonet\(^8\).

2.3 System 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.

Table 1 shows a feature comparison between the proprietary solutions studied. Similarly, Table 2 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.

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. 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.

3. PROBLEM STATEMENT
This section explain why certain types of radio stations are not properly served by the systems listed in Section 2.2. 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.

3.1 Motivation
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.

- **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.

Rádio Zero has been growing its programming almost constantly while maintaining a practically flat (and low) number of people responsible for them. The responsibilities that
editors have take amounts of time that increase with the number of programs. Many of these tasks are repetitive and could be carried out by automated processes. This statement is further supported by considering the overall lifecycle and business processes for broadcasts presented in [10]. As previously referred, there are many software systems to support the activity of radio stations, most of them focusing on the schedule and playout applications.

3.2 Problem
Different software systems for radio stations were described and compared on a functional basis in Section 2.3. Table 3 presents a listing of those systems, now including the variables deemed to be important to categorize different radio stations in Section 3.1. 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.

*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, 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, 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 processes associated with broadcasts 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.

3.3 Objectives
The main objective of the present work is to create 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

---

### Table 1: 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 emission</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>

### Table 2: Feature list for Open Source schedulers

<table>
<thead>
<tr>
<th>Feature</th>
<th>Reboot.fm</th>
<th>Campcaster</th>
<th>Rivendell</th>
<th>Soma Suite</th>
<th>Savonet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Playlist definition</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Different playlist modes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Jingles/Spots</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Time announcements</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Live shows emission</td>
<td>Yes</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>
<td>Yes</td>
</tr>
<tr>
<td>Problem notifications</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Extensibility through APIs</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Remote administration</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Silence detection</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Broadcasted content logging</td>
<td>No</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>
<td>Yes</td>
</tr>
</tbody>
</table>
3.4 Use Cases and Requirements
The main use cases and requirements for this system are available in [10]. Essentially, they are related to the manipulation of the different concepts associated with a radio station programming schedule (e.g., programs, authors, broadcasts, etc.) and the need for said schedule to be played at the specified times in order to have a broadcasting signal. Also, there are non-functional requirements that state that the system must be modular, with loosely-coupled components, be able to run on commodity/low-end hardware and use Open Source technologies.

4. SOLUTION PROPOSAL
4.1 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 2, and it is based on the requirements and analysis of the problem.

4.2 Software Architectural Styles
An architectural style is a coordinated set of architectural constraints that restricts the roles/features of architectural elements and the allowed relationships among those elements within any architecture that conforms to that style [3]. 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 [1].

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 [4]. Based on the objectives and requirements established for this work, the desirable attributes for the system and the style guiding its architecture were defined: (1) facilitate loosely-coupled components; (2) facilitate the creation of a service architecture; (3) scalability; (4) simplicity; (5) extensibility; (6) support for standards.

4.3 Representational State Transfer (REST)
Representational State Transfer (or REST) is a software architectural style for distributed hypermedia systems [3]. The largest and most well-known implementation of REST is the World Wide Web [5].

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 [3].

4.4 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 4.2, 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 [2]) in this manner, makes integration more easily achievable, thus providing a modular support to the different radio station applicational needs.

4.5 Architecture

4.5.1 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 (respectively described in Sections 4.6 and 4.7). Because the system is an exploration of the REST architectural style, these modules communicate using standard HTTP interfaces, as shown in Figure 3.

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.

4.5.2 Actors

The necessary actor classes are not complex. As the requirements stipulate, they can be divided in four levels, 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;
4.6 Scheduler
The Scheduler module is concerned with 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.

4.6.1 Structure
There are important issues that are reflected in the class model for this module, stemming from the type of programming and the production environment, some of them addressed in this Section (for more, c.f. [10]). 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.

Another problem is related to programming gaps. This term refers 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.

4.6.2 Resources
The proposed system is an exploration of the REST architectural style. As such, the main business entities are available for manipulation via HTTP as resources. They are accessible through a pre-defined set of URIs. Among these resources are: the programming schedule, programs, broadcasts (either associated with the schedule or to a single program), authors and audio assets (singles, spots and playlists).

4.6.3 Dynamic
There are interesting flows that show the dynamic aspects for this module, covering the most important use cases, such as:

- Schedule update using an iCalendar file;
- Content contribution from users, from different sources (e.g. FTP server), selected through the system;
- Listing audio assets and showing if they are available in the Playout module;
- Listing broadcasts and determining if they are ready or not for their transmission, based on the audio assets they use.

4.7 Playout
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 an audio player or one of the systems mentioned in 2. To achieve this, it is necessary to convert the format provided by the Scheduler to one that the player system accepts.

4.7.1 Structure
Since it has a more varied set of responsibilities, there is 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.

4.7.2 Resources
Similarly to the Scheduler module, this one also implements resources with information relevant to its operation. It is important to note that this service does accept creation of new entities through its interfaces, it simply uses information from the Scheduler. The resources available in this module are: gaps, audio assets (singles and spots) and play logs.

4.7.3 Dynamic
The main processes performed by this module are related to the following issues:

- Obtaining the schedule from the Scheduler and updating its local information.
- Exporting the schedule from the generic XML format to the one needed by the Player application.
- Fetching audio assets needed by the scheduled broadcasts.
5. IMPLEMENTATION

5.1 Technology

5.1.1 Ruby and Ruby on Rails

The Ruby programming language was created by Yuhiro Matsumoto and publicly released in 1995\textsuperscript{10}. It is a genuine object-oriented language—everything that is manipulated is an object and the result of those manipulations are themselves objects [11]. It is part of a group of interpreted languages that have recently become more popular due to their many advantages for development work [9].

Ruby on Rails\textsuperscript{11} (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 [12].

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 [6], helping to separate the main concerns that such applications have to address.

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 interface.

5.1.2 Soma Suite

Soma Suite\textsuperscript{12} 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. This technology was chosen as the component responsible for the playback of audio files at the specified times in the schedule.

5.2 Deployment

This section presents a possible deployment scheme for this system (Figure 5). 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 \texttt{http://scheduler.example.com} and \texttt{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 technological stack is not resource-intensive, thus fulfilling the requirement that the system is able to run on low-end hardware.

5.3 Scheduler

The Scheduler module implementation corresponds to 22 classes for controllers and 26 for models. The application code is organized in the Rails structure [12]. Controllers, models and views are located in their directories under the \texttt{app} folder at the root of the project. Most of the designed resources were completely implemented (i.e. all verbs), while some are still incomplete in this respect. Also, resources have XML and HTML representations.

A large part of the implementation corresponds to the web interface and application that was developed. As an example of what was created, one of the main screens in the system (Figure 6), showing the schedule for the next 24 hours is presented. 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 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.

5.4 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
The objectives, design and implementation of the system focus on those tasks that are related to the programming schedule, which is the essence of any radio station. These are their main broad responsibilities: (1) decide and set up the schedule for each broadcasting season; (2) receive, organize and prepare broadcasts delivered by authors to remote repositories; (3) control assiduousness of authors; (4) responsibility for special programs whose authors are not members of the station.

Table 4 shows a list of activities and the average time that it takes a person to fulfill them (based on 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 4: 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>Placing delivered broadcasts</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>30 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. Essentially, once they create content, they have to deliver it to the station, add a description for it and notify the editors. There are other activities that may be performed (e.g. podcast feed creation), but are not relevant in the context of this work.

6.2 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.

- **Content delivery (according to editors):** Editors were extremely satisfied with 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.

- **Content delivery (according to authors):** 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.

- **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 them from another task. Users insisted that this feature should be implemented after addressing the issues in the **Playout** module.

- **Communication:** The simple messaging tool between users was considered adequate.

- **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

7.1 Main Contributions

This work reviewed the current landscape of radio station software systems, according to the two most important functional areas: schedule management and content playback. Also, through interviews, a set of aspects that are useful to
specifically targeting stations based on author programs and financial resources. The design that was presented is focused on alignment between what they offered and a specific group. It was possible to conclude that there was a fundamental mismatch between the existing systems according to those referred aspects, so group radio stations in categories were identified. 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 on mixed production environments in the playout applications that radios need, specifically targeting stations based on author programs, with mixed production environments, and low human and financial resources. On each of this business characteristics, there are specific contributions:

- **Author programs**: The system includes the notion of authorship associated to programs and broadcasts, making it easier to manage a station that emphasizes this type of programming.
- **Mixed production environments**: Content can be delivered to the station from many different sources, thus supporting this type of environments that reflect how some stations operate.
- **Low human resources**: 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.
- **Low financial resources**: the implementation for this system will be released under an Open Source license.

Due to its different design and architecture relative to current radio station applications, the proposed system brings other contributions to this area, such as:

- Architecture based on REST interfaces (i.e. services), thus easily decoupling applicational components by responsibilities and facilitating the adoption of different technological solutions.
- It supports a “virtual radio station” that does not possess a physical studio, due to the way that the system deals with content contributions.
- 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.

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 Section 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 lists the recommendations given by users and possible future developments for the system, such as: (1) enhancements to the Playout module; (2) better visualization and UI; (3) use of standard metadata format for further integration; (4) development of additional applications and services.

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.

### 8. REFERENCES