VRIJE UNIVERSITEIT AMSTERDAM
FACULTEIT DER EXACTE WETENSCHAPPEN

FINAL PROJECT THESIS

HearMe: Voice communication in Web 2.0 and beyond

Author:
Javier Quevedo Fernández
1731645

Supervisor:
Dr. Anton Eliens
Co-supervisor:
Dr. Patricia Lago

August 26, 2008

Forward The development of this project was possible for various reasons. One of them was the technical skills learned in the course *Intelligent Multimedia Technology* given by *Anton Eliens* and *Zeljko Obrenovic*. After the course *Anton* and *Zeljko* were very patient and helpful giving me the chance and support to develop the idea. *Anton's* role in the project was crucial, without his guidance I would have not been able to structure the writing of the thesis or would have seen the real possibilities or further possible improvements of the system. *Patricia Lago* as a co-supervisor made great improvements to the thesis with very helpful comments and suggestions increasing the documents quality in many aspects.

The project was deployed on a small testing environment due to hardware constraints. The need of a server machine and a professional bandwidth internet connection made it impossible for it to be tested on a wider scale. This leaves an interesting business opportunity for the future to deploy it in a commercial context.

Javier Quevedo Fernandez
August 26, 2008

Contents

Abstract

The web is today one of the most extended platforms for communications. Through the web users send emails, discuss articles or share pictures or videos. For this uses there are all sort of Web Sites and Web Services which provide users the necessary tools to create the desired content. But, where are the real time *voice recordings*?. Why can we not leave *voice messages* anywhere on the web? In this paper I study the benefits of using real time voice recordings on the web, I evaluate the available technologies that deal with this type of media and I implement a system to satisfy the needs.

Chapter 1

Introduction

Nowadays we perform all sort of communication through the Internet. What began as a network to share information on research and development in scientific and military fields[12] has turned into the most powerful communication tool for machines and humans. One of it's most extended applications is the Web. Apart from other Internet utilities such as email or file transfer, the World Wide Web is being widely used for human communication. Every time we post in a forum or in a blog, every time we share a video, we are communicating. We have many ways to do this, we can communicate with images, with text or with videos, however, there is no simple tool on the Web to communicate with voice, simply with our own voice.
How wonderful would it be if wherever we write a text message we could also leave a voice message? How wonderful would it be if in every Web Site we visited we could hear what others had said?

1.1 Definition of the problem

Formally the problem can be defined as it follows:

The nonexistence of a platform or Web Service to allow users to create audio content on the fly and place it anywhere on the Web.

Therefore, the main concern of this thesis is the following question: *How can I hear and be heard in the collaborative Web era and beyond?*

1.2 Document Organization

This thesis is divided in eight different chapters.

Chapter I and II are the *Introduction* and *Background*. The background places the reader in the context of the Web2.0, social applications and the understanding of why the *HearMe* infrastructure is a move ahead of the current situation.

Chapter III is the *Technology Survey*. This chapter analyzes the different technologies that currently exist and which are involved in some way in the research and development of this project. It focusses on the design of a Web Service and on how to deal with a non conventional input -voice- within the context of a Web Application.

Chapter IV is the *Architecture and Design*. This chapter defines the functional and quality requirements of the system and the use cases. Additionally it includes an overview of the architecture and a description of the API of the Web Service.

Chapters V and VI includes a detailed description of the *HearMe* player and the *HearMe* recorder design, and usage.

Chapter VII is called *Evaluation and future work*. This chapter is divided in different subsections, where the work done so far is evaluated and the deployment experiences are described. Furthermore the benefits of creating richer clients for the recorder and player applications are studied as well as issues about scalability or OpenSocial like the creating of gadgets.

Chapter VIII is the *Summary* over all.

1.3 Motivation

For a few years I used to play a MMORPG (Massively multiplayer online role-playing game) called *World of Warcraft*. In this game teams of 25 persons have to coordinate with each other in order to kill monsters and advance through the game. Because of the sophistication of the games tactics, it was absolutely necessary to use a voice communications tool to speak during the combats. The tool we used was a VOIP (voice over IP) software called *Ventrilo*. After a while we all got used to talk for long hours and felt very conformable with it. Another tool that we had to use was a Web Forum to keep track of tactics and other issues. We noticed that voice messages in the forum could come very handy, they would help us in the debates or in any of the topics we discussed about. This lead me to realize that I was never using my voice on the Web. Not in forums, wikis, blogs or anywhere. Why not? Why could I not leave my voice with all that it implies anywhere where

I was already leaving text? What would it need to be done to make this possible? With this motivation I began to think of how this idea could merge with the current web and how it could be implemented. With the help and support of *Anton Eliens* the concept got a shape and with this shape I began to implement parts of the it while writing this thesis document. There are other services that may seem close or similar to *HearMe*, but none of them are based on audio and none of them permit you to create the content on the fly and from an external page. So because nothing like it really exists, the work had to be totally self developed but using technologies available in the market that deal with the multimedia inputs in the web context.

Chapter 2

Background - Audio on the web

In the beginning we could see the Web as a huge memory that users would query. This information was mainly text, and the ability to create it was limited to just a few with lots of knowledge and resources. For regular users it was very complicated to contribute to this information in any ways until Web Sites like *wikipedia.org* or *debianart.org* appeared. This type of sites were called social networking sites, under the label of Web 2.0. In the Web 1.0 era, users were either publishers or readers[20], but there was no collaboration between both. As new Web technologies appeared, the Web evolved into a collaborative space, regular users began to have the ability to take part in the development of the Web content. According to *Tim O'Reilly*, what makes a Web be 2.0 it's not just the combination of new technologies such as *Javascript* or *Web Services*, it is that *they have embraced the power of the web to harness collective intelligence*[18]. As we can see on his article *What is Web 2.0*[18] there are many reasons why sites like *Adsense* have survived against sites like *Doubleclick*, or *Google* over *Altavista*. All of the reasons could sum up into one: *Network effects from user contributions are the key to market dominance in the Web 2.0 era*[20]. In social networks users *tag* the content defining metadata that gives additional information about it. User can also comment it and share it in many different ways. Users decide what is interesting and popular because without even realizing about it, they have become the publishers.

According to *K. Gottschalk, S. Graham, H. Kreger, and J. Snell*, a *Web service is an interface that describes a collection of operations that are network-accessible through standardized XML messaging*[13]. In the Web 2.0, Web Sites turn into Web Services by standardizing their operations and permitting other Web Sites or Web Services to connect to them and interexchange data. We can consider *social networks* as Web Services of a specific type of data. For example, we can

consider *twitter.com* as a Web Service for short text messaging, or *del.icio.us* as one for bookmarks. So speaking in a general way, most of the existing Web Sites act as *Web Services* of the nature of the data that they manage. This is why users have one or more accounts in each type service that they use, they have an account to publish videos in *youtube.com*, an account to manage blogs in *wordpress.com* or *blogger.com* and so on. There are plenty of *Web Services* for almost any kind of data that someone can think of, but yet, there is no *Web Service* that deals with voice recordings.

Although the technology doesn't make a Web be 2.0, it also has its part on it. The Web 2.0 would not exist if the technology was not able to support it. It is thanks to buzzwords like *PHP*, *Ajax* or *Webservices* why the sites have been able to create the infrastructure to provide this rich collaborative experience. In the last years there has been an increment of multimedia content such as audio and video. There are many reasons for this, but the most important are the increment of the internet speed access and the development of new hardware and software products. The Web has taken advantage of this technologies. In between 2004 and 2005 sites like *youtube.com* or *flickr.com* appeared. Users can upload their photos or videos and share it with the rest, creating databases of multimedia content. Now that the technology supports it, Web Sites can choose from different multimedia elements to include them. Most of the Web Sites use text, pictures or video for their material, but it seems like the voice is being left apart and it is not being widely used.

2.1 Media

Voice communications have many pros and cons comparing them with video or text. Initially it may seem that video is more complete than audio because video includes both audio and video, but this is not really be the case. We can look for example at the differences between the television and the radio. The television needs both visual and audible attention, which does not happen when you listen to the radio. It is much easier to do something else while you are listening to the radio than while you are watching television. Another good comparison could be a phone conversation versus a videoconference. In a videoconference, specially when you are doing it with someone you don't feel familiar with, you have to take care of your looks, take care of the photograph of the place you are at (illumination, scope, etc) while in a phone conversation you just have to take care that your voice is in good conditions and that the place you are at is silent enough. There

is also an endless debate between oral vs written communication. Voice communication carries many more elements than written, for example when we listen to someone we can easily tell if the person feels scared, in pain, or enthusiastic and also we have more elements to detect if the person is being sarcastic, lying or being sincere. Additionally, not everyone is able to read. The possible receivers of a message can vary depending on if it is a text, an audio or a video one. Speakers of one same language may not understand each other when talking because of the accent or their specific language skills. With text some receivers may speak the language but are illiterate, or they might be blind, but text has the advantage in that for someone who is not too skilled in one language or who has a dramatically different accent than another speaker of the same language, it is easier for him to understand a text message than an audio one. So the possible targets can vary depending on the type of media that is used. There are several ranking criteria to discuss about the benefits and looses from one type of communication to another. In the table 2.1 I gather some of these, to show that neither one of text, audio or video are better than the others, they are all complementary. Depending on the type of message than you want to transmit, you should choose from one kind to another. This is the main reason why voice should not be left out of the web, voice enhances the user experience by permitting him to express himself in another way, a way that we have been using for ages to communicate with each others.

Criteria	Text	Audio	Video
Edition	Simple edition	Semi-complex edition	Complex edition
Acquirement of information	The user sets the speed. Tradeoff between speed and understandability	Constant speed (seeking posibilities)	Constant speed (seeking posibilities)
	User decides the level of attention (user tends to be concentrated)	User decides the level of attention (tends to loose more information)	User decides the level of attention (user tends to loose some information)
	Possible to read simultaneusly multiple texts	Only one at a time	Somewhat simultaneous viewing
Machine processing	Easy to translate. Somewhat multilingual	Complex to translate, almost impossible to translate automaticaly	Complex to translate, almost impossible to translate automaticaly
	Easy to process and automate data retrieval	Complex to process and automate data retrieval	Complex to process and automate data retrieval
Information	Information is solid and concise	Information is in the air, not solid	Mixture of visual text and audio, solid and not
	No additional communicatil elements	Additional communication elements (voice tone, etc)	Many additional communication elements (gestures, tone, etc)
	Linkable, additional interactive elements	Interactive voice, time cuepoints	Interactive video, time cuepoints
Accessibility	Requires paper or screen (simpe to transport or recreate)	Requires audio equipment (semi complex transport or recreation)	Requieres audio and video equipment (complex transport or recreation)
	Only accessible to persons who can read (requires some language skills and education)	Anywhone who speaks the language (requieres high language skills)	Anywhone who speaks the language (requieres high language skills) but also possible to retrieve partial information without knowing the language

Figure 2.1: Text vs Audio vs Video

2.2 Possibilities of audio on the web

Imagine that there was a very elastic system that permitted you to build any kind of Web Application using audio recording. That you could either upload audio from files, you could create them *on-the-fly*, listen to them from any Web Site or download them to your personal music player.

The potential applications that could be built with this technology are yet to be discovered. The initial ideas that come into mind are to enhance current Web Sites like forums to include voice recordings. Updating the only text content to text and audio content. Another example of this could be to create a *twitter.com* clone of voice recordings, or a voice news system.

New applications that no one has thought about yet could be developed. For example suppose a person who has really good reading skills, he could build a Web Site where he offered readings of texts to other people, maybe news, maybe books. Another person may want to build recordings to create voice tutorials (voicecasts) to help other people be more productive with the tasks they usually

do. Other Web Sites could use it to enhance the communication between their users, for example *ebay.com* could include it so that consumers could send a message to a seller and vice versa.

Other interesting applications in the field of accessibility could be created. There are many Web users who are blind or illiterates for whom text is not a legible format. For this group of people there could be some applications related to translation, readings, guides and on.

There are already Web Sites that use voice recordings in some ways. Usually these are called podcasts. The system described in this thesis could be used as a support tool for them, making their task easier and more accessible to unexperienced users.

Summarizing, we can assure that with the proper audio recordings system, existing Web Sites could improve their services and content and that new Web applications could appear with new ideas and functionalities not ever seen before.

Chapter 3

Technology survey - Services and media

3.1 Considerations

The software to be developed consists in two different parts, each one requires to survey specific technological aspects. The first concern relates to how to deal with a non conventional input, voice, in the context of a web application. The second part focuses on how to develop a proper Web Service to fulfill the functional and quality requirements (detailed in the chapter Architecture Design). Also additional concerns about data storage are evaluated.

3.2 HearMe Web Service

The first thing to understand before explaining what a Web Service is, is why should *HearMe* be designed as a one. According to *Sun Microsystems*, Web Services should be used when you need interoperability across heterogeneous platforms. That is, when you need to expose all or part of your application to other applications on different platforms[17]. Because the *Hearme* is meant to be a neutral and externally accessible platform, its design as a Web Service is obligatory.

3.2.1 What is a Web Service?

If you ask five people to define Web Services, you'll probably get at least six different answers [15]. The standard definition according to the W3C is:
A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards [10]. This definition is slightly complicated and personally I prefer the one that Anne Thomas gives: A Web service is an application that provides a Web API. An API supports application-to-application communication. A Web API is an API that lets the applications communicate using XML and the Web[15].

Thanks to the use of Web Services, Web Sites now have the ability to inter operate ones with another exchanging data and resources easily. It is crucial for the *HearMe* to be designed this way for its purposes as a extensible voice communications platform.

3.2.2 RESTful Web Services

The concept of REST architectures was introduced by Roy Thomas Fielding in his PhD. dissertation [11]. The term REST stands for Representational State Transfer (REST) architectural style and in his dissertation he describes how REST web applications can request and manipulate web resources by the use of standard HTTP methods GET, POST, PUT and DELETE. There are many advantages of using a RESTful design for the architecture of *HearMe*, most of them can be seen in the book RESTful Rails Development[19]. In the section 1.2 of the mentioned book, the authors give us the following key advantages:

- Clean URLs. REST URLs represent resources and not actions. URLs always have the same format: first comes the controller and then the id of the referenced resource. The requested manipulation is independent of the URL and is expressed with the help of HTTP verbs.

- Different Response Formats. REST controllers are developed in a way that actions can easily deliver their results in different response formats. Depending on the requirements of the client, the same action can deliver

HTML, XML, RSS, or other data formats. The application becomes able to handle multiple client demands, cleanly and simply.

- Less Code. The development of multi-client-capable actions avoids repetitions in the sense of DRY (don't repeat yourself) and results in controllers having less code.
- CRUD-oriented Controllers. Controllers and resources melt together into one unit, each controller is responsible for the manipulation of one resource type.
- Clear Application Design. RESTful development results in a conceptually clear and maintainable application design.

3.2.3 Ruby on Rails

RESTful Web Services can be developed using various Web development frameworks such as Konstrukt, Cake PHP, JSF, Struts and on. Rails is one of these based on the programming language Ruby. Marcus Baguley discusses in the article[2] *10 Reasons why - ruby on rails* about the benefits of using this framework amongst the others. In the following section the key concepts and philosophy of the framework are analyzed in order to justify its usage for the development of the *HearMe* Web Service.

Introduction

The Ruby programming language was released by Yukihiro Matsumoto in 1995. Yukihiro affirms[16] that Ruby is designed for programmer productivity and fun, he stresses that systems design needs to emphasize human, rather than computer, needs and that the language should behave in such a way as to minimize confusion for experienced users. In July 2004, David Heinemeier Hansson released the first version of Ruby on Rails, a web development framework based on Ruby with a model-view-controller architecture. The philosophy of Rails is to make it easier to develop, deploy and maintain web applications.

Rails philosophy and key concepts

In order to make easier to develop, deploy and maintain Web applications, Rails focuses on a couple of key concepts[9].

- DRY: DRY stands for *don't repeat yourself* and it means that every piece of knowledge in a system should be expressed in one place. Thanks to the advantages of Ruby and its syntactic sugar there is very little duplication of code in Rails web applications.

- Convention over configuration: Rails has sensible defaults for just about every aspect of knitting together web application. By following the conventions, usually called the Rail's way, you can write a Rails application using less code than a typical Java web application uses just in XML configuration. It is also possible to override the conventions in case the application does not fit into the Rails standards.

- Model-View-Controller architecture: Model-View-Controller is the concept introduced by Smalltalk's inventors (TrygveReenskaug and others) of encapsulating some data together with its processing (the model) and isolate it from the manipulation (the controller) and presentation (the view) part that has to be done on a UserInterface. A model is an object representing data or even activity, a view is some form of visualization of the state of the model, and a controller offers facilities to change the state of the model. Rails follows this principle in order to structure application which helps the development and maintenance.

- Generators: Rails includes many generator scripts that create Ruby skeleton code leaving the programmer with just the need to fill it with their code, simplifying the task of code organization and also helping the developer to place each thing were it belongs. A really good example of this scripts is called the Scaffold script.

- ActiveRecord: Active Record connects business objects and database tables to create a persistable domain model where logic and data is presented in one wrapping[3]. Its an implementation of the object-relational mapping (ORM) pattern by the same name as described by Martin Fowler: An object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data. ActiveRecord has support for most of the current database systems, such as PostgreSQL, SQLite or MySQL. Because of this Rails developers do not have to deal with specific SQL statements depending on the SQL manager that they are using . Its implementation is Thread Safe, supports easy associations, it has built-in validations

and supports custom value objects such as timestamps, money or temperatures.

- Other goodies: Rails includes integrated Web Services support, a unit testing framework, and isolated environments for development, testing and production.

Rails and RESTful resources

When the RESTful concept begun to be used in the design of Web Services and resources Rails rapidly adapted to it. At first a plugin called *simply_restful* was developed in 2006 that was soon included in the Rails core.

The scaffold generator takes a name and the model attributes and creates a resource with it. It creates the model, controller and view, and also a migration script to generate the database table with the appropriate fields. In addition to all of this, the generator adds a mapping entry *map.resources :resource_name* on the *routes.rb* file which is responsible for the RESTful character of this resource. So simply with the primitive *map.resources* the Rails magic takes care of creating the RESTful resource.

REST actions are activated through a combination of a resource URL and an HTTP verb. Normally the clients of a resource are web browsers but also Web Services may want to manage the resource. If a RSS reader request data from a REST resource it will probably expect an Atom formated response, a Web Service a XML one and a Web Browser an HTML one. For this, Rails includes a primitive called *respond_to* which specifies the type of response to an specific action. The following example illustrates how the *respond_to* works.

```
def show
@project = Project.find(params[:id])
respond\_to do |format|
format.html # show.rhtml
format.xml { render :xml => @project.to_xml }
end
end
```

Thanks to this the RESTful resource can be easily accessed from many different types of clients and creates the expected response from each one of them.

Example of creating a RESTful resource with Rails

Let's say that we want to create a web site called School that includes a resource of students. After installing Ruby, Ruby gems and Ruby on Rails, we would have to create the skeleton of the project. For this we would invoke the *rails school* script. This script creates a group of directories and files which serve as a skeleton for the resource. The skeleton includes the configuration, controller, model and views. Additionally it creates a database schema, unit testing and log files. The details of the output of the script can be found in the Appendix section.

After executing the command *rails school* Rails automatically creates different files and directories. Now we would have to create a model for the students resource, the views and controller for each operation and etc. We can do all of this together by executing the scaffold generator:

```
> script/generate scaffold student
      exists  app/models/
      exists  app/controllers/
      exists  app/helpers/
      create  app/views/students
      exists  app/views/layouts/
      exists  test/functional/
      exists  test/unit/
      create  app/views/students/index.html.erb
      create  app/views/students/show.html.erb
      create  app/views/students/new.html.erb
      create  app/views/students/edit.html.erb
      create  app/views/layouts/students.html.erb
      create  public/stylesheets/scaffold.css
  dependency  model
      exists    app/models/
      exists    test/unit/
      exists    test/fixtures/
      create    app/models/student.rb
      create    test/unit/student_test.rb
      create    test/fixtures/students.yml
      create    db/migrate
      create    db/migrate/001_create_students.rb
      create  app/controllers/students_controller.rb
      create  test/functional/students_controller_test.rb
```

```
      create  app/helpers/students_helper.rb
       route  map.resources :students
```

In the file *db/migrate/001_create_students.rb* we have to include the attributes of a student. If a student has an Identificator, a date of birth, an name, we would have to include the following lines in the file:

```
class CreateStudents < ActiveRecord::Migration
  def self.up
    create_table :students do |t|
      t.column :name, :string
      t.column :birth, :date
      t.timestamps
    end
  end
  def self.down
    drop_table :audios
  end
end
```

After we have included those lines we have to invoke the following script:

```
>rake db:migrate
(in /Users/senc/Documents/temp/school)
== 1 CreateStudents: migrating ==============
-- create_table(:students)
   -> 0.0034s
== 1 CreateStudents: migrated (0.0041s) =====
```

This automatically connects to the database and creates the table students with a column for each of the students attributes. Thanks to ActiveRecord the model students is now dynamically linked to the database. With just a few lines of code and after executing a few scripts we have created a RESTful resource for students. By accessing *http://server/students* we can create, edit or delete students and also access it from another Web Service with standard REST http verbs.

Another goodie about Rails and ActiveRecord is the easiness to create associations. The file *app/models/student.rb* details the associations between this model and any other models. For example if there was another model for courses which had an association with students that said that one course could be taken by one or many students we would have to include the following lines in the model files.

```
file: app/models/student.rb
class Student < ActiveRecord::Base
  belongs_to :course
end

file:  app/models/course.rb
class Student < ActiveRecord::Base
  has_many :students
end
```

As it has been show, it is really easy to create associations between different models. You can also create other type associations with primitives like *has_and_belongs_to_many* or *belongs_to_many*. Thanks to this, if we had a variable called *@course* and we wanted to get an array of the students that are enrolled in that course we would do it with *@course.students*.

HearMe and Rails

In the chapter *Architecture Design* there is a detailed description of the functional and quality requirements and those match with the Rails and RESTful principles. Indeed, because of all of the benefits previously mentioned, Rails also appears to be the perfect Web development framework to implement the *HearMe* Web Service.

3.3 Voice on the web technology

HearMe deals with a non conventional input, voice, from the user in a Web Site context. In order to retrieve the voice the user needs a capable hardware, normally a microphone, and a software piece which can operate inside a Web Page.

3.3.1 Hardware

There are two hardware devices that are able to retrieve sound. One are mobile phones, and the other one are the computer microphones. Although it would be very interesting to analyze the possibilities of permitting users to create or upload recordings made with the mobile phone, this feature will be discussed in the chapter *Evaluation and future work* as a possible enhancement to the system. Most of the PCs and Laptops have microphones or at least a sound card with

a microphone input. If not there are also USB microphones. This means that virtually any desktop or laptop computer has the capability to retrieve the audio. The quality of these microphones may not be really good, but it is enough for the needs of the system. Computers with no microphone will have to buy one, but the prices are really low compared to any other piece of computer hardware.

3.3.2 Software

Web browsers basically understand variations of HTML, CSS and Javascript. With Javascript or any other technology that is built-in the mainstream web Browsers, we can read the inputs of the user with the mouse and the keyboard, but we cannot read other inputs like the webcam or the microphone. If we want to connect our web application with the end-user's microphone or webcam we have to either embed a *Java* or a *Flash* application.

Java

Java applications require not only the *Java Runtime Libraries* but also a *Java Plugin* on the browser. Java supports the reading of the microphone or webcam and supports the communication with a back-end server to send the streams, but the main problem with *Java* is the low performance of *Java* applications and that it is very complicated to deal with security and permissions when reading multimedia inputs.
Thanks to that *Flash* has many advantages over *Java* for this specific purpose, *Java* is mentioned and evaluated but it will not be used.

Adobe Flash & Flex

Adobe Flash - previously called *Shockwave Flash* and *Macromedia Flash* - is a set of multimedia technologies developed and distributed by Adobe Systems and earlier by Macromedia[6]. It is available for most of the web browsers and some mobile phones and it only requires to have the *Flash Plugin* installed. The mature markets present a 99.3% of ubiquity of the *Flash plugin* in web browsers [1], so we know that if we deploy a *Flash based* web application almost everyone is going to be able to use it.

Adobe Flex is a collection of technologies released by *Adobe Systems* for the development and deployment of cross platform, rich Internet applications based

on the *Adobe Flash platform*[7]. ActionScript is the language used for the development of *Flash Applications*[5].

Actionscript 3.0 is the language that *Flash* applications are written in.*Actionscript* was created precisely to deal with graphics and multimedia purposes on the web, it includes primitives to easily work with the multimedia inputs. When recording the reading of multimedia inputs require an interface between the multimedia hardware layer and the client application. This interface is given by the *Flash plugin*. With just one line of code we can gain control of the microphone and with a few more we can connect to a *Flash Media Server*. *Flash Media Server* is the streaming server which *Flash* applications work with and it is used to stream in both directions. For example, we can stream an audio or video from a *FMS* (*Flash Media Server*) to a web application, and we can do it the other way around, we can stream the sound gathered from the clients microphone to a *FMS*. So combining a *Flash* application on the client side with a *FMS* on the server side, we can retrieve, stream and store the sounds produced by the user on a server. In the same way we can have a set of voice recording on the server side and stream them to the clients web application.

There are two possible alternatives to implement the *Flash Media Server* part. One of them is the proprietary *Adobe's Flash Media Server* and the other one is the open source alternative, *Red5*. Both of them do more less the exact same task, *Red5* is programmed in *Java* and *FMA* in Actionscript. I will use *Red5* simply because it is free, well documented and it runs on more operating systems.

Flex characteristics:

- Flex is a free, open source framework for building highly interactive, expressive web applications that deploy consistently on all major browsers, desktops, and operating systems.
- It provides a modern, standards-based language and programming model that supports common design patterns.
- MXML, a declarative XML-based language, is used to describe UI layout and behaviors, and ActionScript 3, a powerful object-oriented programming language, is used to create client logic.
- Flex also includes a rich component library with more than 100 proven, extensible UI components for creating rich Internet applications (RIAs), as well as an interactive Flex application debugger.

- RIAs created with Flex can run in the browser using Adobe Flash Player software or on the desktop on Adobe AIR, the cross-operating system runtime.
- LiveCycle Data Services ES is a high performance, scalable, and flexible framework that simplifies development of Flex and AIR applications. Backed by a powerful data services API, LiveCycle Data Services simplifies complex data management problems such as maintaining a single instance of data across the application, data synchronization across clients and applications, and conflict resolution really simple.
- LiveCycle Data Services ES is architected to be scalable, secure and high performance server. Besides basic remoting capabiliites, it supports a rich set of features to create real-time and near real-time applications. It abstracts the complexity and realiability required to create server push (comet like) applications.

3.4 Data Storage

There are two main concerns when dealing with data storage. The first one would be the size and compression of the files to hold, the second one the distribution of the storage hardware.

3.4.1 File type and compression

Flash uses *.FLV* file for the streaming operations. The *.FLV* works as a container of both audio and video. The initial system only supports audio therefore the video compression will not be discussed. The audio can be encoded with different codecs. As we can see in the wiki Open Source Flash[4], the codecs that Flash supports for audio are MP3, AAC, PCM and ASAO. The ASAO codec is the default for speech compression. It has been developed by Nellymoser, a company in Arlington, Massachusetts specialized in mobile computing. According to the Wikipedia[8], the codec is optimized for real-time and low-latency encoding of audio. Adobe Flash Player clients, when recording audio from a user microphone, use the Nellymoser ASAO codec and do not allow Flash programmers to select any other. The sampling rate of the audio capture can be controlled by the Flash programmer to increase and decrease encoding bitrate and quality. Flash forces us to use this codec in the client side in order to stream the audio to the Flash

Media Server. Once that this audio has been retrieved in the flash media server it could be re-compressed using another codec such as MP3 or ACC, but because the ASAO is perfect and the standard for the systems needs the audio streams will not be re-encoded..

3.4.2 Hardware storage architecture

The initial system is very simple since it is used as a proof of concept. Basically all it needs is a hard drive with some gigabytes of space in the computer where the Flash Media Server runs. The streams are stored as regular files in a file system, not in a database. SQL managers permit the storage store files in databases, but managing such a big amount of files within a database would compromise the system's performance.

In the chapter *Evaluation and future work* I evaluate a solution to the storage scalability problem based on multiple Flash Media Servers.

Chapter 4

System Design - HearMe and TellIt

4.1 Introduction

HearMe is a Social Web Service that handles voice streams create by the user on-the-fly directly from a Web Site. The Web Service offers an external API so that other Web Sites or Web Services can easily use it to include real time voice recordings. In order have a working environment that uses *HearMe*,*TellIt* has been developed. *TellIt* is a social Web Site where users create voice streams and share in many different ways. We can consider *TellIt* as a *YouTube* clone but about voice clips instead of video clips.

This design documentation is divided in four different parts. The first part corresponds to the *Functional and Quality requirements* of both *TellIt* and *HearMe*. Because the design of the *TellIt* Web Site is not relevant to the purpose of the thesis, only its functional requirements are included in this chapter. The second part corresponds to the *Use Cases*, the third part is a general overview of the *Architecture* of *HearMe* and the fourth is its *External API*.

4.1.1 Purpose and readers

This document offers a design solution for the *HearMe* and *TellIt* systems.

The aimed readers of this document are the design team, the developer team and any other team related to the design and implementation of the system.

4.2 Requirements

4.2.1 Functional Requirements (TellIt)

1. Account-Management
   (a) Registration: A generic guest user can signup and become a registered user obtaining an account providing his/her personal data .
   (b) Edit-Details: A registered user can modify his/her personal data.
   (c) Cancel-Account: A registered user can delete his account. This will also delete all of his voice streams.
2. Friends-Management
   (a) Add-Friends: A registered user can add another user as a friend.
   (b) Remove-Friends: A registered user can delete any user from his friends list.
3. Registered-Users-Commnunication:
   (a) Private-Messaging: Registered users can send other registered users private messages.
   (b) Edit-Private-Messages: A registered user can delete any private message in his inbox.
   (c) Subscriptions: Registered users can subscribe to other registered users streams.
4. Stream-Management
   (a) Create-stream-Reg-User: A registered user can create a voice stream providing the data. This voice stream will be a part of the users stream collection.
   (b) Create-stream-Generic-User: A generic user can create a voice stream. The voice stream will be part of the anonymous user stream collection and it is forced to be a public stream. The streams data will not de editable once it has been created.
   (c) Edit-stream: The streams data of a stream created by a registered user can be edited by its creator. Anonymous streams cannot be edited.

(d) Stream-Privacy-Policy: A stream created by a registered user can be set to be either public, private, friend-shared, or user-specific-shared.

(e) Stream-Listen: A generic user must be able to listen to all of the public streams in the system. A registered user must be able to listen to all of the public streams, all of his private streams and all of his friends friend-shared streams.

(f) Stream-Ranking: Any user can rank any stream.

(g) Stream-Comment: Any user can comment any stream.

(h) Stream-Edit-Comments: The creator of a non-anonymous stream can delete any comment of any of his streams.

(i) Stream-Tag: Any user can tag any stream.

(j) Stream-Edit-Tags: The creator of a non-anonymous stream can add, edit or delete any tags associated to any of his streams.

(k) Stream-External-Embed-Policy: The creator of a non anonymous stream can permit or not the stream to be embedded from a third party website.

5. Dashboard:

   (a) View-top-ranked: View list of the top ranked streams.

   (b) View-latest-streams: View list of the latest streams.

   (c) View-subscriptions: View list of the new streams from the users subscriptions.

4.2.2 Quality Requirements (TellIt)

1. Usability: The interface with the user must be simple enough that a none experienced user may be able to use the system without the need of a previous learning process.

2. Cost: The cost for the end user must be free.

3. Availability: The *TellIt* Web Site must always be available. The uptime must be 100%.

4. Performance: The system must be fast enough to perform the tasks in real time. Users must be able to create and retrieve streams *on-the-fly*.

5. Security: The system must grant the privacy policies established by the user.

6. Accessibility: A user must be able to access the system with a Web Browser and the cross platform *Adobe Flash Player Plugin*.

4.2.3 Functional Requirements (HearMe)

1. Access-Key-Management

   (a) Request-Access-Key: A third party Web Site can request a Key to access the system's operations.

   (b) Destroy-Access-Key: A third party Web Site can destroy its Key so that it is no longer able to access to the system.

2. Clip-Management

   (a) Clip-create: A third party Web Site can embed a *HearMe* recorder application providing the access Key and the clips parameters to create a clip if the privacy policies permit it.

   (b) Clip-edit: A third party Web Site can embed a *HearMe* recorder application providing the access Key and the clips parameters to edit a clip if the privacy policies permit it.

   (c) Clip-reproduce: A third party Web Site can embed a *HearMe* player application providing the access Key and the clips parameters to play the clip if the privacy policies permit it.

   (d) Clip-edit-privacy: The creator of a clip can determine the external privacy policies. The clip can be public, password protected or private.

4.2.4 Quality Requirements HearMe

1. Usability: The interface with the user must be simple enough that a none experienced user may be able to use the system without the need of a previous learning process.

2. Cost: The cost for the third party Web Site must be free.

3. Availability: The *HearMe* system must always be available. The uptime must be 100%.

4. Performance: The system must be fast enough to perform the tasks in real time. Users must be able to create and retrieve streams *on-the-fly*.

5. Security: The system must grant the privacy policies established by the user.

6. Accessibility: A third party Web Site must be able use the system by including the necessary calls to the *HearMe* API.

4.3 Use cases

Use case 1: Clip playback

- Primary Actor: A generic user or a registered user
- Goal: Play an audio clip on a Web Site
- Preconditions: The Web Site has embedded a HearMe Player on the website with a clip identifier.
- Main Success Scenario:
  - The Web Site displays a Player Application
  - The user clips the Play button
  - The clip begins to play with a default volume if the clips is available.
  - The clips ends.

Use case 2: Clip record

- Primary Actor: A generic user or a registered user
- Goal: Record an audio clip on a Web Site
- Preconditions: The Web Site has embedded a HearMe Recorder on the Web Site with a new clip identifier.
- Main Success Scenario:
  - The Web Site displays a Recorder Application
  - The user clicks the Recorder button

  - The clip begins to be recorded if the identifier is correct.
  - The user stops speaking.
  - The user clicks on the stop button.
  - The user clicks on the play button.
  - A preview of the clip begins to play.
  - The user introduces textual information in the title, description or metadata text box.

Use case 3: Clip edit

- Primary Actor: A registered user
- Goal: Edit an already existing audio clip on a Web Site
- Preconditions: The Web Site has embedded a HearMe Recorder on the Web Site with a new clip identifier.
- Main Success Scenario:
  - The Web Site displays a Recorder Application.
  - The user clicks the Recorder button.
  - The clip begins to be recorded if the identifier is correct.
  - The old clip is overwritten.
  - The user stops speaking.
  - The user clicks on the stop button.
  - The user clicks on the play button.
  - A preview of the clip begins to play.
  - The user edits the textual information in the title, description or metadata text box.

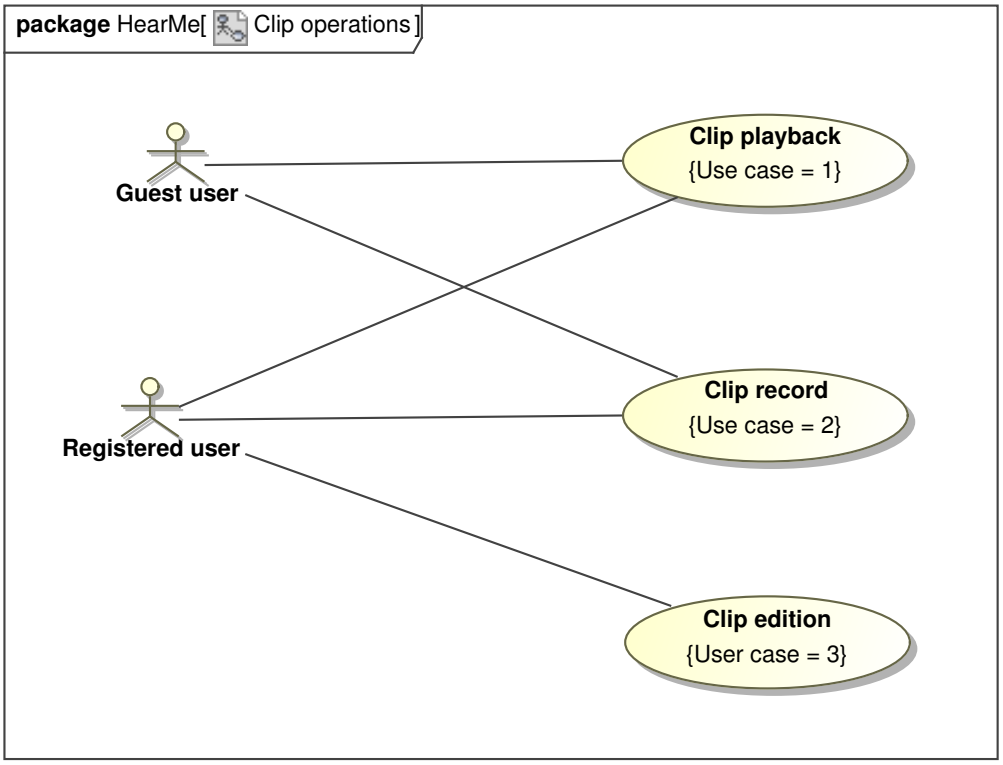

Figure 4.1: HearMe Use cases

4.4 Architecture overview

The figure 4.2 shows a basic diagram of the *HearMe's* architecture . It consists in three main parts:

- *HearMe* Player/Recorder client
- *HearMe* Web Service
- *HearMe* Stream Server

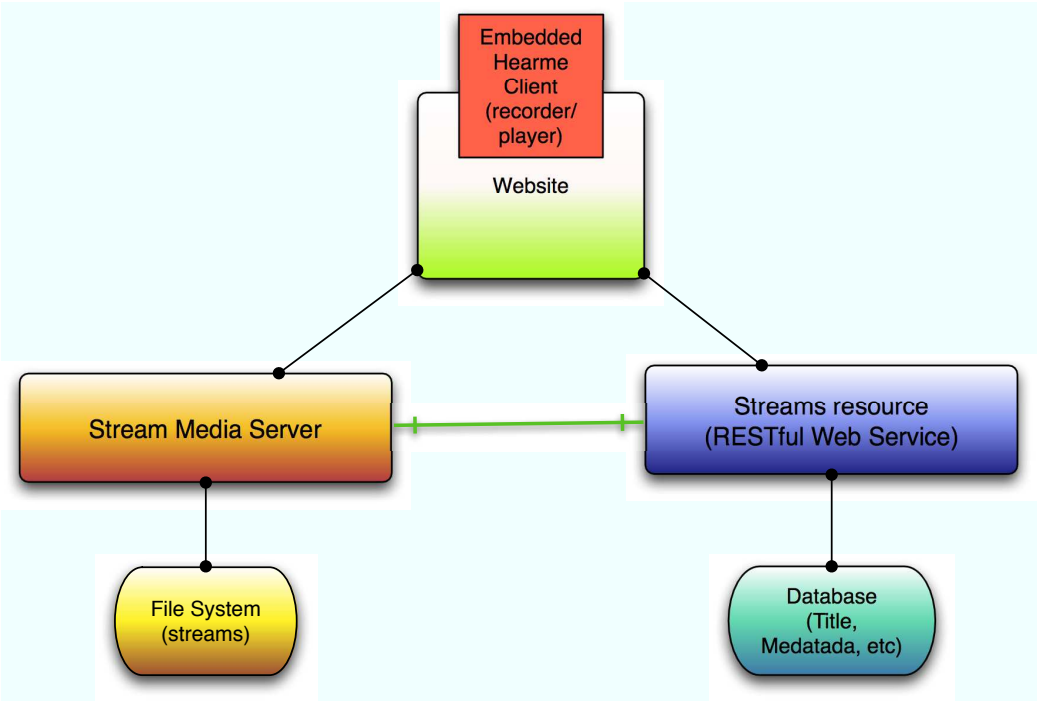

Figure 4.2: HearMe General Architecture Overview

4.4.1 HearMe Player/Recorder Client

This clients are *Flash* based web applications which are meant to be embedded in the Web Page that wants to play or record a stream. Their main task is to act as a layer between the physical hardware and the Web Application and to dialogue with the Stream Server. When recording, the recorder obtains the voice from the users microphone and streams it to the Stream Server; when playing, it retrieves the stream from the Stream server and plays it through the users audio system. This software pieces will be explained in depth in the chapters *HearMe Player* and *HearMe Recorder*.

4.4.2 HearMe Stream Server

The Stream Server is the server application which offers progressive download and upload of the clips created by the users. This piece of software retrieves, stores and makes available the audio recordings. The Recorder and Player client

connect to the Stream Server in order to publish or retrieve a stream.

In the initial version of the system only uses one Stream Server but includes partial support for using multiple. In the chapter *Evaluation and Future Work* additional study about scalability and security issues of the Stream Server are done.

4.4.3 HearMe Web Service

The *HearMe* Web Service is a RESTful Web Service which contains all of the information of the clips of the system and it is the responsible for the coherence and structure of the data. The information of a clip that holds is the following:

- Identifier
- Title
- Description
- Metadata
- Stream Server URL
- Creation time-stamp
- Last edit time-stamp

When a Web Page wants to create a clip it connects to this Web Service and requests it; the Web Service assigns a new identifier and a Stream Server for the clip. The same occurs when a Web Page requests the playback of a stream, the Web Service tells it which Stream Server to retrieve it from and sends all of its textual information to the Web Page.

An additional example of how the creation of a clip is done is included to help in the comprehension of the systems architecture:

Example: Creation of a clip Suppose that a client Web Site - *Voice Forum* - wants to include real time voice recording messages in its forum system. The page will have a form where the user can input the text that he wants to post. The page has to embed a *HearMe* recorder application in its form page. For this, first it would have to talk to the *HearMe* Web Service to obtain a stream Id and a Stream

Server url to use as parameters. With these parameters it creates an instance of a recorder application. With this parameters, the recorder application knows where to connect to stream the voice of the user and how to identify it.

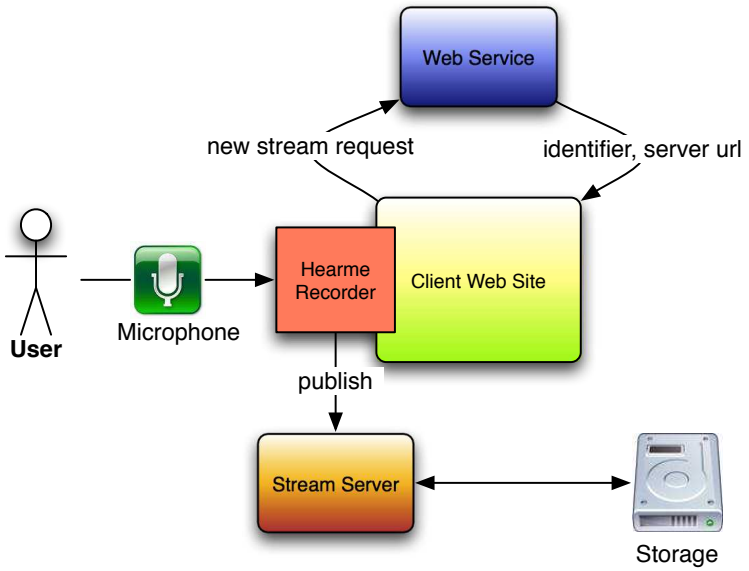

Figure 4.3: Recording process general

It will first connect to the *HearMe* Web Service to request a new stream. The *HearMe* Web Service will grant the client Web Site with a new GID (global audio identifier) and a Stream Server url. With this parameters the Web Site can create an instance of the *HearMe* recorder application, to

4.5 HearMe API

This API defines a language for clients to interact with *HearMe* from an external Web Page, Web Service or application. It is intended to be easy to implement in any language and on any platform. The protocol operates in terms of resources and operations on them and uses the standard HTTP methods (GET, POST, PUT, DELETE, etc.) to retrieve and change server state.

The protocol defines Audio resource. The operations consist of retrieving (GET), updating (PUT), creating (POST or PUT), or destroying (DELETE) these resources.

It is important to notice that this API is intended to be used with the *HearMe* Web Service. With the data obtained the external Web Application must create an instance of the *HearMe* Player or Recorder client to actually record or play the audio. The usage of the Player and Recorder client is specified in the chapters *HearMe Player* and *HearMe Recorder*.

4.5.1 Audio resource

An audio element contains the following attributes:

- Title
- Description
- Audio global identifier (GID)
- Metadata (tags)
- Server url (where the audio is physically located)
- Creation time
- Modification time

The audio is represented in XML as it follows:

```
<audio>
    <created-at type="datetime">2008-04-29T11:28:10+02:00</created-at>
    <description>This is a test audio</description>
    <gid>s151</gid>
    <metadata nil="true"></metadata>
    <server-url>rtmp://localhost/test</server-url>
    <title>Initial test</title>
    <updated-at type="datetime">2008-04-29T11:28:10+02:00</updated-at>
  </audio
```

Retrieval of audio collection

GET url/audios.xml The result of this operation is an Atom XML list of the available audio.

```
<audios>
  <audio>
    <created-at type="datetime">2008-07-28T20:12:00+02:00</created-at>
    <description>This is an audio about my cat</description>
    <gid>s183</gid>
    <metadata nil="true"></metadata>
    <server-url>rtmp://localhost/test</server-url>
    <title>Cat audio</title>
    <updated-at type="datetime">2008-07-28T20:12:00+02:00</updated-at>
  </audio>
  <audio>
    <created-at type="datetime">2008-07-30T17:16:48+02:00</created-at>
    <description>This is an audio about my dog</description>
    <gid>s184</gid>
    <metadata nil="true"></metadata>
    <server-url>rtmp://localhost/test</server-url>
    <title>Dog audio</title>
    <updated-at type="datetime">2008-07-30T17:16:48+02:00</updated-at>
  </audio>
</audios>
```

Retrieval of a single audio

GET /audios/{GID}.xml The result of this operation is an Atom XML with the data of the audio.

```
  <audio>
    <created-at type="datetime">2008-07-28T20:12:00+02:00</created-at>
    <description>This is an audio about my cat</description>
    <gid>s183</gid>
    <metadata nil="true"></metadata>
    <server-url>rtmp://localhost/test</server-url>
    <title>Cat audio</title>
    <updated-at type="datetime">2008-07-28T20:12:00+02:00</updated-at>
  </audio>
```

Creation of an audio

POST /audios.xml This operation creates a new audio. It takes the parameters specified in the request to fill the new audio's data.

```
<request>
<audio>
     <title>Example title</title>
        <description>Example description</description>
```

```
        <metadata>Example tag</metadata>
    </audio>
<request>
```

It returns an XML with the information of the created audio:

```
  <audio>
    <created-at type="datetime">2008-07-28T20:12:00+02:00</created-at>
    <description>Example description</description>
    <gid>s183</gid>
    <metadata">Example tag</metadata>
    <server-url>rtmp://localhost/test</server-url>
    <title>Example title</title>
    <updated-at type="datetime">2008-07-28T20:12:00+02:00</updated-at>
  </audio>
```

Edition/update of an audio

PUT /audios/{GID}.xml This operation updates the attributes of an already existing audio. It takes the parameters specified in the request to update the audio's data.

```
<request>
<audio>
     <title>New title</title>
        <description>new description</description>
        <metadata>New Tag</metadata>
    </audio>
<request>
```

It returns an XML with the new audio's data:

```
  <audio>
    <created-at type="datetime">2008-07-28T20:12:00+02:00</created-at>
    <description>New description</description>
    <gid>s183</gid>
    <metadata>New tag</metadata>
    <server-url>rtmp://localhost/test</server-url>
    <title>New Title</title>
    <updated-at type="datetime">2008-07-28T20:12:00+02:00</updated-at>
  </audio>
```

Delete an audio

DELETE /audios/{GID}.xml This operation destroys the audio specified by a GID.

The system does not support authentication or authorization. The usage of external application KEYS to provide security and control is evaluated in the chapter *Evaluation and Future Work*. If the system already supported it the only change in the API would be that in each request the KEY parameter would have to be specified.

Chapter 5

HearMe Player

The *HearMe* Player application provides the user of an interactive reproduction of an audio clip. The Player is a *Flash* based application which is embedded on a Web Page provided and identifier of the clip to play. The application connects to the Stream Server to retrieve a sound clip and plays it on the Web Site.

5.1 Interface

The interface is designed to be clear and simple in order to match the usability criteria of the quality requirements. A user who has never seen the application should be able to use it right away without learning anything previously. The GUI capabilities are:

- Provides the user the ability to play, stop and seek through an audio clip.
- Display the audio clips time information.
- Link to the clips Web Page and to HearMe Web Site.
- Volume playback control.

As it is shown in figure 5.1, the interface has two buttons to control the reproduction of the clip, as well as a progress bar to display the percentage of the clip which has been downloaded. This progress bar is also clickable so that it provides with seeking functionality. The GUI contains two additional buttons, one of them links to the clips website and the other one to the *HearMe* Home Page. Additionally it contains a display area with information about the duration of the clip

and its current position in time. The interface has some animations to increase its usability. When the user places the cursor over the buttons, it shows a small glow effect which makes him realize that the cursor it is properly placed. When the user clicks play or stop, the color of the button shades so it looks like it has been pushed.

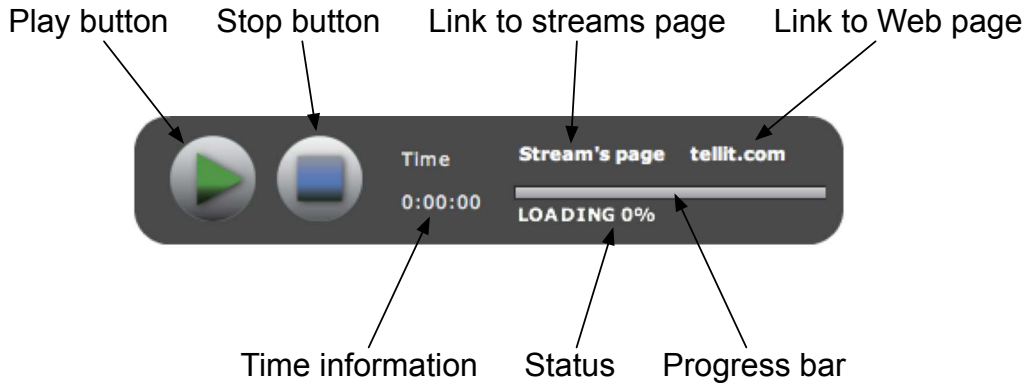

Figure 5.1: HearMe Player Interface

5.2 Player Architecture

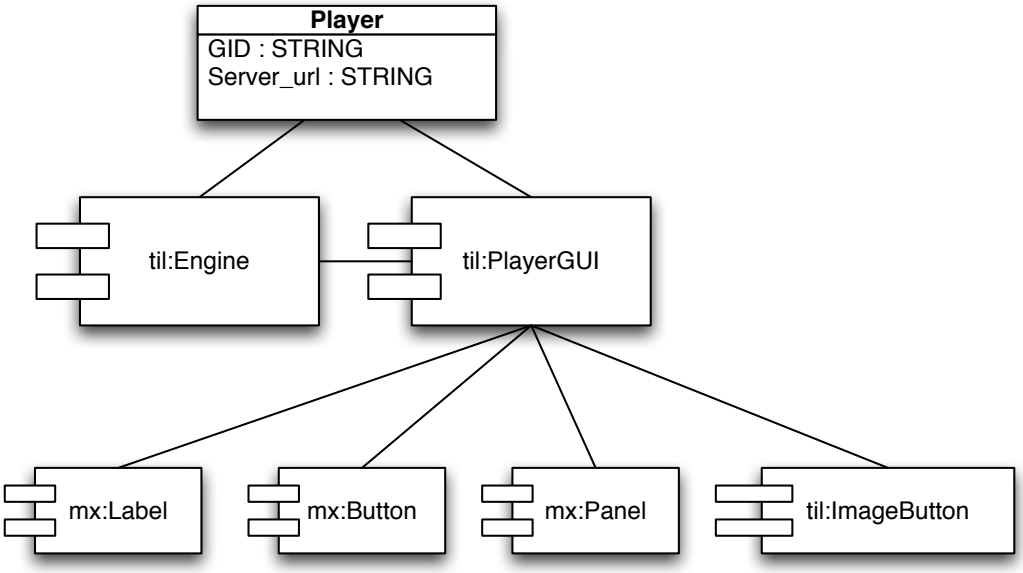

Figure 5.2: HearMe Player Class and Component diagram

The Player retrieves and plays the clip from the Stream Server. In order to perform this taskm it needs to dialog with the Web Service to get the clips information and then it connects to the Stream Server to recover the stream. The Web Page which embeds it, provides the Player with the clips identifier (GID). The player takes this identifier and queries the Web Service, if the clip exists and it privacy policies permit it, the Web Service sends back to the Player the clips information. This information usually contains the Stream Server address where the clip is physically located and some additional textual information such as Tags, Name and Description. Once the Player has all of the information it needs, it connects to the Stream Server and requests the streaming of the clip. If everything is correct, a *NetStream* connection between the Stream Server and the Player is established and the player begins to reproduce the clip.

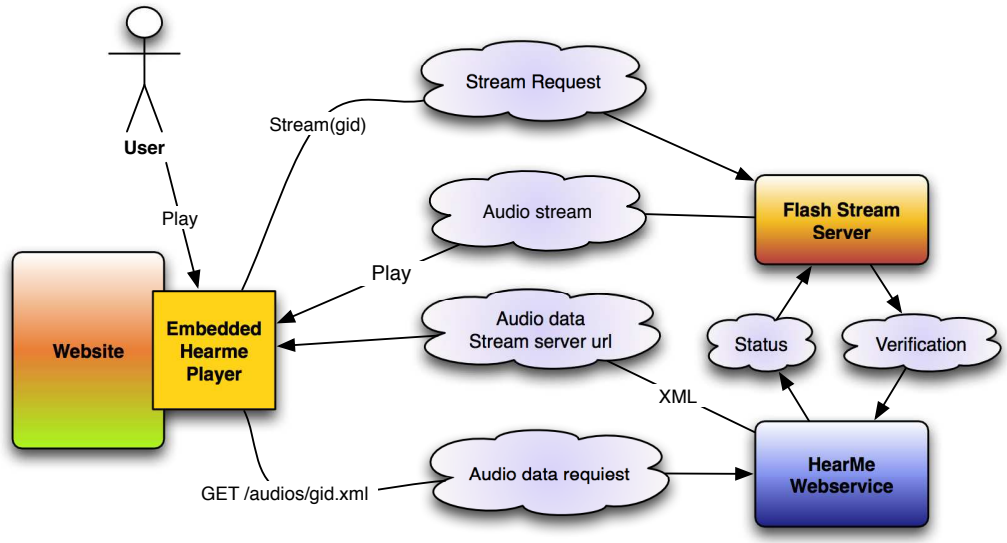

Figure 5.3: HearMe Player work flow

5.3 Web Page integration

The *HearMe* player is instanced in a Web Site with the OBJECT tag, which is normally used to include objects such as images, videos, Java applets, and Flash applications. In this tag the site sets the identifier of the clip as a parameter of the Player. It is important to realize that the Web Site needs to manage the clips identifiers as seen in the following example

Suppose a Web Site which is a voice forum. The Site might have in a single page many instances of the *HearMe* player, each one of these instances corresponds to a message that a user has created, but the *HearMe* doesn't know which one belongs where. Therefore, in order to use the clips properly the site must manage the identifiers, store them and organize them so that each instance of the Player corresponds to the message the site wants to show.

An example of how to embed the Player is as it follows:

```
<object width="400" height="74">
<param name="movie" value="tellitplayer.swf" />
<param name="wmode" value="transparent" />
<embed wmode="transparent"
src="http://localhost:3001/flash/tellitplayer.swf?gid=s161
```

```
type="application/x-shockwave-flash"
width="400"
height="74">
</embed>
</object>
```

The first two parameters, *width* and *height*, correspond to the size of the player. The parameter *movie* specifies the program filename and *wmode* is to set the player graphics as transparent for the rounded borders. After that we have the *embed* tag which specifies the url of the player application, the *mime* type and again the size. It is in the *src* parameter where the website must include the streams identificator as it can bee seen in the example.

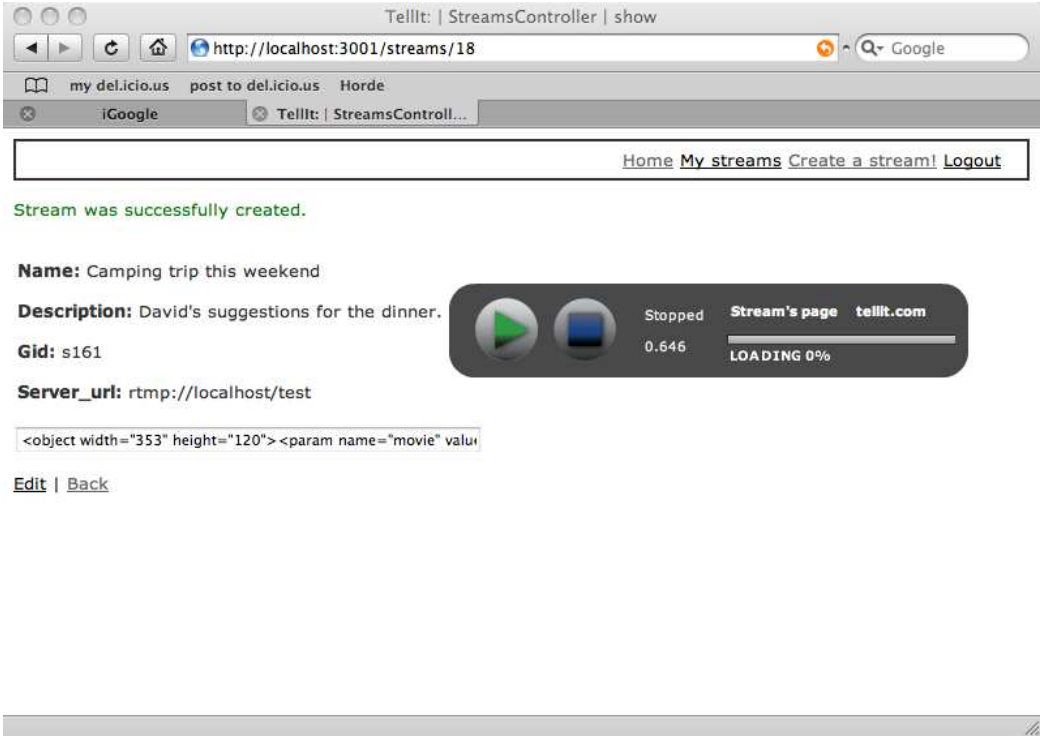

Figure 5.4: HearMe Player Embedded

Chapter 6

Hearme Recorder

The *HearMe* Recorder application provides the user with the tools to record a voice clip. This application connects to the Stream Server to publish and store the sound captured by the user's microphone.

The Recorder is a Flash based application meant to embedded on a Web Page.

6.1 Interface

The same way the Player interface, the Recorder interface is designed to be clear and simple for usability concerns. A user who has never seen the application before should easily be able to use it right away without having to learn anything previously. The GUI capabilities are:

- Provides the user the ability record and preview a clip.
- Displays the audio clips time information.
- Select from the different possible inputs or microphones.
- Volume recording control.

As it is shown in figure 6.1, the interface has three buttons to control the recording and playback of the clip. It also contains a display area with information about the duration of the clip, its current position in time and network connection information. The same way as the Player, this interface has some animations to increase its usability. When the user places the cursor over the buttons, it shows a small glow effect which makes him realize that the cursor it is properly placed.

When the user clicks play or stop, the color of the button shades so it looks like it has been pushed.

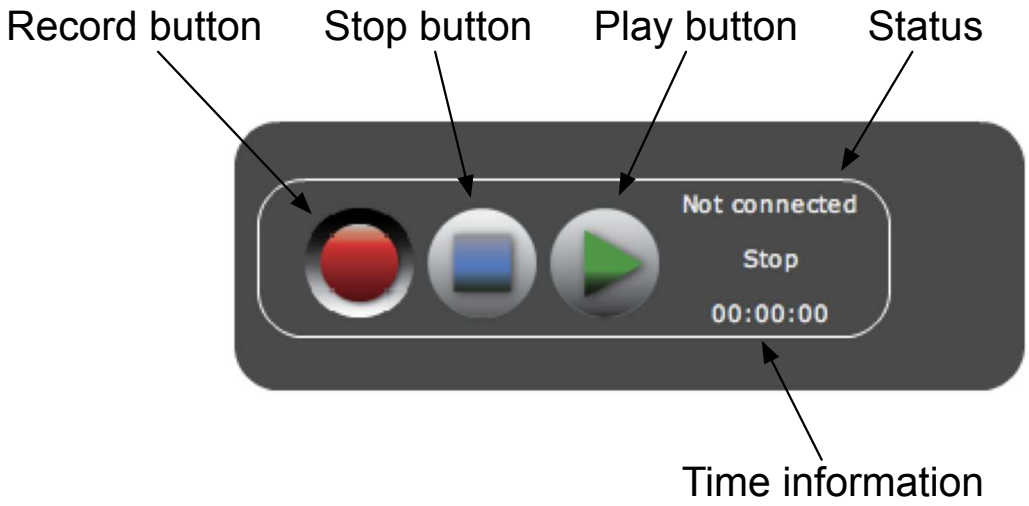

Figure 6.1: HearMe Recorder Interface

6.2 Recorder Architecture

The Recorder application is divided in three components.

- Engine
- RecorderGUI
- Recorder

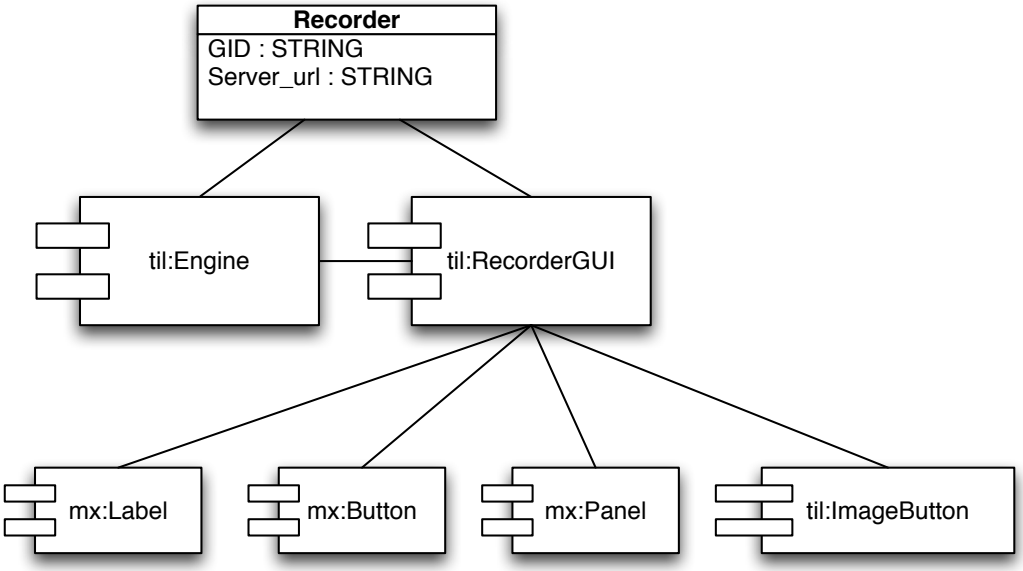

Figure 6.2: HearMe Recorder Class and Component diagram

The Recorder publishes a clip on the Stream Server which is stores it on its file system. This operation is slightly more complicated than the one performed by the Player. In this case, the Web Site which is embedding the Recorder first has to dialogue with the Web Service to obtain the new clips identifier (GID) and a Stream Server url. It will do this by a HTTP request (detailed in the API section) and the Web Service will return a XML file with the information. With this information the Web Page embeds a Recorder which knows where to connect and how to identify the clip it is going to publish. Once the Web Site has properly embedded the Recorder, the user can click on the record button and the Recorder will ask the user for permission to read the microphone (this is inevitable due to the Flash security and privacy policy). After this, the Recorder connects to the Stream Server and requests to publish the stream. If the Stream Server validates it, a *NetStream* connection between both is established and the sound will begin to be stored on the Stream Server's file system. The user can also play the clip it just created, in that way the Recorder behaves the exact same was as the Player

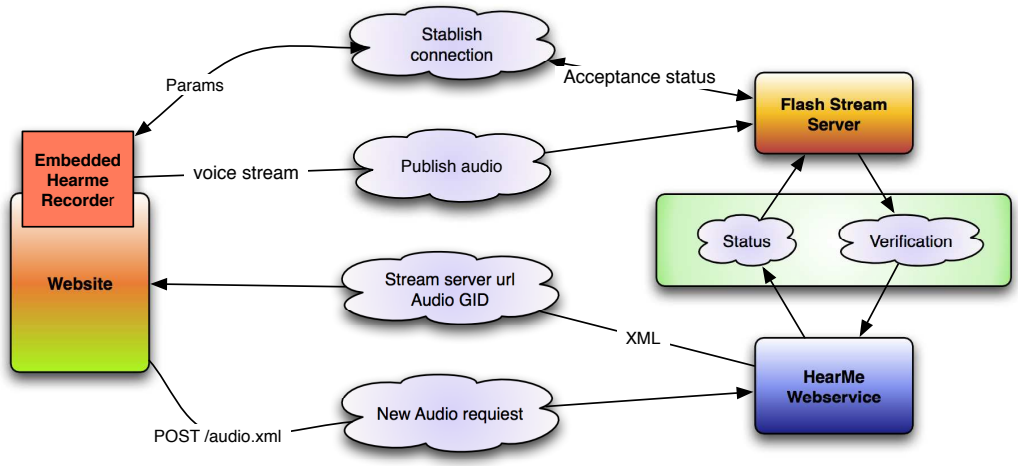

Figure 6.3: HearMe Recorder work flow

6.3 Web Page integration

The *HearMe* Recorder is instanced in a Web Site with the OBJECT tag, which is normally used to include objects such as images, videos, Java applets, and Flash applications. In this tag the site sets the identifier of the clip and the stream server url as parameters of the Recorder. The same way as in the Player, it is the Web Site who has to manage the identifiers (GID)

Example of embedding a Recorder:

```
<object width="353" height="140">
<param name="movie" value="tellitrecorder.swf" />
<param name="wmode" value="transparent" />
<embed wmode="transparent"
src="http://localhost:3001/flash/tellitrecorder.swf?gid=s185&server=rt
type="application/x-shockwave-flash"
width="353"
height="140">
</embed>
</object>
```

The *width* and *height* parameters correspond to the size of the Recorder. The parameter *movie* specifies the program filename and *wmode* is to set the recorders

graphics to be transparent, for the rounded borders. The *embed* tag specifies the url of the Recorder application, the *mime* type and again the size. In the *src* parameter the website must include the streams identifier and the Stream Server url.

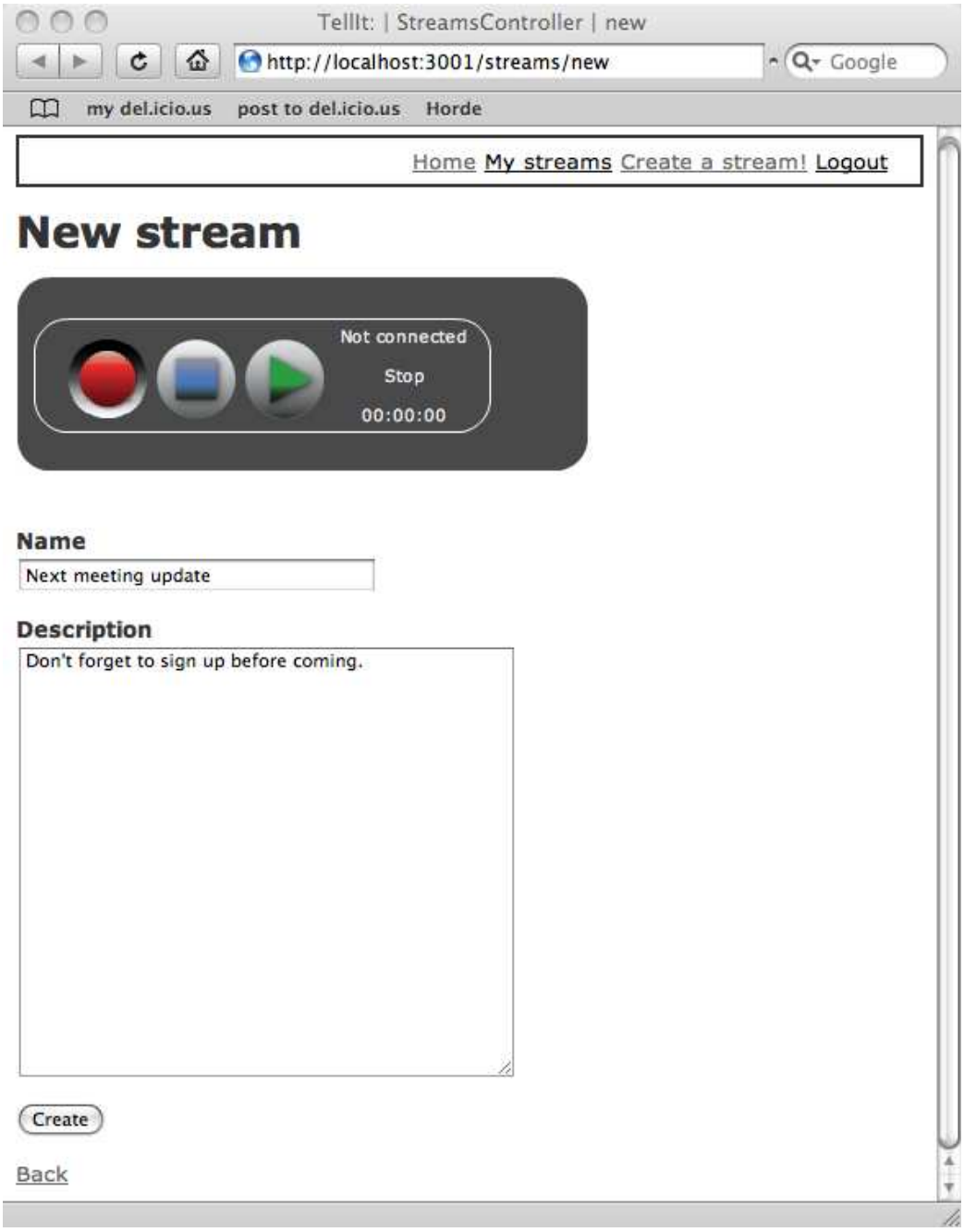

Figure 6.4: HearMe Recorder Embedded

Chapter 7

Evaluation and future work

7.1 Evaluation

HearMe is designed to be used by third party Web Sites so they can include voice recordings in real time for creating content . The basic features like playing, creating or editing a voice clip are implemented, but some other advanced features were not, the most noticeable are the ones that deal with security and privacy. Technically the Stream Server should dialog with the Web Service to make sure that the Web Site which wants to record or play a clip has the permission to do so. This does not happen at this point, the Stream Server directly permits to record or play any stream and does not contact the Web Service to check if all of the parameters are correct and valid. So currently anyone can retrieve any stream from the Stream Server, even private ones. Also the usage of a Key to validate if a Web Site has access to the system is not implemented, so any external Web Site can use it without any restrictions.

The Player and Recorder applications have some minor bugs which were not fixed. The progress bar of the player does not work, and neither does the seeking. The applications can crash if the Stream Server is not available. Volume feature is implemented but there is no graphical component to control it. The recorder should permit the user to choose between the different inputs he might have (microphone, line in, etc) but it does not yet implement this feature.

Voting and ranking of the clips are not implemented in the *TellIt* community Web Site (explained in the next section) and the page does not yet offer feed syndication to the most popular or recent clips.

A more detailed status of the development of *HearMe* and *TellIt* can be found

on tables 7.1 and 7.2. In these two tables it can be seen which ones of the functional requirements of the applications where fully implemented in the final version of the software for the thesis. The non implemented features are left for future work in case the system ever goes on a real productivity environment.

Functional Requirement	Status
Account Management	
Registration	Implemented
Edit-details	Implemented
Cancel-account	Implemented
Friends-management	
Add-friends	Not implemented
Remove-friends	Not implemented
Registered-Users-Communication	
Private-messaging	Not implemented
Edit-Private-Messages	Implemented
Subscriptions	Not implemented
Stream-Management	
Create-stream-reg-user	Implemented
Create-stream-generic-user	Implemented
Edit-stream	Implemented
Stream-privacy-policy	Not implemented
Stream-Listen	Implemented
Stream-Ranking	Not implemented
Stream-comment	Not implemented
Stream-edit-comments	Not implemented
Stream-tag	Implemented
Stream-edit-tags	Implemented
Stream-External-Embed-Policy	Not implemented
Dashboard	
View-top-ranked	Not implemented
View-latest-streams	Implemented
View-subscriptions	Not implemented

Table 7.1: TellIt implementation status

Functional Requirement	Status
Access-key-management	
Request-access-key	Not implemented
Desrtoy-acess-key	Not implemented
Clip-management	
Clip-create	Implemented
Clip-edit	Implemented
Clip-reproduce	Implemented
Clip-edit-privacy	Not implemented

Table 7.2: HearMe implementation status

7.1.1 Deployment experiences

TellIt voice community

The most practical way to test *HearMe* was by creating a Voice sharing Web Site that used it. *TellIt* uses the *HearMe* API to create a social Web Site about voice clip recordings. *TellIt* is a very simple Site where users can create voice clips to share them. We can picture *TellIt* as a *youtube.com* or a *flickr.com* of voice recordings. *TellIt* also includes some additional features like a dashboard with the latest clips or the most popular.

TellIt was developed almost at the same time as *HearMe*. It uses the *HearMe* API to provide its users with voice recording functionality. When *HearMe* was in an early stage of development it was not certain what external Web Sites would need to include it. *TellIt* started to use it, but the initial tests did not work as expected. The API was not completely defined or implemented, there were many problems embedding the Player and Recorder applications and it was not very clear how to attach the whole system. Very slowly and carefully both pieces grew together until they coupled.*TellIt* follows the steps detailed in the chapters *HearMe API*, *HearMe Recorder* and *HearMe Player*.

TellIt was developed with *Rails* technology. It manages two resources, users and streams.

Users are the persons who are registered on the site, they can create their own streams, edit or delete them. Generic users can also create streams but once they have created them, these cannot be modified.

The stream resource contains the clips created by the users. The resource is RESTful so it can be accessed both through the *TellIt* Web Site or as a Web Service.

Two additional features where studied to implement in the future, both related to *Google's OpenSocial*. The first one was to expose the users resource as an *OpenSocial* container so that other *OpenSocial* apps could connect to *TellIt* and use the site's users data. The second one was to create a collection of *OpenSocial gadgets* so that it could easily be included into other *OpenSocial* sites. As said, both features were left for future work.

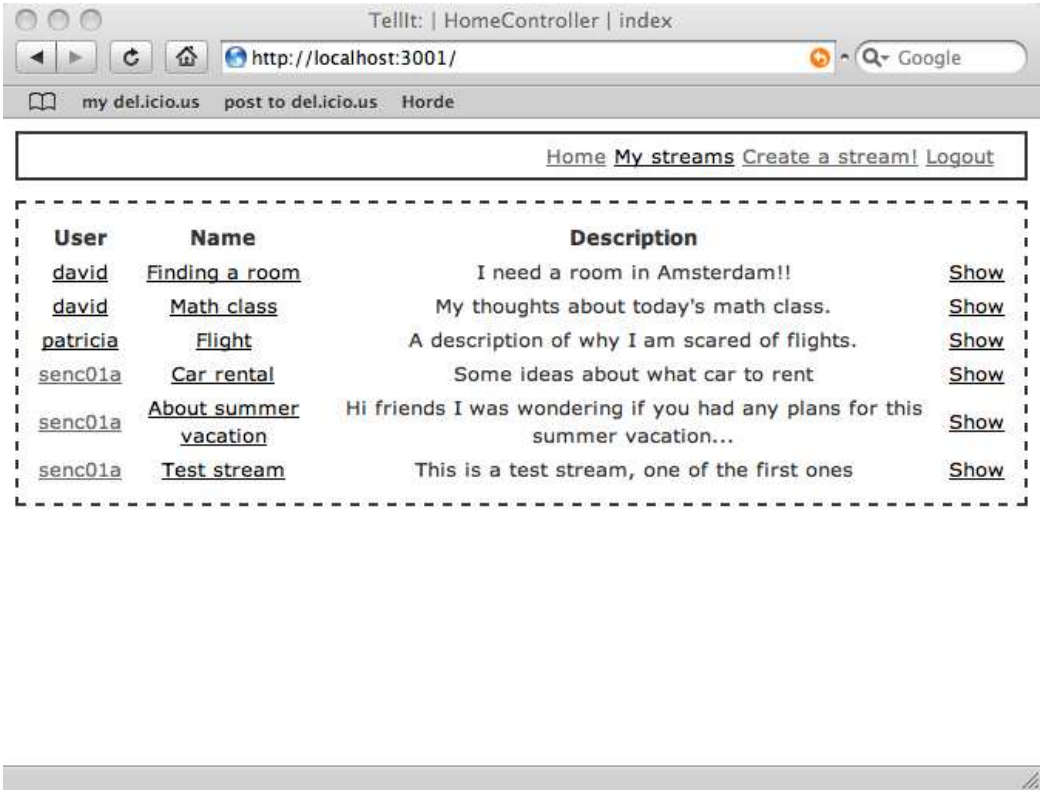

Figure 7.1: TellIt

The test system was deployed in a personal machine. Two *Mongrel* Web Servers had to run, one for *HearMe* and one for *TellIt* and also the *Red5* flash media server. The machine is a rather small machine, about 2 GHz of processing speed with 2 Gigabytes or RAM. This machine was enough to test do the basic testing of the system but it was not able to test it in a wide productive scale. For

this task a server machine with a very fast internet connection would have been needed. With such machine the system could have been tested in a wider scale, including it in small forum communities or blogs.

7.2 Future work

In this section I study some technical problems that could appear, as well as some of the enhancements or extensions that could be done on *HearMe*.

7.2.1 Rich recorder client

HearMe is all about voice and recordings. The initial Recorder client is very simple, with it you can only record a clip all the way from the beginning, having to start over again if you want to make any changes and it does not support to pause while recording. At the same time, it can only record one track per stream and the user cannot include music in the background or anything that does not go through the microphone. This means that he cannot upload sound files. This situation is not a big problem initially, but if *HearMe* was really being used a *Rich Recorder Client* would be needed.

Figure 7.2 shows us a screenshot of a very popular Sound Editor called *GarageBand* from Apple corporation. *HearMe* would need a simplified version of one of these type of applications that supported at least the following operations:

- Complex edition: Record, stop, pause, partial removal, etc.
- Multitrack support.
- Upload of sound files.
- Basic Sound filters.
- Save, stop and resume of the edition.

This application would have to be a Web based Sound Editor and fortunately it can be done with *Adobe Flex*, the technology used to develop the initial Recorder Client.

Figure 7.2: Apple Garage Band Screenshot

7.2.2 Scalability

Scalability is a big concern in *HearMe*. Web Sites usually start with a very small amount of users but when they get popular this amount can grow many times in very small periods of time. The system must be designed to fulfill the markets needs, no matter the number of users.

Web Service

The *HearMe* Web Service only has to deal with regular Web Service operations which are quick, small and simple. Its scalability concerns are the same as any other *Ruby on Rails* Web Site. A good example of large scale deployment with *Rails* technology is *Twitter.com*. This Web Site has to process millions of requests per day and it had some scalability problems, Alex Payne explains in an interview[14] how they solved them:

Rails is a matter of cost: just throw more CPUs at it. The problem is that more instances of Rails (running as part of a Mongrel cluster, in our case) means more requests to your database. At this point in time theres no facility in Rails to talk to more than one database at a time. The solutions to this are caching the hell out of everything and setting up multiple read-only slave databases, neither of which are quick fixes to implement.

Indeed, *Rails* applications can suffer from scalability problems but as we have seen they can be fixed, and furthermore, most of these problems will be solved soon in the core of the framework, so no manual fixes will need to be developed.

Stream Server

The Stream Server is a total different thing. When dealing with multimedia contents the amount of bytes to transmit and the amount of time needed by a computer to process it grows exponentially.

In a global scale, the number of bytes to transmit depends on the following factors:

- Stream Bitrate
- Average time per clip
- Number of clips created or reproduced per day

Table 9.1 in the Appendix section shows a comparison of the total amount of data transmitted per day, depending on those factors.

Supposing the service is being used by a very small community that creates an average of 60 seconds per clip, recording 50 clips per day and playing 200, the system would only have to transmit about $13.18 + 52.7 = 65.88$Mbytes per day. In this case the bandwidth of the Stream server does not have to be big, an average of 0.001 MBPS for the downloads and 0.005 MBPS for the downloads. Almost any connection can hold this, even a home connection. It is also important to notice that the measurements of MBPS are an average through the whole day, but at certain hours of the day there would be peaks of at least four or five times the average. With this small amount of users pike would not be a problem. If there were amount of 50000 clips created daily and 200000 played, a symmetric connection of 10MBPS could struggle . Further away, we can suppose a situation where the *HearMe* was massively being use in a large scale. Suppose that everyday about 50 million voice clips where created and 200 reproduced. In this case we would need a connection of 12 GBPS of download capacity and 48 GBPS for upload in a hypothetic situation where there were no peaks. By looking at this numbers we can assure that at some point a single server for the Stream Server will not be enough. This is as evident with bandwidth as it is with disk space and with processing capacity, therefore the system must support some way to scale the Stream Server. The initial design supports more than one Stream Server, but it does not fully implement it. Policies about how and where to store the files and distribution of the work load between the different servers would need further in depth studies. This topic is complex that it would represent a new project itself.

Although all of the mentioned above is true, some small tweaks can be easily develop to reduce the server load. One of these tweaks could be to support dif-

ferent qualities of streaming when the users create the voice clips. This quality is constantly set to 9 (view table 7.3). An improvement would be to dynamically change this depending on by the users internet quality, the server load or the users choice. The Recorder client could perform a speed test with the Stream Server and depending on its results vary the streaming quality.

Quality value	Quality bitrate (Kbps)
0	3.95
1	5.75
2	7.75
3	9.80
4	12.8
5	16.8
6	20.6
7	23.8
8	27.8
9	34.2
10	42.2

Table 7.3: Encode quality and bandwidth

Chapter 8

Summary and conclusions

The aim of this thesis was to create a software system that would allow users to create real time voice clips on the Web. The system had to be designed in a way that any already existing Web could include it and allow its users to hear and be heard freely around the site.

The first goal to study was if the concept itself would actually have any interest in the real world. Why would anyone want to leave his voice on a Web Page and what possible applications could be created using peoples voice clips. The results of this study was that voice, comparing it to video or text, is a really strong and powerful way of communication, not better or worse than any of the others, but simply complementary. Voice clips should be used as an additional type of content to enhance the users experience and the Web Site value. The applications of this technology are yet to be discovered, but some examples could be voice forums, voice comments in blogs, gadgets for social networking applications, voice short messaging *(voicetwitter*) and on. Its applications may be very interesting in the field of electronic business where the confidence between buyer and seller is the key to the market dominance. Permitting sellers and buyers to send each other voice messages could be very productive in that way, mainly because voice messages turn out to be more trustable than text ones for psychological reasons.

Once that it was clear that the idea was worthwhile to develop I had to deal with more technological aspects of its implementation. For this task I had to read about the current philosophy of the web, known as the *Web 2.0*, about social networking and learn about the technologies that Web Sites use communicate one with others. Furthermore studies about multimedia on the Web had to be done. The outcome of this time was the chapter *Technology Survey* and a draft of the systems software architecture.

At that point the design and implementation of the system began, what required additional in depth study about the technologies used in the project, such as *Rails*, *Flash*, *Flex*. There were many complications in the beginning , specially in putting all of the pieces together, because the system itself has many parts, each one is rather simple but connecting them all in a proper way was not an easy task. After all, a working version was developed which brought new concerns, most of the about security, privacy and scalability. These concerns are exposed in the chapter *Evaluation and Future Work*.

In a personal aspect the project was very challenging. I was never given the change before to develop a personal idea, specially not such an ambitious one. There was really not too much written about something similar and I was not even sure if it was technologically possible or viable to do it. There were many moments where I was stuck and was not sure about how to advance. Mostly thanks to my supervisor and with some time most of the problems got fixed. At the end, the project was functional which was very pleasant to see, although a lot of work was left over for the future.

Bibliography

[1] Peter Armstrong. *Flexible Rails*. Manning, second edition, 2008.

[2] Marcus Baguley. 10 reasons why - ruby on rails. 2000.

[3] Daniel Choi. Ruby on rails activerecord. 2008.

[4] Red5 community. Flash video specs, open source flash. 2007.

[5] Wikipedia Community. Actionscript, wikipedia.

[6] Wikipedia Community. Adobe flash, wikipedia.

[7] Wikipedia Community. Adobe flex, wikipedia.

[8] Wikipedia community. Nellymoser asao codec. 2007.

[9] David Heinemeier Hansson Dave Thomas. *Agile Web Development with Rails*. The Pragmatic Bookshelf, 2005.

[10] Francis McCabe Eric Newcomer Michael Champion Chris Ferris David Orchrad David Booth, Hug Haas. Web services architecture. 2004.

[11] Roy Thomas Fielding. Architectural styles and the design of network-based software architectures. 2000.

[12] Walt Howe. A brief history of the internet. 2007.

[13] H. Kreger K. Gottschalk S. Graham and J. Snell. Introduction to web services architecture. 2002.

[14] Bob Kenzer. 5 question interview with twitter developer alex payne. 2008.

[15] Anne Thomas Manes. Web service basics. 2003.

[16] Yukihiro Matsumoto. The ruby programming language. 2000.

[17] Sun Microsystems. Using web services effectively: Web service processing and interaction models. 2002.

[18] Tim O'Reilly. What is web 2.0. 2005.

[19] Thomas Baustert Ralf Wirdemann. *RESTful rails development*. b-simple, 2007.

[20] Yihong-Ding. A simple picture of web evolution. 2007.

Chapter 9

Apendix

9.1 Source code

The source code can be found on a digital format at: http://www.senc.yoteinvoco.com/project/public/sourc

9.2 HearMe system

Rails init script output

```
>rails school
create
    create  app/controllers
    create  app/helpers
    create  app/models
    create  app/views/layouts
    create  config/environments
    create  config/initializers
    create  db
    create  doc
    create  lib
    create  lib/tasks
    create  log
    create  public/images
    create  public/javascripts
    create  public/stylesheets
```

```
create  script/performance
create  script/process
create  test/fixtures
create  test/functional
create  test/integration
create  test/mocks/development
create  test/mocks/test
create  test/unit
create  vendor
create  vendor/plugins
create  tmp/sessions
create  tmp/sockets
create  tmp/cache
create  tmp/pids
create  Rakefile
create  README
create  app/controllers/application.rb
create  app/helpers/application_helper.rb
create  test/test_helper.rb
create  config/database.yml
create  config/routes.rb
create  public/.htaccess
create  config/initializers/inflections.rb
create  config/initializers/mime_types.rb
create  config/boot.rb
create  config/environment.rb
create  config/environments/production.rb
create  config/environments/development.rb
create  config/environments/test.rb
create  script/about
create  script/console
create  script/destroy
create  script/generate
create  script/performance/benchmarker
create  script/performance/profiler
create  script/performance/request
create  script/process/reaper
create  script/process/spawner
```

```
create  script/process/inspector
create  script/runner
create  script/server
create  script/plugin
create  public/dispatch.rb
create  public/dispatch.cgi
create  public/dispatch.fcgi
create  public/404.html
create  public/422.html
create  public/500.html
create  public/index.html
create  public/favicon.ico
create  public/robots.txt
create  public/images/rails.png
create  public/javascripts/prototype.js
create  public/javascripts/effects.js
create  public/javascripts/dragdrop.js
create  public/javascripts/controls.js
create  public/javascripts/application.js
create  doc/README_FOR_APP
create  log/server.log
create  log/production.log
create  log/development.log
create  log/test.log
```

Bandwidth comparison table

Amount of users	Direction	Duration	Kbps	Clips/day	Size (Mbytes)	Size (Gbytes)	MBPS
Very small	Download	60	36	50	13.18	0.01	0.001
	Upload	60	36	200	52.73	0.05	0.005
Small	Download	60	36	500	131.84	0.13	0.012
	Upload	60	36	2000	527.34	0.51	0.049
Optimistic	Download	60	36	50000	13,183.59	12.87	1.221
	Upload	60	36	200000	52,734.38	51.50	4.883
Very optimistic	Download	60	36	5000000	1,318,359.38	1,287.46	122.070
	Upload	60	36	20000000	5,273,437.50	5,149.84	488.281
Nonsense	Download	60	36	500000000	131,835,937.50	128,746.03	12,207.031
	Upload	60	36	2000000000	527,343,750.00	514,984.13	48,828.125
Very small	Download	120	36	50	26.37	0.03	0.002
	Upload	120	36	200	105.47	0.10	0.010
Small	Download	120	36	500	263.67	0.26	0.024
	Upload	120	36	2000	1,054.69	1.03	0.098
Optimistic	Download	120	36	50000	26,367.19	25.75	2.441
	Upload	120	36	200000	105,468.75	103.00	9.766
Very optimistic	Download	120	36	5000000	2,636,718.75	2,574.92	244.141
	Upload	120	36	20000000	10,546,875.00	10,299.68	976.563
Nonsense	Download	120	36	500000000	263,671,875.00	257,492.07	24,414.063
	Upload	120	36	2000000000	1,054,687,500.00	1,029,968.26	97,656.250

Table 9.1: Encode quality and bandwidth