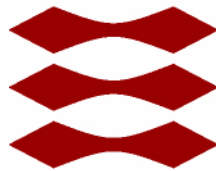


IoCast

**Bring Participatory Journalism Within A
Smartphone's Reach**

Fabrizio Becci

DTU



Kongens Lyngby 2013
COMPUTE-M.Sc.-2013-38

COMPUTE-M.Sc.-2013-38

ISSN: 0909-3192

ISBN: XXXXXXXXXX

Technical University of Denmark

DTU COMPUTE

Department of Applied Mathematics and Computer Science

Matematiktorvet, DK-2800 Kongens Lyngby, Denmark

Phone +(45) 45 25 33 51

Phone +(45) 45 88 26 73

compute@compute.dtu.dk

www.compute.dtu.dk

Acknowledgements

To all the beautiful and terrible moments I have shared with my friends, my real friends. To those too many words of my mother and sister, and those too few of my father and brother, that helped me staying alive the past four years of my life.

To the inspiring teaching of Professors Michael Kai Petersen and Peter Olaf Looms.

København, May 30 2013.

Table of Contents

	Page
Acknowledgements	i
Table of Contents	iii
1 Introduction	1
1.1 Motivation	1
1.2 Project Goals	2
2 Related Works	5
2.1 Mobile Video Streaming	5
2.2 Web Radios	6
2.3 Web TV	7
3 Analysis	9
3.1 User Hierarchy	10
3.2 Supporting Technologies	11
3.3 Event Models	13
3.4 Scenario Models	14
3.5 Use Cases	16
3.6 Requirements	17
4 Design	25
4.1 Mobile application	25
4.2 Web TV Website	27
5 Implementation	37

6	Tests	43
6.1	Mobile Application	43
6.2	Web TV Website	44
7	Discussion	47
A	A brief business plan	51
A.1	Background	51
A.2	Business Idea	52
A.3	Value proposition	52
A.4	Customer profile	53
A.5	Market	53
A.6	Competitors and Strategic Partners	53
	Bibliography	55

CHAPTER 1

Introduction

1.1 Motivation

There are three ways to look at how society is informed. The first is that people are gullible and will read, listen to, or watch just about anything. The second is that most people require an informed intermediary to tell them what is good, important or meaningful. The third is that people are pretty smart; given the means, they can sort things out for themselves, find their own version of the truth.

This is what Dale Peskin, co-Director of *The Media Center* at The American Press Institute, has written in 2003 for the introduction of an article entitled *We Media - How audiences are shaping the future of news and information*[8]. I consider myself part of the third group of people that Peskin is talking about. As being myself involved in the technical direction of shows for a Internet Web Radio¹, it is quite important to me to have the best solution for *finding my own version of the truth*, or better, *telling* my own version of the truth. By exploring the context of Internet Web Radios, and by analyzing the behavior of the users within such context, I will - in this project - present an *ad-hoc* solution for producing information using smartphones where the production

¹See www.radiopizza.net

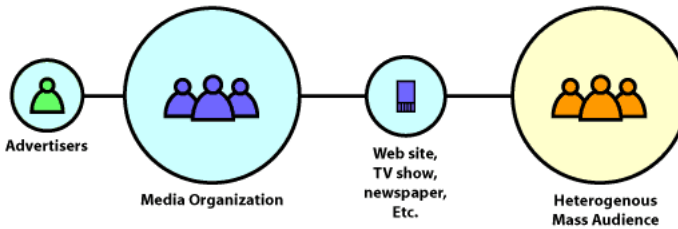
process is moderated by users, and information is distributed via a wide range of media systems, including the very same smartphones, the Web, Web TV and the common DVB, digital video broadcasting.

1.2 Project Goals

The goal of this project is to present an *ad-hoc* solution for producing and consuming information within the context of *Participatory Journalism* using smartphones as the main media, together with the Web, Web TV and DVB. The intent of such *participation* is to provide independent, reliable, accurate, wide-ranging and relevant information that a democracy requires[8]. The concept of participatory journalism is well described in [8]. It is also called *bottom-up* news model. Figure 1.1 shows how it differs from the classic *top-down* model.

Broadcast: Top-down news

Model also called transmit, push. Characterized by media organization control. All news is filtered through organization before getting to audience.



Intercast: Bottom-up news

Also called peer-to-peer, social network. Participants are peers and have ability to change roles. News is often unfiltered by a mediator before getting to its audience.



Figure 1.1: *Bottom-up and top-down news. Source: [8]*

Producing information will be done using a simple and efficient Android mobile application. The users will be using a mobile application both for producing and enriching informations with new media content. Information consumption will be done by using a wide range of media: the Web, Web TV (I am going to use Google Web TV ²) and the common Digital Video Broadcasting for digital television³. In order to achieve this goal, I will present a mobile application that users will use for producing information. Then a Web page and a Google TV website will be presented, which users will use for consuming information. The interaction between all of the above components requires the design of a complex software framework that will be introduced in Section 3.2. A general design overview of the software framework is shown in Figure 1.2. All the figures shown in this document are available in high resolution at https://www.dropbox.com/sh/yz4sebwft46ygro/3FZF-_0_z7.

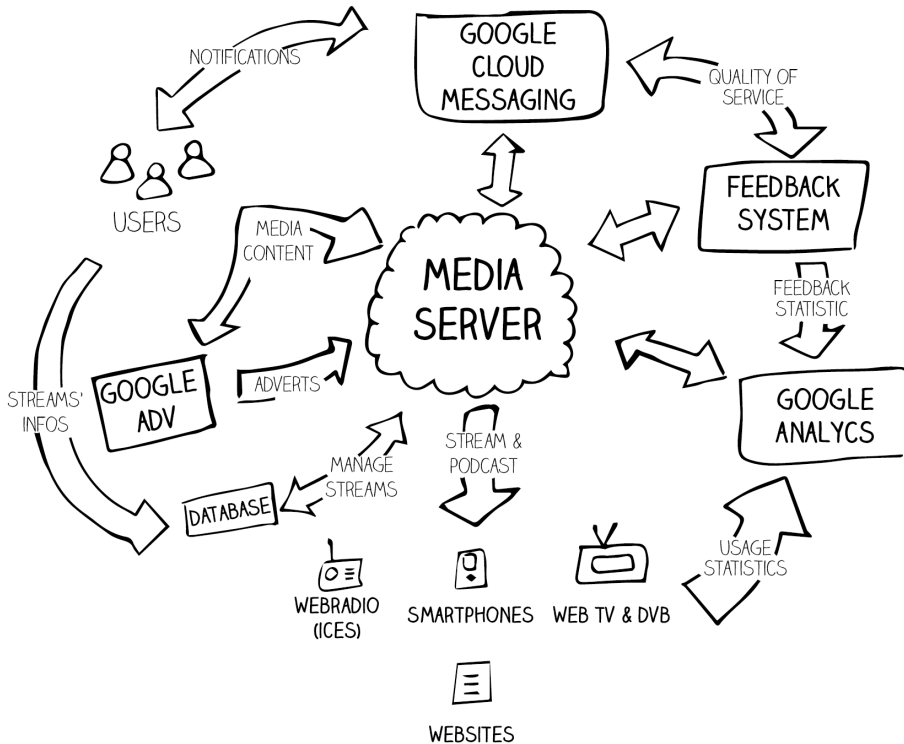


Figure 1.2: General design overview of the ioCast system.

²www.google.com/tv/

³http://en.wikipedia.org/wiki/Digital_Video_Broadcasting

Related Works

2.1 Mobile Video Streaming

Streaming video is video that is constantly received by the end-user while it is being delivered by a streaming provider. Streaming *content* is media (video, audio, text) that is consumed while it is being delivered, and streaming reflects the delivery system approach, which can be on-demand or live. Mobile video streaming is a video taken from a mobile device that appears in real time on the Internet through applications. Generally, users can push mobile video streams to various social networking sites and also embed videos in any website and share video links. The video can then be shared via email, SMS or social networking sites like Facebook and Twitter. Most live streaming applications work over cellular networks or WiFi networks. The basic set-up steps for a mobile streaming application are:

1. Create an account on a mobile video streaming site;
2. Download the software or the app to the smartphone;
3. Set up an account;
4. Start recording the mobile phone video;
5. Share the video (public or private);

There are already many existing solutions for streaming media (see Table 2.1). The table shows that there isn't a solution that connects the live broadcasting experience with the Web radio nor the Web TV media. Nor is there any solution that connect the world of the video streaming to the world of the common Digital Televisions Broadcasting DVB. **ioCast** is meant to broadcast video from smartphones to Web radios, the Web, Web TVs and DVB within an unique product.

Name	Description
www.qik.com	Live video capture and sharing with video gallery
www.ustream.tv	Live interactive broadcast platform
www.justin.tv	Broadcast what users are doing
www.vimeo.com	Respectful community of creative people who are passionate about sharing the videos they make
www.bambuser.com	Broadcast and chat live, stream live to blogs
www.twitcam.livestream.com	Creates a new page and a video player for the livestream broadcast. It then publish the link to a Twitter account so that followers can join the broadcast and the user can chat with his viewers via Twitter right from the broadcast page.
www.livecliq.net	Stream or share videos live to another mobile, the Web, family, friends, a channel or just public to the LiveCLIQ.net site.

Table 2.1: *Video Streaming solutions.*

2.2 Web Radios

Streaming radio over the Internet is similar to the video streaming: content is constantly received by the end-user while it is being delivered by a web radio hosting service. Users that want to set up a web radio has to connect to

hosting services such as **CheapestStream.com**¹ or **spacialnet**² by means of a broadcaster software such as **SAM Broadcaster**³. Those hosting services provide broadcast to end users very often using the three most popular web radio servers: SHOUTcast⁴, Windows Media⁵ and IceCast⁶ servers. While many broadcaster softwares (used as *input sources* for radio servers) exist for desktop computers, smartphones are left out of such softwares's market. Web radios transmit information over the Internet using media formats (*codecs*) that are available on any Android⁷ and iOS device⁸, therefore many web radio player both for Android and iOS powered devices exists, and they actually do connect to web radio servers and play (but merely play) web radio contents. The same content can also be played back on any website by *linking* the website to the web radio server's playlist, which is responsible for delivering the media content⁹.

2.3 Web TV

Designing a system where users can consume informations on their Web TV¹⁰ is for this project a considered choice. Statistics¹¹¹² shows that homes have an average of 2.5 people and 2.86 TV sets. And those people *watch* an average of about five hours of television per day. Statistics shows that there are more televisions then people, and thus probably more televisions than smartphones. Moreover, smart televisions are more and more common. In 2010 over 20% of all televisions were sold with integrated Internet functionality, and by 2014 smart televisions sales are expected to reach 123 million¹³. Statistics shows there is a lot of opportunities for Web TV developers. Users approach television in a very different way than they do with smartphone, laptops or tablets. Basically, they just want to relax and be entertained. Entertaining user with information lead me to choose the Web TV as a powerful *in-house* media for information distribution.

¹www.cheapeststream.com

²www.spacialnet.com

³www.spacial.com/sam-broadcaster

⁴www.shoutcast.com

⁵[http://technet.microsoft.com/library/ee822833\(WS.10\).aspx](http://technet.microsoft.com/library/ee822833(WS.10).aspx)

⁶www.icecast.org

⁷www.developer.android.com/guide/appendix/media-formats.html

⁸www.developer.apple.com/library/ios/#documentation/miscellaneous/conceptual/iphoneostechoverview/MediaLayer/MediaLayer.html

⁹For example M3U playlists, <http://en.wikipedia.org/wiki/M3U>

¹⁰As I already mentioned, this project will focus on Google Web TV.

¹¹US statistics, The Nielsen Company: Television Audience Report 2009, Three Screen Report www.nielsen.com

¹²Presented by Christian Kurzke, *San Francisco Android User Group*, January 2011. Source: <http://youtu.be/fN4qPgGoRSc>

¹³Quarterly TV Design and Features report. NPD DisplaySearch www.displaysearch.com

CHAPTER 3

Analysis

Having in mind the motivations and the goal for this project, I am now going to describe what **ioCast** will be like. We will see in detail how users will take an active part both in producing and consuming information using **ioCast**. Moreover, we will see how media companies can also use **ioCast**. I will start by defining a hierarchy of users that **ioCast** will have, to have clear in mind who are the actors and what they can do when using **ioCast**. I will then describe the supporting technologies I have used to design the **ioCast**'s software framework. Starting from the analysis of the users' hierarchy I will define the UML use cases for **ioCast**. UML use cases will clarify three important aspect of the project: the domains of interest stakeholders are related to, the prior requirements **ioCast** must have (as suggested in [7]), and the initial prototype's design of **ioCast**. I will describe in detail what users can do when using **ioCast** by means of *event models* and *scenarios*. Event models and scenario models are powerful tools described in [7], and they are used to model what a system is supposed to do by an user prospective. First of all, I will introduce the user hierarchy.

3.1 User Hierarchy

In this section I will basically *translate* the Bottom-Up news model¹ as the basis for the design of *ioCast*.

The Director

The director is at the top of the users hierarchy. He does direct all the audio and video of a show with his smartphone. He has control of the participant that are speaking and he can mute and un-mute them whenever is necessary. Moreover, he privately communicates with all the other participant of the show for any reason without disturbing the ongoing show. Whenever a guest participant outside the studio wants to interact with the participant and the show, he can put him on air and join him to the show. He controls the airing of a video that he or another participant is taking and he privately chat with any participant if a voice communication is for any reason not allowed.

The Participant

The participant will participate to a show as a guest or as a member of the crew, after being approved by a director, and using his smartphone. Participant can also privately communicate with the director or other participants by using chat or by sending vocal messages. A participant will also take a video of the show, which can be aired by the director.

The Internet user

The Internet user does listen to the voices and watch at the video that *participants* are taking *live* during a show. He can consume information using the *ioCast*'s Website, Web TV or digital television. He can review the show that has been aired days before (podcasts) and share or comment it. The Internet user might also have right to participate on the creation of content that will enrich the audio or video content of other's, as for example by adding subtitles or voice tracks with a different language to a media content.

¹See Figure 1.1 on page 2

The Producer

The producer is responsible for a television channel that broadcasts its shows using the common Digital Video Broadcasting systems (DVB). The producer can link his DVB system studios with a live video (or podcast) that can be produced by any users using *ioCast*. Such video can then be seen on common digital television channels.

3.2 Supporting Technologies

An overview of the supporting technologies for this project is again shown in Figure 3.1. I will now discuss the technologies for *ioCast* in more details.

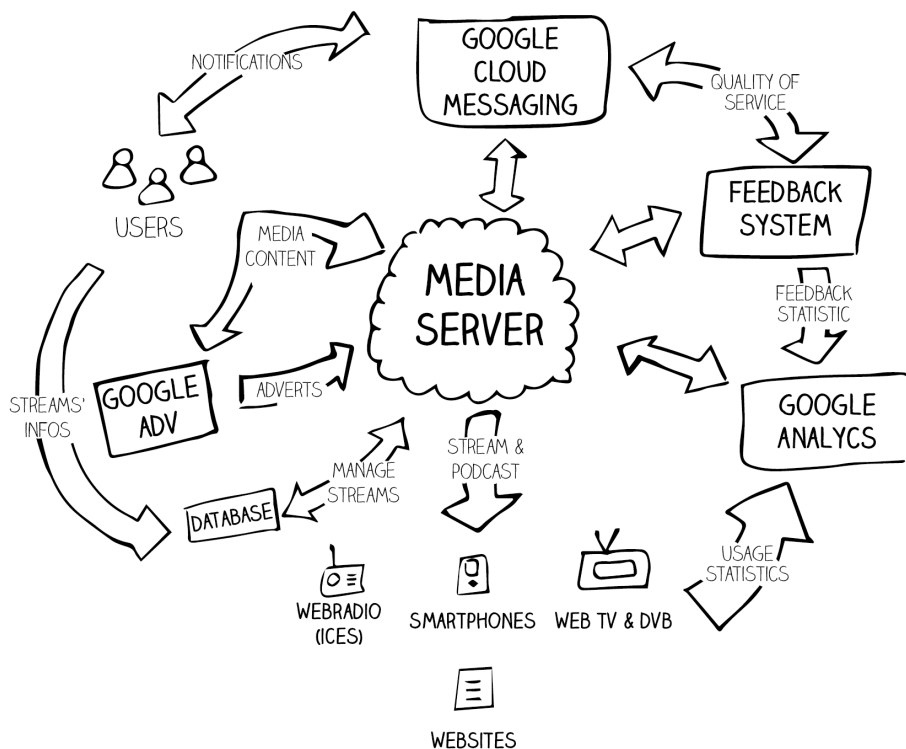


Figure 3.1: General design overview of the *ioCast* system.

The Media Server

The media server is responsible for catching the signals coming from all the users and mix them together with other signals coming from other sources. Signals include: audio, video, subtitles and advertisements. The server will provide the streaming of all the above mixed signals to the end users via the Web, Web TV and digital television. Generally, the server is controlled from the outside world by sending messages over network communication protocols.

The Database

A database will manage user's identity and control their interactions with the media server. Basically, the database will be the gateway for the creation and the cancellation of streams on the media server.

The ADV system

The ADV system is responsible of *injecting* advertisement over a video or over an audio during the streaming of a show. The ADV system works by recognizing key words within the live streamed audio track (the voices of the participants) that can potentially be used to show (or to play) commercials over a show's streaming.

The Cloud Messaging system

The Cloud Messaging System is responsible of sending small size messages to the users's smartphones connected to the media server. Messages will notify users with instructions of any kind.

The Feedback system

The feedback system will work using the support of the Google Cloud Messaging system, and it will provide a quality of service for the smartphone connected to the media server. Users connected to the media server will be notified about the signal's quality of the information they are sending to the media server.

The Analytic system

The Analytic system will track all the statistic about the usage of the **ioCast** system, including statistics on the quality of the signals sent to the media server from the users's smartphone (also location data to see what area are best served by mobile networks) and statistics on the media contents's consumptions (for example, number of views) by the Internet users.

The Podcasting system

The Podcasting system will record all the information sent to the media server and will keep it stored for future access.

3.3 Event Models

The term *business event* signifies the system partitioning as a convenient way to look at each of the system's functions, so we can deal with them in a way that is familiar to the users. Moreover, partitioning will simplify the design work of the system[7]. I will now list the events, and in Section 3.4 I will model each event using *scenario models*.

Video Streaming This is the main event of **ioCast**: users will be able to stream their generated content (this includes audio and video) from their smartphones *to* any other devices supported by **ioCast**;

Radio Streaming Users will be able to go on air on the Internet using their smartphones and without using any other software nor hardware equipment.

Participating Participation can be done in two ways: *social* participation and *media* participation. The first involves the use of social networks for sharing and commenting all the content that **ioCast** will distribute to the end users. The *media* participation involves media companies that want to integrate their content with the one generated with smartphones using **ioCast**.

Podcasting This event is generated from the above two *streaming* events (video and radio) and from the *social participation* event: **ioCast** will be able to store all the streams generated by the users, which can be later seen at any time.

Watching This event is the result of the two above *streaming* events (video and radio): users can watch the content of **ioCast** on the Web and on any other media platform supported by **ioCast**;

Integrating Users have the possibility to integrate whatever they want on other users's content: this event is essential for using **ioCast** by the wider audience target.

Advertising **ioCast** will be able to recognize the content that users generate. During the streaming of such content commercials may be integrated within the content itself. Advertising may potentially give **ioCast** a revenue stream.

3.4 Scenario Models

In this section we will build a *user's point of view* story for each event listed in Section 3.3. Scenarios are here listed in non-chronological order, as there is no such order on the usage of **ioCast**.

Video Streaming Mark is having a nice trip on a bus in central America, and he decides to share the beautiful landscape he sees while moving on the bus, so he runs **ioCast** on his smartphone and soon after with just pressing a button he starts streaming his video live to the **ioCast** system; the network condition outside are changing and are not the best, so Mark receives some warning messages: one says *Move to a better covered area* and another says *You're back to a good covered area*. As a consequence of the first message, the video Mark is taking will be sent in a low quality resolution to the **ioCast** system; Instead, by receiving the second message, the video is being sent in the highest quality resolution. Mark keep moving on the bus when another warning message arrive, *Mute the microphone!* it says: it seems that somewhere another user is watching Mark's video and noticed that there's no need to record the disturbing background noise of the other people on the bus, so as soon as Mark reads the message he just press a button and mute the mobile phone's microphone.

Media Participation This scenario belongs to any media company that uses **ioCast** as a part of their broadcasting. Jack dreams of being a professional reporter, and he has his own blog on which he posts many video taken all around Europe documenting tips on where having good not expensive food in the European capitals. He is now in Copenhagen and he has been contacted by a television program in Spain which talks about travels. They want Jack to have a short live video about Copenhagen's places to eat. Jack is very happy about that and after being around Copenhagen for a week, he's ready to give his report to the Spanish television. They

agreed on going on air on a Tuesday afternoon, and the event has also been posted on the **ioCast** website. When Tuesday comes, Jack runs **ioCast** on his smartphones and start the live reporting for the Spanish television. Meanwhile, the television company is able to control the audio and video that Jack is sending, including sending him warning messages, and they can easily live stream Jack's video directly to their channel by simply connecting to the **ioCast** system, without using any other software or hardware. As we have mentioned, Jack's report is also being streamed in the **ioCast** Website, and Ida (Jack's friend, and student from Copenhagen whom also speak Spanish) is live translating Jack's report from Spanish to Danish so that users on the Web or on the digital television in Spain can select their preferred language when watching Jack's report, besides sharing and commenting it using a social network.

Social Participation Internet users of **ioCast** can go to the **ioCast** Website and see the ongoing broadcasts. If the user has the rights, he can select a broadcast and add subtitles (only for podcasts) or add a live translation audio track for such broadcast via the very same Website or the **ioCast** smartphone application. **ioCast** will then notify other users and producers about the new available translation and subtitle. Moreover, Internet users can share and comment all the ongoing broadcasts and podcasts using the main Internet social networks.

Radio Streaming This scenario includes the *Video Streaming* scenario. Christian is the show director for a small web radio called *RadioCast*. This evening he's going on air with his radio crew and they have a guest at the show, Maria, who is not at the radio studios with the crew and needs to participate remotely. Christian does not want to set up difficult software or hardware in order to interact with Maria during the show, so he runs **ioCast** on his smartphone, and so do Maria and all the other member of the radio crew. Christian can see that Maria and all the crew is connected to **ioCast** and he is the actual director of the show. When it comes to Maria's turn, Christian is able to air what Maria is saying by simply pressing a button on his smartphone application. Now Maria is on air and everything goes fine until Christian must ask for something to Mark, but no interruption of the show should be done, neither can he distract the rest of the crew who is interviewing Maria. So he just press another button on his smartphone application to start a private conversation with Mark who then reply to Christian. Now Maria would like to show where she is (which is important for the subject of the interview) so she just press a button on her smartphone application and the camera activates. All she's now shooting will be live streamed to the **ioCast** system upon the permission of Christian, who allows it by just pressing a button on his smartphone application. During this period Christian - who is the director of the show - has the right and the possibility to suspend the audio and/or the video of Maria at any time, for any reason, by just pressing a button.

When Maria finishes to speak and to stream her video, Mark would like to continue sharing the live video of the show from the studio where the crew is, so he just does the same as Maria did for sharing hers, upon the permission of Christian which is requested via a private conversation between the two. At the end of the show Christian can log out all the crew including himself from **ioCast** so that no more voice or video is on air.

Advertising A video is being streamed on **ioCast** by Christian during his web radio interview to Jack, a movie expert, for a small web radio called *RadioCast*. Jack is speaking about the best movies that can be seen at the cinema in Copenhagen. While Jack speaks, he says many words related to movie titles and cinemas names located in Copenhagen. **ioCast** recognizes those key words and the streaming is being injected with advertising which redirect to online booking website for cinemas and online shops for movies. Those advertising are appearing on the **ioCast** Website and also in another website, where the video is being shared by another web radio called *CinemaRadio* which is linked to the **ioCast** website of *RadioCast* and found Jack's interview very interesting.

Video Sharing This is one of the most important scenarios of **ioCast**. Thomas is an **ioCast** user and he is taking a video with his mobile phone, which is being streamed using the **ioCast** smartphone application. Ida is sitting on a train and she's watching Thomas's video using the **ioCast** smartphone application. Mark is home sitting on his sofa and he is watching the same video on his laptop from the **ioCast** Website. Thomas's video is also being broadcasted on a local digital television channel in Denmark, and Ida is watching it from her television in her bedroom. Thomas's video is also being streamed to his channel on the **ioCast** Web TV.

Podcasting Christian is an **ioCast** user and he missed his friend Mark participation to a web radio called *RadioRock*. Mark has been the guest of the Tuesday night's show of *RadioRock*. It is Friday evening and Christian would love to watch the last Tuesday night's show of *RadioRock*. Hopefully, Christian connects to the **ioCast** Website and there he can find all the past episodes of *RadioRock*, including the one with his friend Mark. He just press Play on the video and he can now see his friend Mark on his computer.

3.5 Use Cases

In this section we will produce the UML *use cases*[3] by having in mind the related works of Chapter 2 and the analysis of this chapter's previous sections. Use cases will describe *what* the system is expected to do, and they will be subject of identification of detailed requirements. By defining the requirements

for prior use cases we will see *how ioCast* will perform its job. The main use cases for this project are shown in Figure 3.2. We will analyze prior use cases for which we will define *functional* and *non functional* requirements using the process described in [7]. Functional requirements are the thing the product must do, whereas non functional requirements are the properties the product must have. For both type of requirements we will give a description to capture the user's intentions when using *ioCast*, and a *fit criteria* which will set the standard to which the design of *ioCast* will be based on. Fit criteria will also provide the goals that we will use when testing *ioCast*, to determine if each requirement has been met. We will only define requirements for prior use cases of Figure 3.2, which are the ones that are critical for the prototyping of *ioCast*. Requirements and their related fit criteria are presented in the next section.

3.6 Requirements

Requirements and their related fit criteria are shown in the following tables.

Use case	Functional requirements	Non functional requirements
Direct the participants of a show	<ul style="list-style-type: none"> • The director shall control all the other user's streams (audio or video); • The director shall privately communicate by text or voice with all the other users; • The director shall join any user to the streaming; 	<ul style="list-style-type: none"> • The direction of users' streams shall be easy for the directors; • Joining any user to the streaming shall be an easy process for the directors;
Send audio or video stream	<ul style="list-style-type: none"> • Any user shall send audio or video stream to the media server; • Users shall receive feedback on the quality of the stream they're sending to the media server; • Users shall privately communicate by text or voice with all the other users; 	<ul style="list-style-type: none"> • Sending audio or video from the smartphone must be a very easy process; • Data transmission to the server must take care of delays due to the network link between the smartphone and the media server; • Other users shall receive the content within an acceptable delay; • The smartphone shall change the recorded audio and video quality according to the network's conditions;

Table 3.2: *Functional and non functional requirements. Part 1 of 3.*

Use case	Functional requirements	Non functional requirements
View a live/podcast stream online	<ul style="list-style-type: none"> • The user shall log on the system and visualize all the current live/podcast streams; • The user shall select a stream and watch it; • The user shall select a stream's audio track; 	<ul style="list-style-type: none"> • The window of a selected stream must only show essential information of such stream; • The user shall be able to switch between streams and their audio tracks very easy and without noticing interruptions such as page reloading;
Participate to a stream	<ul style="list-style-type: none"> • Users shall select a stream and integrate it with a new language (new audio track); • Users shall select a stream and integrate it with a new subtitle track; • The system shall check if the user has the rights to integrate content to a selected stream; • Integration of subtitles shall be done only for podcasted streams and not for live streams; 	<ul style="list-style-type: none"> • Integrating a new audio or subtitles track shall be an easy experience for the user;

Table 3.4: *Functional and non functional requirements. Part 2 of 3.*

Use case	Functional requirements	Non functional requirements
Connect to a stream of the media server and broadcast the stream on DVB television	<ul style="list-style-type: none">• Television producers can select an user's stream and broadcast it on their DVB television;• ioCast shall provide producers the selection of a stream;• ioCast shall check if the producers have the right to broadcast a selected stream;• ioCast shall provide a way to broadcast the selected stream in DVB television;	<ul style="list-style-type: none">• The link to a selected stream must be able to interface with DVB Television;

Table 3.6: *Functional and non functional requirements. Part 3 of 3.*

Use case	Functional criteria	Non functional criteria
Direct the participants of a show	<ul style="list-style-type: none"> • The smartphone shall receive commands for start, pause or stop the streaming of any content to the media server; • The media server shall send a notification to the smartphone that a given stream has been dropped; • The smartphone shall receive audio or text input and send it to other users; 	<ul style="list-style-type: none"> • A simple user interface must be implemented with only few buttons for managing the content's streaming; • A simple user interface must show the director which user can join the streaming;
Send audio or video stream	<ul style="list-style-type: none"> • An user interface will accept audio or video input from a smartphone; • Notification messages regarding the quality of the stream shall appear on the users smartphone's screen; • The smartphone shall record audio or text input and send it to other users; 	<ul style="list-style-type: none"> • Two different windows for the audio and video streaming will be used to facilitate the user with the streaming process; • The streaming of the content must be done only after the connection to the media server has been established; • Audio quality and video resolution will change accordingly to the network's conditions;

Table 3.8: *Functional and non functional criteria. Part 1 of 3.*

Use case	Functional criteria	Non functional criteria
View ^a live/pod- cast stream online	<ul style="list-style-type: none"> • A log in window will lead the user to a windows where he can browse all the live/pod-cast streams; • A window for a selected stream shows the user the video and plays the audio; • A window within the one of the selected stream shall show all the available audio tracks (languages) for such stream; 	<ul style="list-style-type: none"> • A window for the select stream will only show basic information regarding the stream, such as author and time informations; • Windows navigation within the streams and audio tracks must include the least number of windows switches; • Clear and expressive components of the user interface must easily guide the user through the browsing of both live and podcasted streams, including the selection of their audio and subtitles;
Participate to a stream	<ul style="list-style-type: none"> • When selecting a live stream, the user shall both listen to the stream and record his translation on the same window; • When selecting a pod-cast stream, the user shall listen, pause the stream, and write the subtitles on the same window; 	<ul style="list-style-type: none"> • Windows navigation within the streams, podcasts and the content integration's windows must include the least number of windows switches; • Clear and expressive components of the user interface must easily guide the user through the integration process, both when recording audio tracks and integrating new subtitles;

Table 3.10: *Functional and non functional criteria. Part 2 of 3.*

Use case	Functional criteria	Non functional criteria
Connect to a stream of the media server and broadcast the stream on DVB television	<ul style="list-style-type: none">• A log in interface will lead producers to audio and video content page;• A browser will guide producers to the selection of a stream;• An interface will provide a link to the selected stream which producers can use to link such stream with their DVB Television systems.	<ul style="list-style-type: none">• The selected stream which the link refers to will be a multimedia stream that can be encoded using several encoders. This will give the opportunity to find the best encoder compatible with DVB television systems.

Table 3.12: *Functional and non functional criteria. Part 3 of 3.*

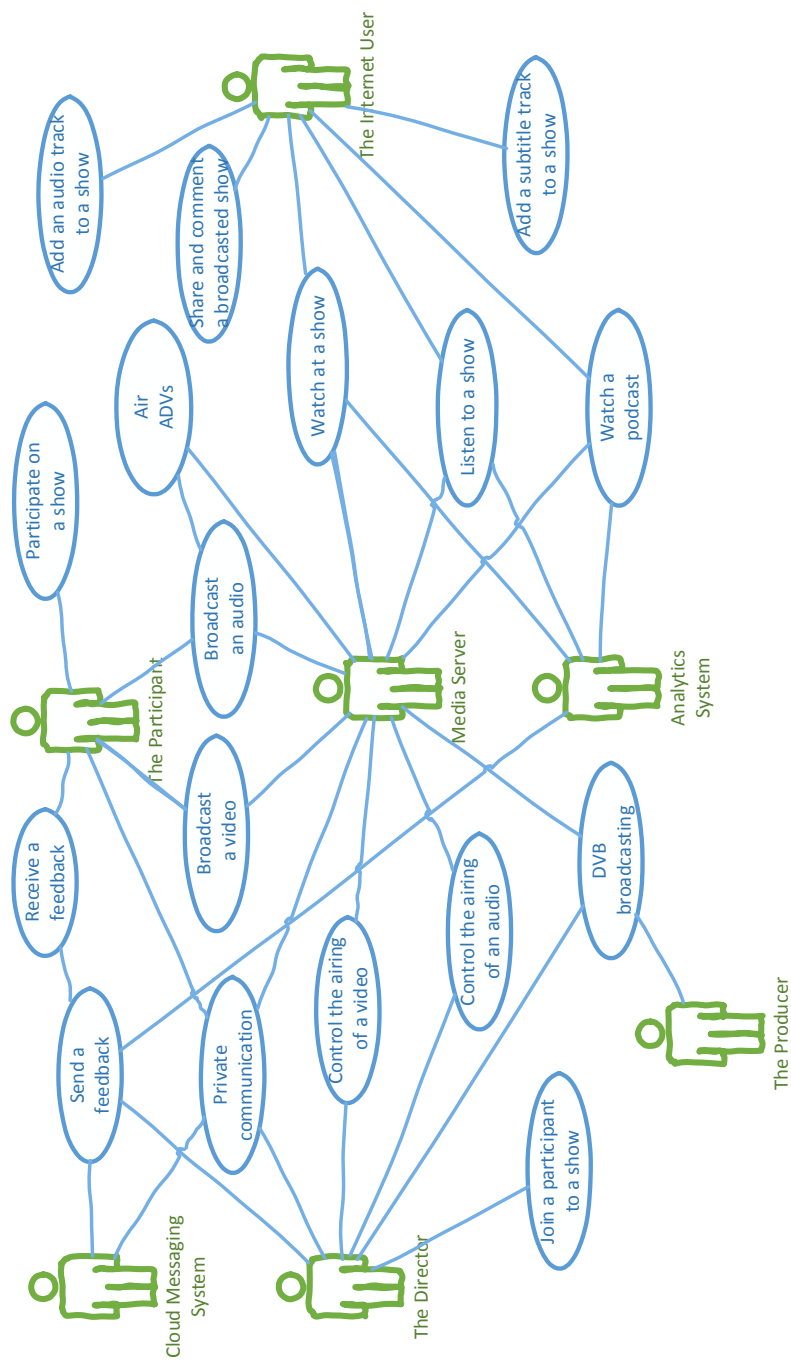


Figure 3.2: Use cases for *ioCast*

CHAPTER 4

Design

This chapter is based on the work of [5] where the indication of *skatching* a low fidelity prototype is the best way to get an idea of what a feature of any system has to do. The low fidelity prototype can be than used to create the real final implementation of the user interface for **ioCast**. After the analysis chapter, I will now focus on the user experience and I will present mockups for the user interface of the smartphone application. Mockups for Website and Web TV are also being presented in this chapter. Web Tv and Website will share the same design because the Website will be a high polish user interface which will be optimized for Web TV[4]. I am now going to list the design for the user interfaces of all the user hierarchy presented in Section 3.1.

4.1 Mobile application

Directors will be able to control the participant's of a show as shown in Figure 4.1 where directors can control all the participants and the airing of their audio and video.

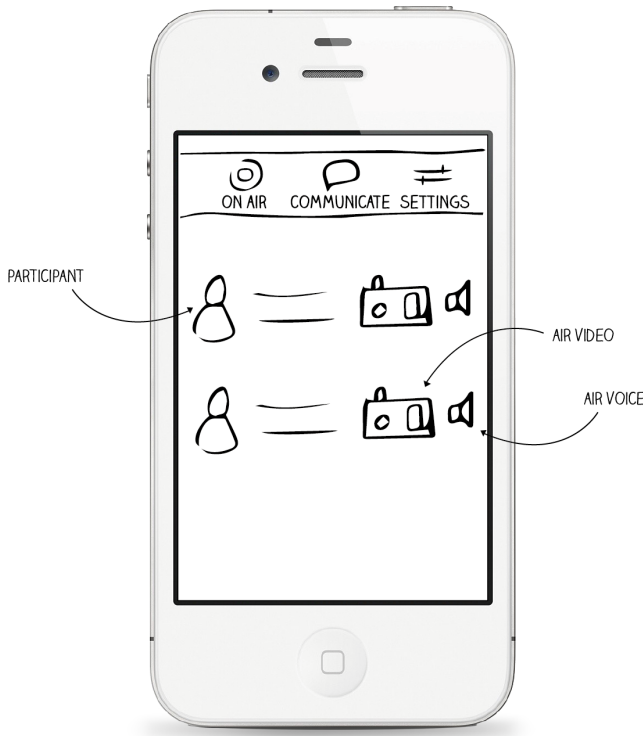


Figure 4.1: *Directors's User Interface.*

Participants will be able to go on air as shown in Figure 4.2. The figure shows how participants can start and stop streaming a video. The figure shows also how a participant can receive a feedback notification.

Participants will privately communicate as shown in Figure 4.3. The figure shows how participant can start a private chat session, and send a private audio message to other participants. The figure shows also how participants will receive private audio messages from other participants.

Internet users will be able to browse on air shows and podcasts as shown in Figure 4.4. The figure shows also how Internet users will also be able to integrate subtitles to podcasted streams, and a language audio track to a live show.

Internet user will be able to integrate subtitles and language translations using the user interface shown in Figure 4.5. With this UI Internet users will be able to control the playback of the video while they're providing text and audio.

4.2 Web TV Website

As I have already mentioned, Web TV and website will share the same design. Browsing of on air shows and podcasts is shown in Figure 4.6 and Figure 4.7, respectively. Here, Internet users can browse the show, search for shows and see information about the shows.

Detailed information about a selected live broadcast is shown in Figure 4.8. Detailed information about a selected broadcast is shown in Figure 4.9.

Playing of a broadcast or a video is shown in Figure 4.10, where the Internet user is able to share the video on social networks, select available subtitles and audio languages.

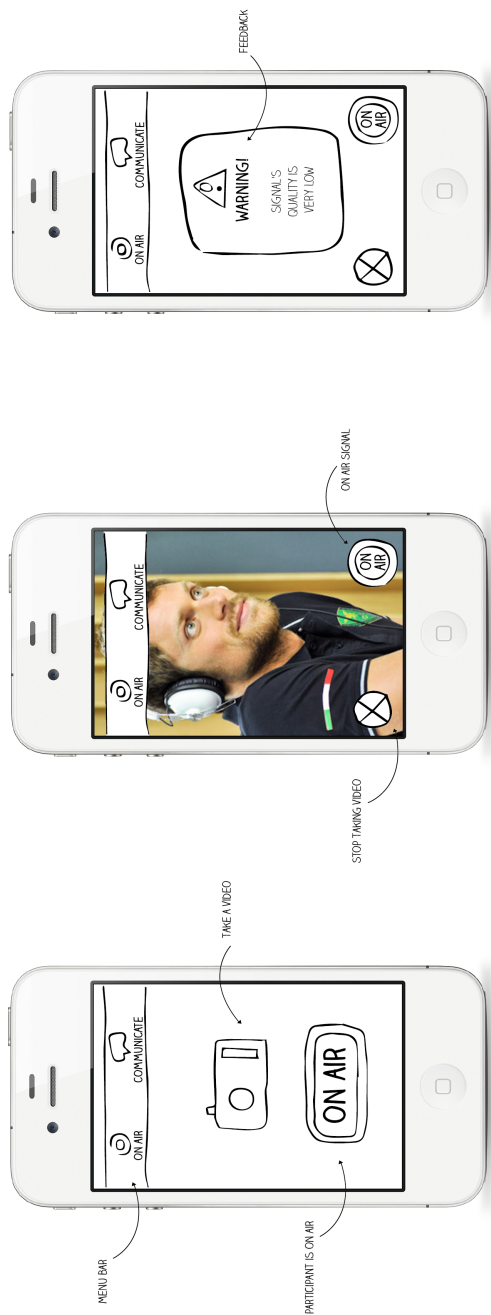


Figure 4.2: Participants's UI: Airing audio (left) airing video (center) and feedback notification (right).

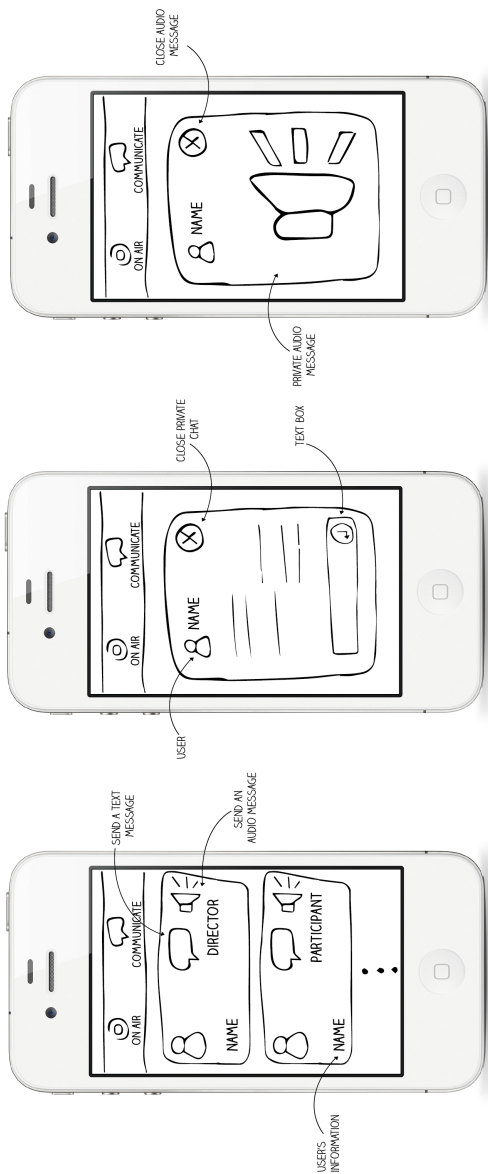


Figure 4.3: *Participants’s UI: Main private communication window (left); Private chat communication (center); Private audio message reception (right).*

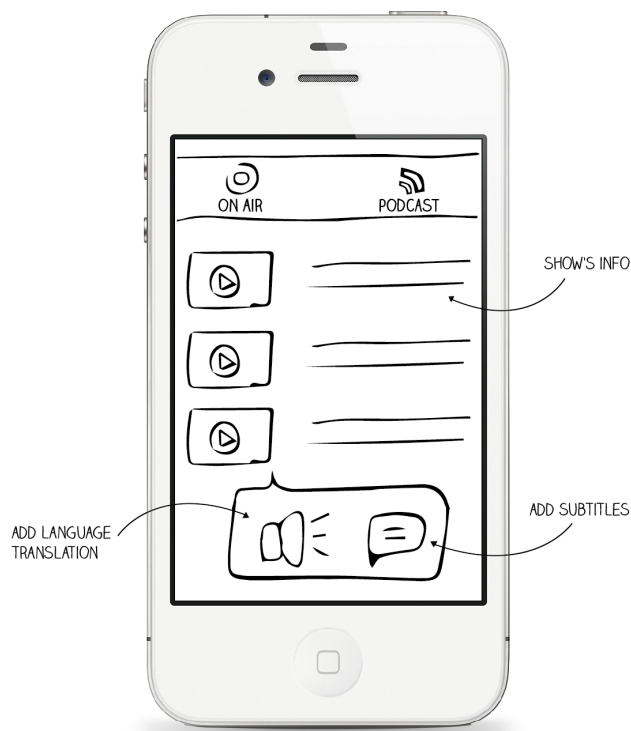


Figure 4.4: Internet users’s UI: Streams’s browser and player. Subtitles and language audio tracks integration buttons.

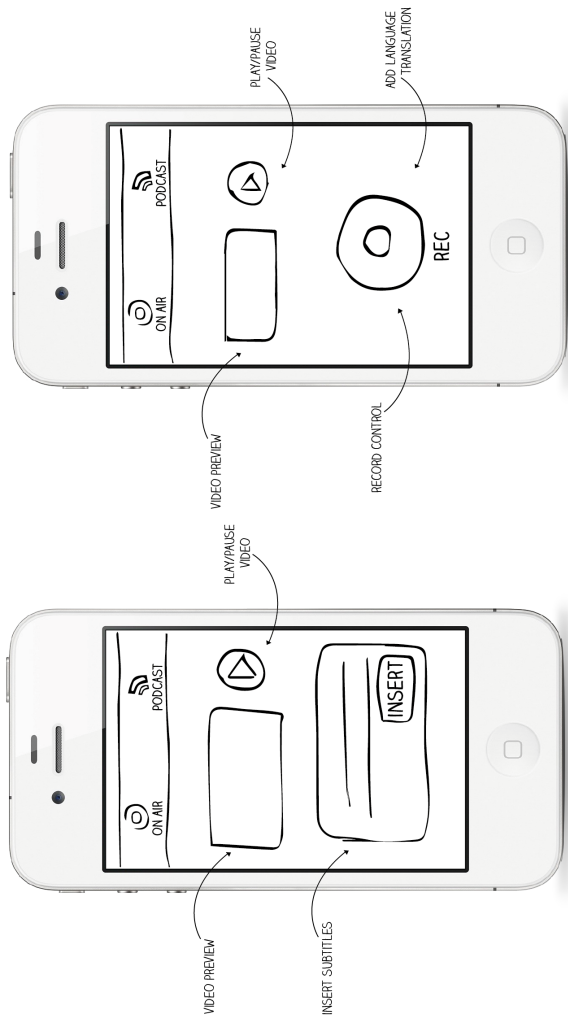


Figure 4.5: Internet users's UI: Subtitles (left) and language translations (right) integration.

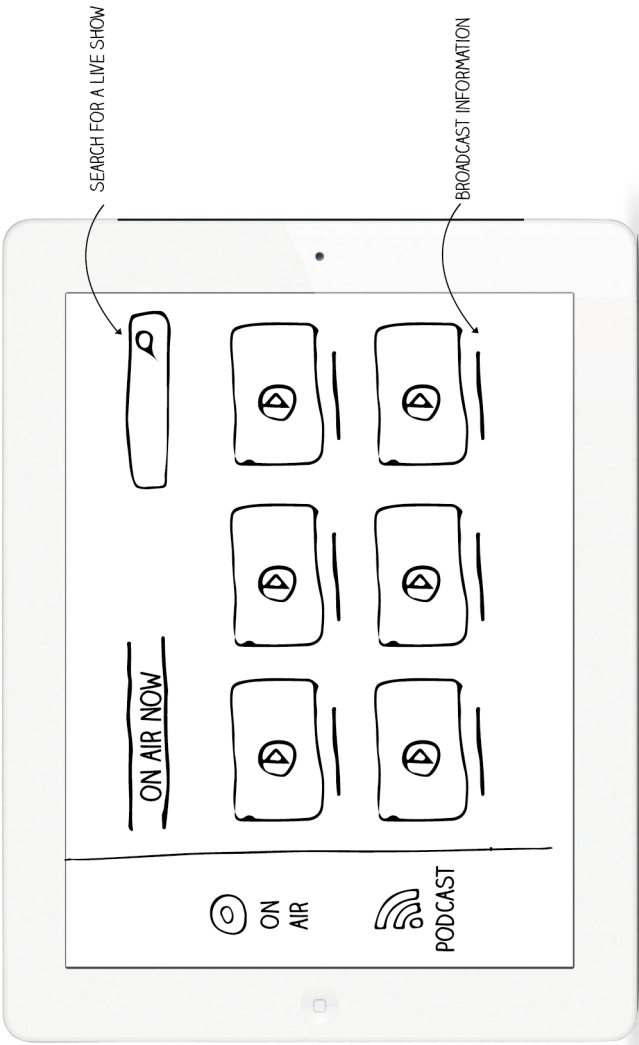


Figure 4.6: *Internet users’s Website and Web TV: Browse live broadcasts.*

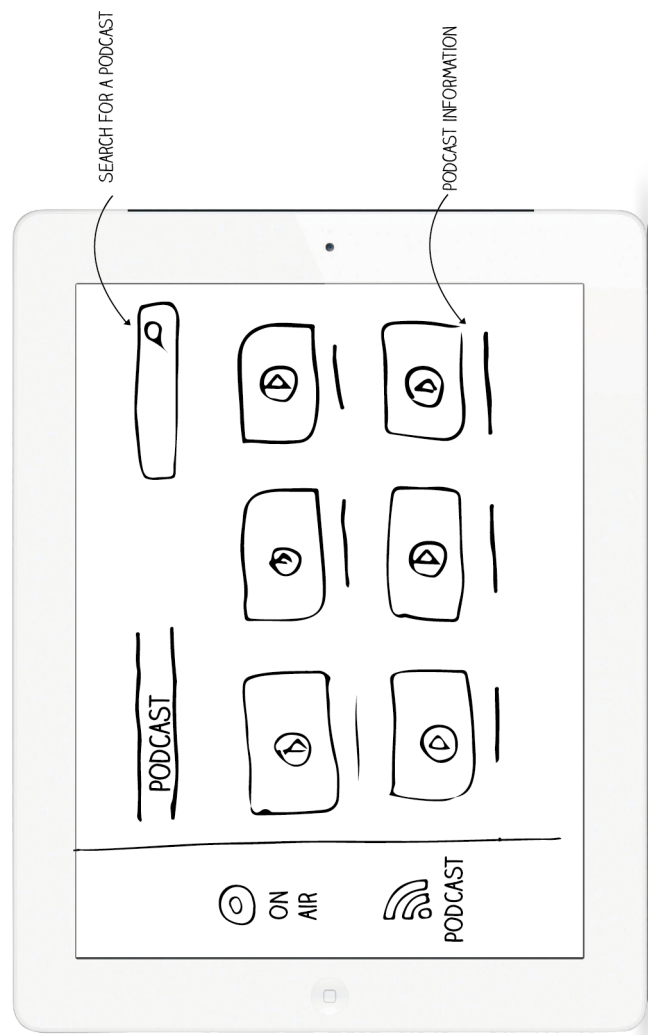


Figure 4.7: Internet users’s Website and Web TV: Browse podcasts.

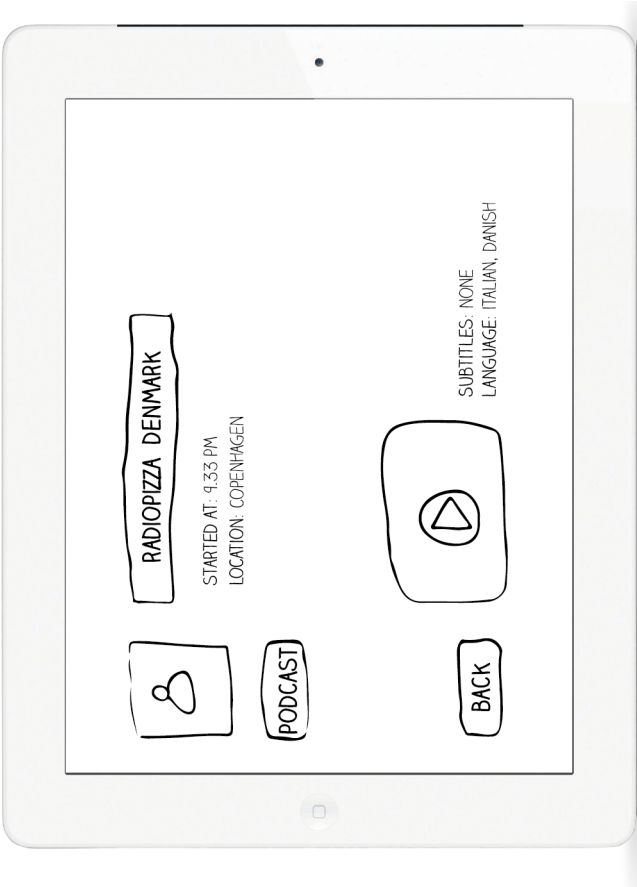


Figure 4.8: *Internet users’s Website and Web TV: A live stream.*

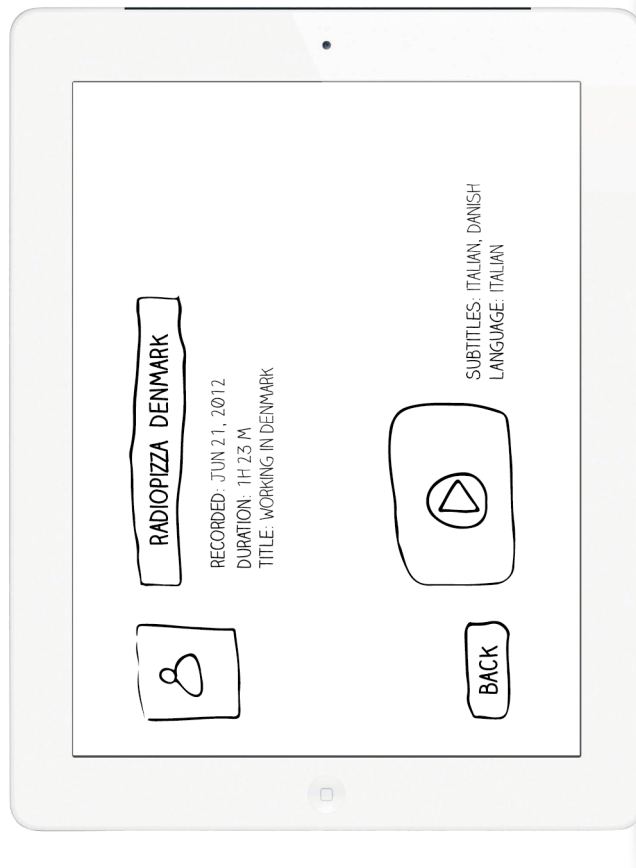


Figure 4.9: *Internet users's Website and Web TV: A podcast.*

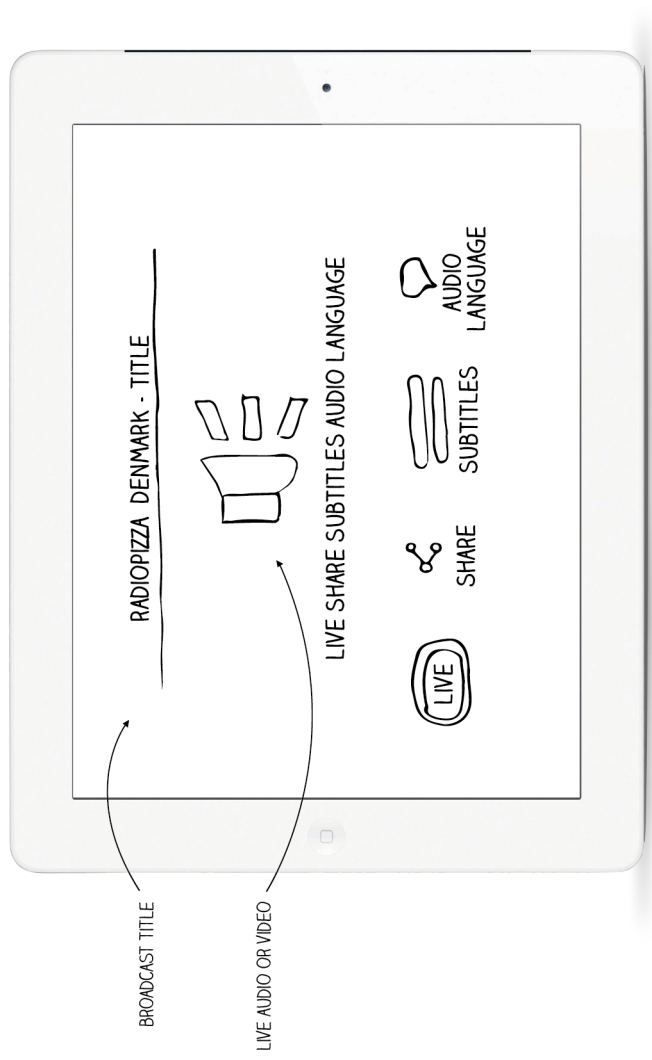


Figure 4.10: *Internet users’s Website and Web TV: The video player. The speaker icon is only shown when a video is not available for a stream.*

Implementation

In this chapter we will present the implementation of the two more important part of the `ioCast` system shown in Figure 1.2 on page 3, the media server and its database.

We are going to use `FFserver` as our main media server system. `FFserver` is part of the `FFmpeg` project, which is a free software licensed under the LGPL or GPL. `FFmpeg` is fast and quite stable, and many known media softwares incorporate work from `FFmpeg`¹. An overview of `FFserver` is shown in Figure 5.1. `FFserver` is a streaming server for both audio and video. It supports several live feeds, streaming from files and time shifting on live feeds[2], thus it is a good solution for prototyping our media server.

The *feeds* are the core elements of our media server and all the `ioCast` system because they are containers wherein we can mix audio, video and subtitles. Each feed is specified by a `<Feed>` section in the `FFserver`'s configuration file. For each feed we can have different output streams in various formats, each one specified by a `<Stream>` section in the configuration file. As we will see later, feeds will be connected to input *sources* (smartphones in our case) by using instances of the `ffmpeg` tool². Such feed containers will be linked to one (or more) stream(s), and they will represent the media content distributed to

¹See <http://ffmpeg.org/projects.html>

²<http://ffmpeg.org/ffmpeg.html>

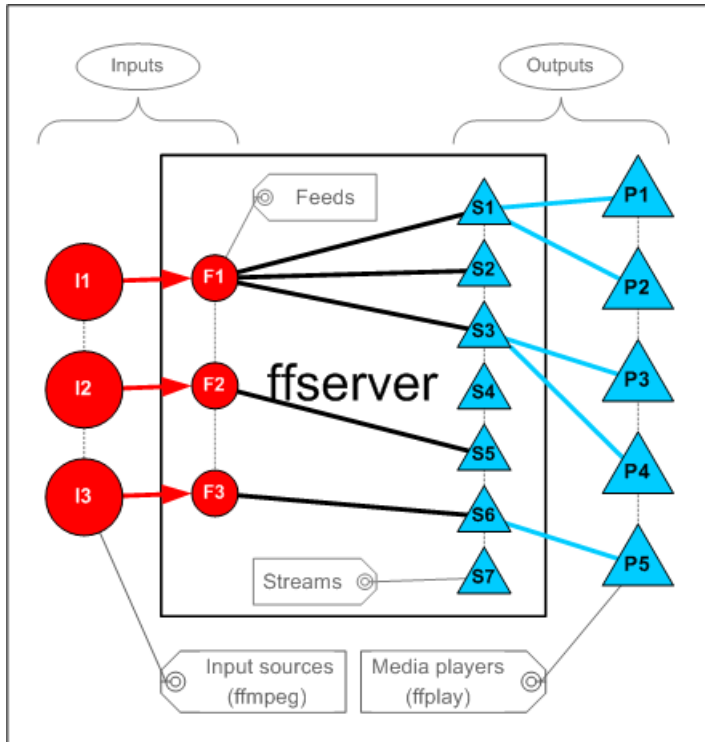


Figure 5.1: FFserver system. Source: <http://ffmpeg.org/>

multiple clients for viewing. Each stream's format has a filename and it is linked to one feed. We can associate a feed with more output streams when we want to stream one input source using several different output formats. We can choose among many stream formats³ thus giving us the opportunity to stream to my chosen platforms: desktops, smartphones, Web TV and DVB. FFserver will send a stream when answering a request containing a given file name. As we will see later, it is necessary to manage the creation and cancellation of the **ffmpegs** instances (users's streams) thus joining or disjoining users to and from a broadcast. Moreover we can only link each media server feed (representing one broadcast) with only one **ffmpeg** instance, thus managing **ffmpegs** instances of different users is a complex process that requires automation. Therefore we will implement a MySQL database. The database will manage user's identity, so that every feed will be bond to users that are participating to a given show. The database will be the gateway to the creation and the cancellation of **ffmpegs**

³MPEG-1 and 2, Ogg, RealNetworks, Windows Media Player, Adobe Flash, H264.

instances, i.e. users's streams. Examples showing how the database manages **ffmpeg**s instances will be given later in this section. We can now introduce the main operations the media server will perform.

Receive a stream from a user This operation is shown in Figure 5.2. First the user will send his ID to the database. The database will then create an **ffmpeg** instance for the user. Such **ffmpeg** instance will receive the media content from the user (in this case using HTTP) and it will be bond to a feed in the media server. The feed will be linked to a stream which will stream the user's content. As we will see later, the database will also bond new users joining the same stream;

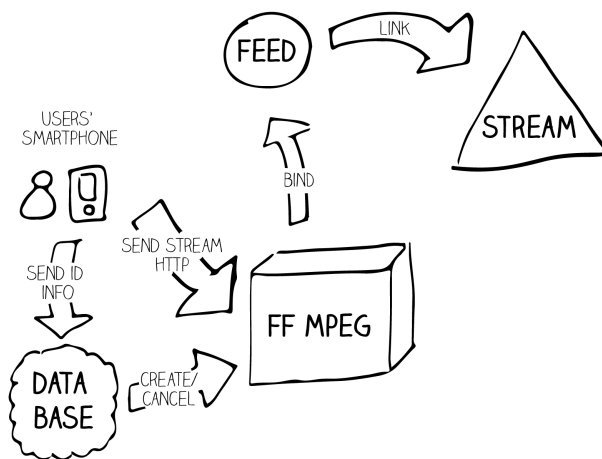


Figure 5.2: *Media server: User's streams creation.*

Control a media stream This operation is shown in Figure 5.3. The director will provide an audio and video content by simply *feeding* a feed with such content coming from a user's smartphone. This operation is simply done by launching (or killing, in case the director wants to put an user off air) a **ffmpeg** instance, thus bond such instance (or un-bond it) from a feed; The *launch* and *kill* operation are done by the director's smartphone via the database.

Control multiple users's streams This operation is shown in Figure 5.4. When a new user does join a broadcast, his audio or video input can be mixed with the other users's into a single output. Mixing is done by creating a new **FFmpeg** instance which will then bonded to the feed. The database will control the creation of the new **ffmpeg** instance and the new binding to the feed.

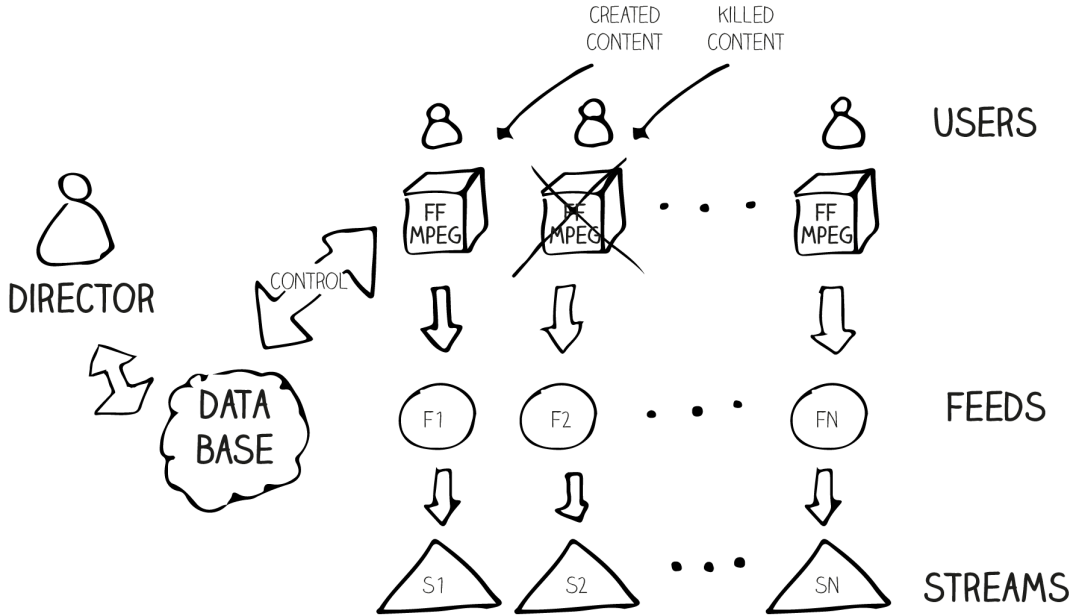


Figure 5.3: Media server: Users's streams control.

Private communications This operation is shown in Figure 5.5. Users will be exchanging private text messages via the Google Cloud Messaging system. Feedbacks will be also sent through the system. Audio messages will be sent to users as new audio streams of the media server. The user sending an audio message will first create a stream via the database. Then the receiver will be notified of the private audio message stream's name through the cloud messaging system. Finally, the receiver will be able to connect to the stream and listen to the audio message.

Feedbacks Much like in the *Private communications* operation, directors will first send feedbacks to the Google Cloud Messaging system as text, which are then sent to the concerned users's smartphones.

Manage subtitles and audio tracks When subtitles and language audio tracks are available, a `ffmpeg` instance will integrate them, and a feed will be bond to such instance. Then a stream on the media server will stream a content with such subtitle and language track.

Podcasting All the feeds provided to the media server can be stored in a file on the media server. The file can be then accessed as podcasts;

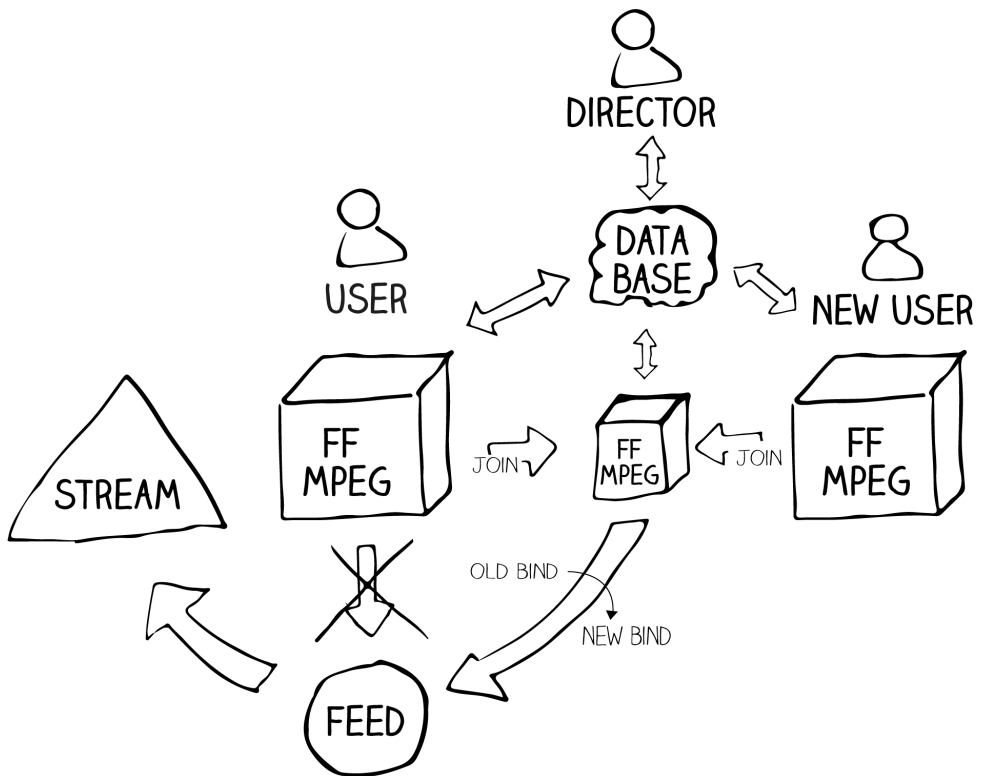


Figure 5.4: *Media server: Join of multiple users's streams.*

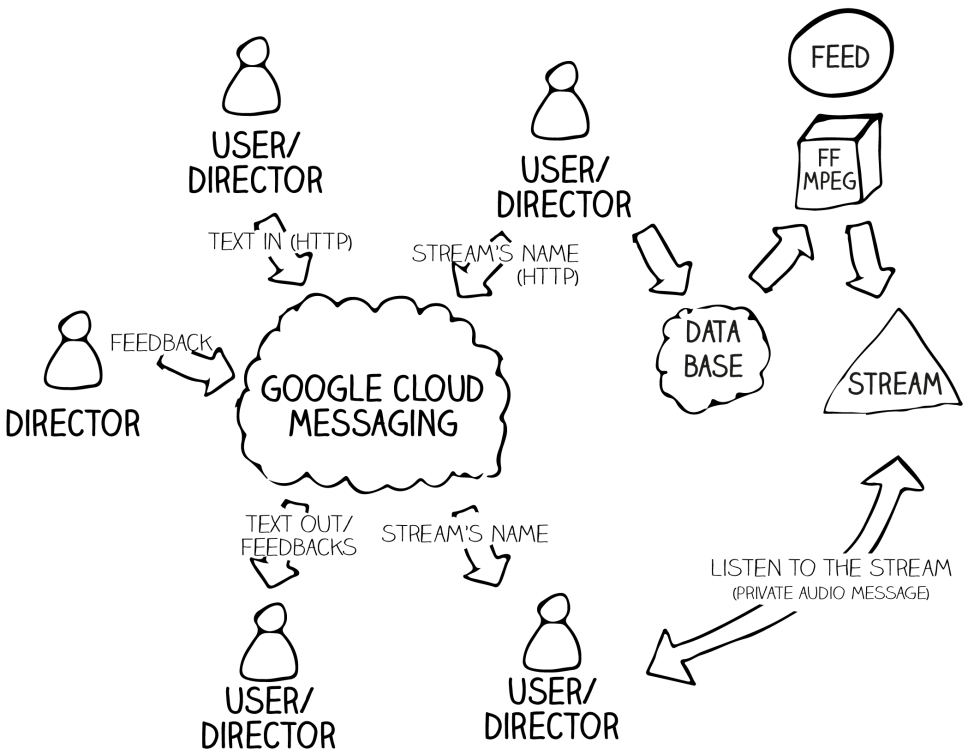


Figure 5.5: Private communications and feedbacks.

CHAPTER 6

Tests

A series of test are here presented for the Android application and the Google Web TV website. These are only suggested tests for **ioCast**. Results for these tests are not going to be presented, as a full featuring prototype for this project has not been implemented.

6.1 Mobile Application

As indicated by the Android Developers website¹, we are going to present three scenarios, and a list of things we should verify when each scenario occurs while users are using the mobile application. The following sentences in italics are taken from the Android Developers Website².

Change in orientation *When Android detects a change in orientation, its default behavior is to destroy and then re-start the foreground Activity.*
We should considering testing the following:

¹http://developer.android.com/tools/testing/what_to_test.html

²Ibidem.

- *Is the screen re-drawn correctly?* Is the user still able to use the application UI for the main operations of: taking and stopping a video? send private text and audio messages to other users? see feedbacks and audio notifications? add language translations and subtitles to a video? Is the director still able to see all the list of the participants and air their voices and videos? Is he also able to see their *air status* (on air/off air)?
- *Does the application maintain its state?* The application should not *forget* its place when: taking a video, receiving a feedback message, receiving a private chat or audio message, adding subtitles and language translations (the entered subtitle text must not be lost);

Dependence on external resources *If your application depends on network access, SMS, Bluetooth, or GPS, then you should test what happens when the resource or resources are not available.* We should considering testing the following:

- If a change on the availability of the microphone occurs, we should test if a warning is fired when the user is: talking and streaming his voice, sending a private voice message to another user, adding a language translation to a stream;
- If a change on the availability of the webcam occurs, we should test if a warning is fired when the user is streaming a video for a broadcast;
- If a change on the quality of the network connection occurs, we should test if the user will be aware of such event. We should also test if the streamed audio and video are encoded adjusting the encoding bit rate accordingly to the network's quality. Finally, test that the application encodes high bit rate audio and video only when WiFi access is available *since WiFi transfers maximize battery usage compared to transfers over 3G or EDGE*. We can use the emulator to test network access and bandwidth³.

6.2 Web TV Website

As suggested by the Google TV Developer website⁴, we are going to present guidelines to be reviewed when deploying Web TV website. As already mentioned in Chapter 4 Web TV and Website of **ioCast** will share the same design. The Web TV website is nothing but a Website for Google TV, thus the following tests apply for the Website of **ioCast** too.

³<http://developer.android.com/tools/help/emulator.html#netspeed>

⁴https://developers.google.com/tv/web/docs/optimization_list

- Displaying**
- Site is tested and confirmed to work correctly on both 720p and 1080p resolution;
 - Site renders properly with auto-zoom enabled;
 - Video dimensions are adjusted according to their resolutions. Auto-zoom is not enabled automatically for videos. According to the their resolution, videos are displayed and rendered with no significant pixelation for the best visual experience.
 - Site uses relevant visual queues to indicate that content is loading. This is especially true for the loading of the audio and video strams.
- Design**
- All text elements are legible (i.e., text is large enough to be easily read from 3 meters away): stream's informations, podcast's information, avaiable subtitles and languages, search text boxes;
 - Text must not be smaller than 18pt on 720p and 24pt on 1080p;
- Navigation**
- Site (including Form elements) can be navigated easily with D-Pad (directional pad). Test the navigation on the browsing windows for broadcast and podcast, and for the selection of subtitles and languages;
 - Display full screen capability via icon/menu for the streams's playback, only for high resolution streams;
 - Display streams and podcasts's play length and current play location;

Discussion

I can conclude that I have only partially reached my goals: the design of a software framework for **ioCast** has been presented in this project, but a prototype for testing all the **ioCast**'s requirements has been sketched only, as outlined in [5], but not fully implemented.

The main user interfaces, and the core system for the media server have been designed. Those two components, described in Chapter 4 and 5, will be the basis for the developing a testable and fully featuring prototype for **ioCast**.

Some considerations on the implementation of **ioCast** have to be discussed. The *Analytics system* for **ioCast** requires a deep understanding of the Google Analytics framework¹ and how the framework can be used with the smartphones and the Web TV websites. This task has been left aside, as the analytic system is not essential for the goals of this project. This can be part of a further work for **ioCast**.

The same applies for the *Advertisement system*, where injecting advertisements over multimedia streams, by recognizing their audio content, requires a deep understanding of the **FFmpeg** system regarding the audio signal processing, and the implementation of an interface to the Google AdSense² for injecting the

¹<http://www.google.com/analytics/>

²www.google.com/adsense

advertises according to the audio's content. Those tasks have also been left aside, as *making money* out of **ioCast** is not an essential part of the goals for this project. Nevertheless, the advertisement system represents a potential revenue stream for this project, thus it has to be considered part of further works: it might be very useful in case **ioCast** will get to the market;

Considerations on the database have also to be done: managing all the **ffmpeg**s instances with a **MySQL** database might introduce latency and delays to the process of creation and cancellation of streams on the media server. This means that users might experience unpleasant delays when using **ioCast** with smartphones. In any case, introducing the database is crucial for managing all the **ffmpeg**s instances, but further works might examine new solutions for this problem.

Transferring information from the smartphones to the media server works fine when using **HTTP**. But more advanced protocols than **HTTP** could be used for improved further works on **ioCast**. In particular, I would definitively choose *Real-Time Transport Protocol* (**RTP**) which is defined by *The Internet Engineering Task Force* (**IETF**)³. The protocol has got many versions, and each of them is defined for transporting data for a particular media format, including **H.264**, **Mpeg-1/2/4**, and **MIDI**. **RTP** is usually used in conjunction with the *RTP Control Protocol* (**RTCP**)⁴. Used together, **RTP** will carry media streams, while **RTCP** will provide transmission monitoring, quality of service and synchronization of multiple streams. Future works for **ioCast** should definitively consider choosing **RTCP** and a version of the **RTP** protocol as transportation protocols for the streams coming from the smartphones. Moreover, **FFmpeg** can handle the two protocols, and using them might benefit the design of the media server by removing the database, thus minimizing the above mentioned unpleasant delays that users might experience when producing media streams with their smartphones.

A last consideration on the implementation of **ioCast** must be done. Further works for **ioCast** might include the possibility of using **ioCast** together with other systems, for example streaming media both from smartphones *and* from computers that run other streaming software for web radios. The design of **ioCast** already includes the use of **Ices**⁵ for streaming to web radio servers, but supposing the use of other streaming software together with **ioCast**, might need a re-design of the media server.

³www.ietf.org

⁴**RTCP** is also defined by The **IETF**

⁵<http://icecast.org/ices.php>

A consideration on the design of the user interface must be also done. Adding subtitles or audio language tracks to a stream is a quite complex process that requires a great user experience. An improved user interface design, better than the one presented in this project might be needed as further work, for giving users a better user experience in such operations.

Last but not least, a consideration on the whole work of this thesis: many sections of this thesis have been written following the guidelines outlined in [7]. I also did choose to apply aspects of *Lean Principles* to improve the user experience when using *ioCast*. As described in [5], lean principles gave me an easy and effective methodology for designing *ioCast*, by following three basic steps:

1. **Collaborative problem solving:** this phase came from my real needs in the context of the Internet web radio. From that experience I got the input for the next phase;
2. **Sketching:** producing a low fidelity design of *ioCast* to get an idea of what its features have to do;
3. **Prototyping:** the main user interfaces and the core system for the media server have been designed. Such design will be the basis for the developing of a fully featuring prototype for *ioCast*.

The last phase described in [5] *Pairing up developers and designers* is essential to really bring alive *ioCast*. This last phase is missing in this thesis's report, and it is part of the further works for *ioCast*. It will be a verifier for all the work conducted in this project, and it will show whether or not the goals of this project have been fully reached.

APPENDIX A

A brief business plan

This chapter introduces a brief business plan for IOCAST. The content of this chapter is build upon the work of [1], and it is a reviewed version of a document that has been presented on January 2013 at the *Venture Cup Idea Competition*¹ under the *Mobile & Web* competition category. This document has been published before the real goals of this project thesis have taken form, thus some differences between the two ideas occur.

A.1 Background

The idea of IOCAST comes from the world of amateur Internet radio broadcasting. Amateur Internet Radio broadcasters go on air on the Internet by using web radio servers (as for example *Icecast* www.icecast.org or *SHOUTcast* www.shoutcast.com) and by using software tools for professional and amateur radio broadcasting which cost money, and are often hard to set up. Moreover, setting up a studio with good sound instrumentation costs time and expertise. Thus, it is not easy for the non expert users to get started broadcasting a radio show, and those who actually can, do experience some problem. Among all they cannot communicate each other privately without disturbing nor interrupting

¹See www.venturecup.dk/idea-competition/

the show (very often shows are aired from small living rooms). The same problem occurs with people outside the *studios rooms* whom also cannot easily be aired to a radio show. Broadcasters do not have any easy software or hardware solution by which they can broadcast their own live videos of the radio show while they are on air. Web radio community have also got some problems when listening to web radio shows, for example they might need a translation of a show when listening to it, thus enriching their listening experience.

A.2 Business Idea

As we have seen in Chapter 1, there is a need for a new product by which amateur broadcasters can easily air their voices and video without using complicated and expensive software, nor by buying expensive and complex instrumentation. Broadcasters need to air their shows using handy devices that work both from inside and outside their *studio rooms* locations. Directing a show must be an easy task for the broadcasters community; And audiences shall listen, interact, comment and personalize their listening in a new way, by having a rich user experience. This idea embrace the world of web radios, Web TV and the common digital television: this means that IOCAST can be used for bringing alive the participatory journalism concept, where participants are reporters, audience, editors, and have the ability to change their roles, and where multimedia content might not be filtered by any mediator before getting to its audience[8].

A.3 Value proposition

IOCAST is an easy to use broadcasting software for smartphones. Broadcasting multimedia content from any place in the world will be an easy and exciting experience. Anyone will easily direct a broadcast using a simple and friendly software for smartphones. Creating *users's generated* media content is with IOCAST a new - more personal - experience for the users. *ioCast's* streaming philosophy is different from any other streaming solution because it not only does solve the hassles of setting up professional broadcasting stations, but it can also connect users's generated media content to Web radios, Web and digital TV in an unique product. IOCAST is a solution for bringing alive the participatory journalism.

A.4 Customer profile

Typical potential customers are amateur and non amateur broadcasters who need a cheap, handy and powerful product to get the most out of their broadcasting experience. This includes Internet radio and Web television broadcasters. IOCAST can be also used by any media broadcasting companies that want to make their audience broadcast *for* them (think of **ioCast** as a public *live* broadcasting platform for radios and TVs, where audience can act as reporters).

A.5 Market

IOCAST will act on the market of Internet Radio stations: the size of this market is bright, for example **SHOUTcast** has got 50,000 free internet radio station with more than 700,000 people streaming audio every day all around the world. This market also includes radio stations owned by big media companies.

A.6 Competitors and Strategic Partners

There are many solutions providing desktop and mobile software for *only* accessing and organizing users's media content for web radios, Web TV or media content in general. On the other hand some application do take an active part on user content's live broadcasting where the users take control of their broadcasts in many ways² but none of them embrace the concept of participatory journalism, as IOCAST does. Therefore, IOCAST can be seen as to act on an uncontested market space (aka *Blue Ocean*[6]) but it must not underestimate the strength and the technological knowledge of the above competitors, that could quickly solve the problem IOCAST is addressing. **ioCast**'s *blue ocean* is not an excuse to act alone on the market, but an opportunity to get strong strategic partnerships. In fact, **ioCast** should go into the market by obtaining partnerships with big web radio service companies, as well as obtaining partnerships with media companies which would like to use **ioCast** as an open broadcasting platform *of* their audiences.

²Read Section 2.1 for more details on mobile video streaming solutions.

Bibliography

- [1] Y. P. Alexander Osterwalder, *Business Model Generation: A Handbook for Visionaries, Game Changers, and Challengers*. Wiley, 2013, ISBN: 9781118656402 (cited on page 51).
- [2] *Ffserver documentation*. [Online]. Available: <http://ffmpeg.org/ffserver.html> (cited on page 37).
- [3] M. Fowler, *UML distilled : a brief guide to the standard object modeling language*, 3rd ed. Addison-Wesley, 2004, Series: Addison-Wesley Object Technology series, ISBN: 0321193687, 9780321193681 (cited on page 16).
- [4] *Google tv developers guide*. [Online]. Available: <https://developers.google.com/tv/web/> (cited on page 25).
- [5] J. Gothelf, *Lean UX: Applying Lean Principles to Improve User Experience*. O'Reilly Media, 2013, ISBN: 9781449311650 (cited on pages 25, 47, 49).
- [6] W. C. Kim and R. Mauborgne, *Blue Ocean Strategy / How to Create Uncontested Market Space and Make the Competition Irrelevant*. Harvard Business School Press, 2005, 240 p, ISBN: 1591396190 (cited on page 53).
- [7] S. Robertson and J. Robertson, *Mastering the Requirements Process: Getting Requirements Right*. Pearson Education, 2012, ISBN: 9780132942843 (cited on pages 9, 13, 17, 49).
- [8] C. W. Shayne Bowman, *We media: how audiences are shaping the future of news and information*, 2003. [Online]. Available: <http://www.hypergene.net/wemedia> (cited on pages 1, 2, 52).