FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# Platform For Interactivity in Concert Rooms Using Targeted Clusters of Mobile Devices

**Rafael Rocha Henriques**


Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Rui Rodrigues

Second Supervisor: Rui Penha

July 29, 2015

# Platform For Interactivity in Concert Rooms Using Targeted Clusters of Mobile Devices

**Rafael Rocha Henriques**

Mestrado Integrado em Engenharia Informática e Computação

July 29, 2015

# Abstract

Technology is nowadays present in most parts of our lives. There are few places where each of us would dare going without our "beloved" smartphones or any such device that would allow us to quickly and easily communicate with others, connect to a network, or simply entertain ourselves during dull moments.

This dissertation intends to make use of this situation. Given that everyone carries these devices everywhere, we might as well make use of them. The idea is to make all the smartphones useful to create a new show dimension, inside a concert room. To make an application that works on a wide range of mobile devices (the audience's devices) so everyone could literally be a part of the show. An application that would aim to elevate the show itself, that would take it to the stands. By locating people inside the concert room, we could possibly create a whole new spectrum of entertainment, without ruining the concert or the show going on, but merely extending it.

The application should be able to locate every device connected to a local venue network and from there reproduce different media contents on devices located in different zones. This would allow for a completely different interaction between public and performers.

**Keywords**: concert room, smartphone, interaction

# Resumo

A tecnologia está, nos dias que correm, presente em grande parte das nossas vidas. Há poucos sítios onde a maioria de nós se atreveria a deslocar sem os nossos "amados" *smarpthones* ou qualquer tipo de dispositivo que nos permita comunicar com terceiros de forma rápida e simples, conectar a uma rede ou simplesmente entreter-nos em momentos de enfado.

Esta dissertação tem como intenção fazer uso da situação descrita. Dado que uma parcela tão grande da população possui e transporta constantemente consigo um dispositivo deste tipo, faz sentido maximizar a sua utilidade. A ideia é fazer com que esses mesmos *smartphones* se tornem num meio de criação de uma nova dimensão num espetáculo, numa sala de concertos. Construir uma aplicação capaz de funcionar num vasto número de dispositivos móveis, os dispositivos trazidos pelas pessoas que constituem a audiência, para que todo o público consiga fazer parte do espetáculo. Uma aplicação que apontaria para a elevação do espetáculo em si mesmo, que o levaria para a audiência. Ao localizar as pessoas dentro da sala de concertos, ser-nos-ia possível criar todo um novo espectro de entretenimento, sem arruinar o concerto a acontecer, mas sim estendendo-o.

A aplicação seria capaz de localizar cada dispositivo ligado a uma rede existente numa sala de concerto e a partir daí reproduzir conteúdo multimédia em dispositivos espalhados por diferentes zonas da sala. Isto permitiria uma interação completamente diferente e inovadora entre público e *performers*.

**Keywords**: sala de concertos, *smartphones*, interação

# Acknowledgements

For the many wars fought together, against hardware and software of all kinds and sizes, for all the problem solving, for the availability and good disposition, for the knowledge, but also the good times, I am grateful to Carlos and Alexandre, brothers in arms.

For allowing me to be a part of this project, of an original idea, for the guidance and help in times of need, for the creative ideas and pushing the limits, I am grateful to professors Rui and Rui. Also, cannot leave out the rest of the team.

For all the quality learning it provide me, for all the friends I met during this huge stage of my life, for all the experiences, even for allowing me to go abroad for a semester to such a wonderful place and still accepting me back, but mostly for making me the engineer I hope to be, I am grateful to FEUP and all the teachers that I ever crossed paths with.

For always being supportive, for always providing for me, for helping me reach all the things I could have never reached without them, for always being there, for all the love they always showed me in their own ways, even in the hardest days of their lives, I am so grateful to my mother and my father. Especially mom, for always caring that bit too much.

For the persistence, for never giving up, for believing me more than I do, for knowing me better than I do, for the support, for the light shed, for never having left anything unsaid, for the reality checks, for sticking through more than anyone would have and least of all, for the creativity, the feminine side, the graphical help even from afar, I am deeply grateful to Mónica.

For the unintended help you kept providing me, especially while writing this report, I am grateful to Bruno and Ricardo. Special word for the latter, for all those months we spent together, for you were there when I chose this theme and all the great times we lived together.

For making me the person I am today and keeping me up when times were hard, for the motivation, for the things they taught me and the advice they gave me, for the experiences I had with them, I am so grateful to all my true friends. They know who they are.

For leaving a part of you with me, for being a part of my life, for taking a part of me with you or not even remembering me, no matter if it was good or bad, I am grateful to everyone I have ever met.

Rafael Rocha Henriques

*"From swamps to stars, we dreamed far,*
*They called it far-fetched, we called it ours.*
*We called them lessons, they called them scars,*
*They call it blessings, this work was hard,*
*That is where we dwell."*

Arian Foster

# Contents

# List of Figures

LIST OF FIGURES

# List of Tables

# Abbreviations

| | |
|---|---|
| AP | Access Point |
| API | Application Programming Interface |
| APK | Android Package |
| ASCII | American Standard Code for Information Interchange |
| BSSID | Basic Service Set Identifier |
| CSV | Comma-Separated Values |
| DHCP | Dynamic Host Configuration Protocol |
| IGMP | Internet Group Management Protocol |
| IP | Internet Protocol |
| OS | Operating System |
| PD | PureData |
| QR | Quick Response |
| ROM | Read Only Memory |
| RGB | Red Green Blue |
| RSS | Received Signal Strength |
| TCP | Transmission Control Protocol |
| UDP | User Datagram Protocol |
| XML | eXtended Markup Language |

# Chapter 1

# Introduction

This is the year 2015 and nearly one in every five people owns a smartphone worldwide. In developed countries this number gets even bigger and more significant. Technology is fast evolving and every day something new is created that gets our attention, improves our lives and extends the realm of our possibilities in some way. Every day new smartphone applications are developed.

A smartphone is nowadays equipped with numerous sensors and is able to perform a wide range of operations that would be unimaginable a few years ago. This is a reality and it is for us to take the best possible advantage out of it. Most people are currently not only able to afford these devices, but also want to do so. These devices allow for a person to be reached at any time, anywhere and in many different ways. The possibility of constant internet connection, makes it easy for owners to stay in touch with the world, even beyond friends and family. It is more than being able to receive a call or a text message. Smartphones brought us the possibility to be constantly and world-widely connected with each other. People can receive and exchange emails from any place in the world with a network connection, be it a Wi-Fi signal or even using the cell phone's data broadband connectivity.

Smartphones have already been brought inside the cultural world. Smartphone orchestras have been created in multiple places and countries. These orchestras make use of smartphones in the place of real instruments and are able to perform live acts. It is not to ever be seen as a replacement for classic orchestras, but merely a new type of spectacle.

The aim of this project is to use audience's smartphones inside a concert room, knowing where each person is seated and from there try to create a new show dimension by creating new possibilities, allowing more freedom to the artist and putting together a more interactive show for everyone.

## 1.1 Problem Description

Bringing smartphones into the picture to allow for inside concert rooms interactivity is something that is not well explored. The problem is mainly related with how to make it work. Currently, smartphones inside concert rooms are mainly used for filming the show that is going on and mostly distracting people from the show itself, whether it is because someone is too focused on

the screen trying to get the best picture, or because someone behind them cannot actually see the show because of the smartphone in the air. It can also be seen as a distraction because it can bring the outer world inside the concert room if people allow it.

But not everything is bad, if advantages are taken out of it. Making use of smartphone's sensors it should be possible to know their position inside the concert room and from there allow different zones of the audience to be part of the show in a different way, by reproducing different types of content on their devices, content related to the show going on, defined by the composer, for that specific show.

This will be done in two distinct phases.

First, a framework that should work on mobile devices needs to be developed. A framework and not a simple application. This framework should be viewed solely as a basis that would allow artists and composers to develop their own pieces, easily applicable, allowing them to manage and create the show. It does not intend to take any part of the creation process away from the artist, only to confer them a new freedom level, extending the show to the audience, by using the smartphones brought in by the people themselves.

It also needs to be accessible for performers, meaning that it should be easy to strive and quickly work their way around it.

Second, a method to know where each device is located inside the concert room, possibly associated with the ticket seat, will be needed. The intention behind this is to give the performer the possibility of adding another level of dynamic to their show. This way we would be satisfying the previously mentioned objective of being able to split and divide zones or spaces inside the concert room, being able to form patterns or playing different sounds.

All of this should be brought together with the support of a Wi-Fi network that allows all of these smartphones to act as clients to a server responsible for controlling what and where things are happening. Both the server and the network must be able to handle at least hundreds of clients, given the normal spectrum of a concert room capability.

## 1.2 Motivation and Goals

The aim of this dissertation is to transform the gadgets, so common nowadays, into a part of the show. Using all the capabilities these devices generally offer, and aiming for the lowest common denominator, something that every smartphone should be able to do, like playing sound clips, reproducing video, showing images on the screen or simply a colour, it aims to bridge the audience and the performer.

Linking the performer to the audience and the audience to the live show going on is a challenge. Our goal is to bring this challenge a step closer to reality with this dissertation, making use of, so far, not so explored areas and technologies.

Performers are a very important part of this project as well, since they will be the ones who will have to use the framework and create the interactive part of their shows, so it must be accessible to

them. The aim is to find a technology that performers are at least somehow familiar with in order to make it easier for them to adapt to the framework and having a shorter learning curve.

## 1.3 Team Work

It should be mentioned that this thesis is part of a larger project, which encompasses several components. These components are mainly divided into three specific divisions:

- Network Component
- Multimedia Component
- Mobile Application and Infrastructure Component

Given the different areas of specialization needed for the development of each component, each of them will be handled by a different thesis.

This thesis focus mainly on the Mobile Application and Infrastructure Component.

This will of course call for a special coordination necessity, a very well organized work distribution and a big sense of responsibility between team members, since the failure of one side may imply the failure of the whole project.

Parts of the project such as multimedia content development and network architecture that are not in the scope of this thesis will only be addressed as far as necessary for contextualizing and explaining the full system.

## 1.4 Dissertation Structure

Beyond this introduction, this dissertation contains five other chapters.

Chapter 2 describes the current state of the art, presenting some relevant related projects.

Chapter 3 aims to give an idea of the concept presented and an overview of the problem, to introduce the following chapter about the solution.

Chapter 4 describes the solution implemented in order to solve the initial proposal of the dissertation.

Chapter 5 describes the tests done to the built prototype and presents the results analysis and interpretation.

Chapter 6 states the main conclusions taken from the developed project and also the future work to be done.

# Chapter 2

# State of the Art

This chapter presents the state of the art of all the technologies involved and considered during the initial approach of this dissertation, but also all the the technologies that proved to be relevant during the development phase. This includes visual programming languages, aiming to create a good and easy experience for composers that don't feel comfortable with coding; related technologies that already exist, such as laptop and mobile phone orchestras; and finally, indoor location methods, as a reference for the possible development of a dynamic location method, given enough time.

## 2.1 Framework Construction

In this section will be presented decisions made when considering possibilities available for technologies that would allow for a universally accessible framework to be used by artists and composers. These technologies should be easily understandable and present a short learning curve.

When looking for the best technologies to build the framework there were certain aspects that were taken into consideration, such as the need to support multiple devices with different operating systems, meaning that it should be cross platform, and the need for adaptability. The intent is not to build an application, but rather a framework that would allow the artists to simply patch it with content of their own authority. To reach this, multiple visual programming languages were taken into consideration. Below, these possibilities are presented and the final choice is thoroughly explained.

### 2.1.1 Visual Programming Languages

Visual programming languages (VLP) are a type of programming language developed to allow users to create programs without the need to learn actual code. This is done by basically creating a graphical interface, much more intuitive then the code itself and a significantly easier to learn and master.

This is especially useful to be used by people who were not educated as software technicians or engineers but still feel the need to develop computer programs.

There are many and more available visual programming languages nowadays, each one serving its own purpose. In this context, we are mainly looking at those created for multimedia content creation purposes. We are looking for a simple, VLP, something that artists should be at least somewhat familiarized with, that would not take them a hard time to learn, understand and being able to use.

Given these parameters, plus the necessity of being able to present not only sound or visual contents, but both simultaneously, the conducted research evidently proved that Max and PureData would be the indicated languages to apply in this context.

### 2.1.2 Max and Pure Data

Max is the mainly spread visual programming language when it comes to multimedia content. It has been widely used in numerous different projects and ideas and it is still used nowadays.

It is commonly referred to as a building-blocks environment, consisting of a canvas and a set of graphical modules, each encapsulating a code block responsible for a given function.

Each of the mentioned modules is equipped with built-in output and input of data specific to its functionality.

Interconnection between the different modules of a program is achieved by connecting lines, called patch-cords, which graphically symbolize the data flow inside the application.

Some examples of successful Max implementations can be illustrated by

- Audio Plugin Player [APP], which allows users to play a different number of musical instruments directly from a Mac computer.
- TapNTempo [TNT], a fully customizable metronome, also developed for Mac computers.
- pMix [pMi], a performance, composing and sound designing tool fully equipped with an intuitive visual interface that allows another user level of abstraction, with the same power to create.

PureData [PD1] is mostly viewed as a natural evolution of Max, when it comes to multimedia visual programming languages. Created by the same author, but developed to allow a broader sense of creation freedom, given that it is open source and the easier control it gives over other components that are not sound ones.

Being an evolution from Max means that it will not be that different to handle or learn and so the familiarity artists generally have with Max, they should also have with PureData, or at least it should not take them long to grasp the basic concepts, which is essential in such a project.

Besides this, a lot of communal libraries have been created, including libpd [lib], which is a library that allows for integration with both Android and Apple devices with a higher level of abstraction. This was one of the main concerns in the project, that the framework would be able to perform equally on both platforms.

Some interesting projects developed with PureData for smartphones are presented as follows:

- CanOfBeats [COB], an Android application that allows users to instantly create hip-hop beats.
- GarageAcidLab [GAL], an application developed for both Android and Apple devices that allows to create Garage/Acid music algorithmically.
- RjDj development team [RjD], a team of developers that has created multiple smartphone applications based on PureData. Their main aim is to create virtual sound environments for users, based on the real environment they are in. The applications use smartphone sensor data to read the surrounding ambiance and algorithmically reproduces new sounds, giving the user the sensation of being immerged in an alternative reality.

## 2.2 Cooperative, Colaborative and Networked Music Systems

Using recent technology in a concert room environment is not an unseen event.

In fact, there are some vastly known orchestras that recur to smartphones or laptops instead of the traditional human/instrument components. These orchestras are built based on inter device communication, but never truly insert an interaction with the audience.

Some examples of laptop existent laptop orchestras are:

- The Princeton Laptop Orchestra (PLOrk) - Created in 2005 at the Princeton University, it is composed by a set of 15 computer-based meta-instruments that are orchestrated with the usage of network protocols and synchronization. [TCSW06]
- The Stanford Laptop Orchestra (SLOrk) - In all aspects similar to the previously presented PLOrk. [WBOH09]
- The Carnegie Mellon Laptop Orchestra (CMLO) - a collection of computers that communicate through a wireless network and collaborate to generate music. The main variation presented when compared to the PLOrk is the fact that it does not have a set number of computers/instruments. They can be added or removed at will. [DCA+07]
- The World Laptop Orchestra (WLO) - Composed by 50 performers, each equipped with a laptop instead of a traditional instrument. Its main goal was to explore spatialization techniques independently of the orchestra's physical layout. [HABF08]
- The Linux Laptop Orchestra (L2Ork) - In part inspired by the successes of PLOrk and SLOrk and in part encouraged by the rapidly developing Linux hardware and software support. The main difference is the costs involved, in which the Linux orchestra performs with a significantly smaller budget. [BMSM10]

The concept of Smartphone orchestra derived directly and naturally from the already existent laptop orchestras. The most renown one is MoPho, the Mobile Phone Orchestra of CCRMA [WEP08].

This orchestra is composed by 16 similar mobile phones, all of them Nokia N95. Developed mainly in C++ and Python. C++ was used for audio synthesis and hardware access; Python was used for graphical interfaces.

Lower computational capabilities of these devices when compared to laptops, force significant constraints upon the pieces composed to these devices.

## 2.3 Indoor Location

Indoor location is a theme that is becoming more and more explored in recent years. Solutions have been presented but none seems to qualify for every different kind of use or conditions available. Global Positioning System (GPS) easily emerged as a dominant force when it comes to outdoors location. Its availability worldwide and easy connection due to the amount of satellites available makes it a strong and reliable source of location. While it is very useful in an outdoor environment, GPS technology becomes nearly useless when brought indoors.

This is true mostly for two factors.

One, GPS is based on satellite connection and consequent position triangulation taking into consideration the relative position of the GPS device in relation to at least three orbital satellites, meaning that the finding of these satellites and connection is an essential part of the process. Acquiring indoors signal from these satellites is very hard and so GPS location becomes useless.

The second reason why it is not adequate for the project in practice is the precision of this location method. While it is precise enough to locate us in open, broad spaces, like in a road or a street which are several meters wide, it would not be precise enough to locate someone inside a room that is only some square meters wide.

A lot of the proposed solutions use more than just a smartphone, requiring different extra types of equipment in order to work, such as radio tags, wearable devices and other. These methods imply the installation of a dedicated technical infrastructure which will not be available in this dissertation and would also go against the principles of it, being that it would work only in a specific environment, prepared for this kind of situation. What we are aiming for is creating something that would adapt and fit to different scenarios with minimal needed requirements.

Realising all of this and after some research in the area, three main methods were found when it comes to indoor location that would adapt to smartphone capabilities and these are based on Wi-Fi signal strength, Dead Reckoning methods and Acoustic methods. All of these will be presented and discussed in the following sections.

### 2.3.1 Wi-Fi signal

Wi-Fi signal based location techniques are the most used and explored option when it comes to locating indoors. This technique is widely explored and reported and has been developed over

recent years, having its precision and locating methods increased.

The most widely used method for using Wi-Fi signal location indoors is the Received Signal Strength (RSS) fingerprinting method. This method consists of two different implementation phases. First, an offline phase, where the room or building where the technology is to be implemented must be previously thoroughly scanned and the intensity of Wi-Fi signal reception must be mapped over a previously existing map of the place.

This allows for the second phase, the on-line phase, where devices to be located can just connect to the Wi-Fi network available and by measuring the signal strength, compare it to the existing signal strength map and find out its position in relation to the building or room.

The precision of this method by itself is not enough to be able to locate someone precisely inside a small room. It is usually enough to tell which room inside a building someone is in, but no more than that, given the 3-10 meters precision radius. Because of this, complementary methods are usually required.

This location method demands for a significant number of Wi-Fi access points available, for triangulation to be possible, plus significant network capability. The constant demand for Wi-Fi signal and signal strength measuring is a battery and resource consuming process, which makes the adaptability of this method to the smartphone world highly questionable. Furthermore, the use of this method implies a huge amount of network communication that would prevent the smartphone to use the very same method for communicating with our established and much needed coordinating server. This is the biggest and most important drawback in this technology from our point of view, given the constant need for coordination between server and clients.

"Horus", [YA05] is an example of such a method, and a very accurate one. It claims to reach precisions on a 0.6 meter average, which is a very good number given the limitations of the method described above. A deep study of the limitations of the method made by the author certainly seemed to help when it came to the decision making process, making them able to tackle the right problems. The usage of probabilistic techniques combined with a sense of space continuity allowed them to get a step closer to a reliable technology.

Even though this solution claims to require low computational requirements through the usage of clustering techniques, it was not meant to be used in a smartphone, as it is mentioned in the text it was developed for Windows and Linux operating systems. The solution proposed by [Her06] is that all Wi-Fi access points in a given location should be mapped and well known beforehand. Given this knowledge, by identifying which access points a user can reach at a certain point in space and the strength of their signals, a fair location estimation can be made. The accuracy of this method revolves around the 88% mark.

“ArrayTrack”, [XJ13], is a solution based on multi-antenna Wi-Fi access points that track the angle of reception of the received packets, being able to distinguish algorithmically the path taken by the packets received and whether they came directly from the source or taken reflections on the way. Although it presents fantastic results when it comes to precision, the amount of access points and resources needed are a huge and significant disadvantage point here.

"CUPID" [SLKC] bases its technology on a similar principle, but recurring only to regularly available access points. The goal is to detect the energy of the direct path and the angle of arrival simultaneously and use that data along with user's movement to detect the position. The results shown are not nearly as accurate as the ones given by the fore mentioned "ArrayTrack", but given the fact that no special equipments were bought and that this was an application built for mobile devices, less accurate results were to be expected.

### 2.3.2 Dead Reckoning

Dead reckoning is a location method based solely in a device's available to sensors to determine the relative position in a pre-loaded map in every moment. In order to make this possible, there must be a known starting point that can be identified, for example, by a QR code. By taking a picture of the said QR code, the user is given a position on a building or room and from then on the position will be calculated based on the data received by the accelerometer, gyroscope and any other available sensor, determining the amount of steps taken by the user and the correspondent direction.

A proposed solution [SCM] uses 2D bar codes for initial position identification and map downloading. By simply taking a picture of the mentioned bar code, the user will define its initial position, from which every other consecutive position will be calculated using only the smartphone sensors, with no connection to exterior networks, using a step counting algorithm based on a set of positive and negative peaks in relation to the original smartphone position.

The main problem with this method is that every step taken will result in an estimation of the distance covered and the direction taken, always susceptible to a significant amount of error. This can become a heavy load with the error accumulation over a long route summing up and taking a heavy toll on the final estimation.

Furthermore, it is also susceptible to the device's orientation and position, not allowing for a lot of user movement freedom, or the error accumulation will become even higher and out of proportion.

Given the limitations of this method, it is easy to understand why it is not given a lot of attention as a sole indoor location method. However, it has some potential when used as a complementary method to another technology, given that it can work in cooperation with other type of received data and minimize the error accumulation by considering the other method's location estimation.

A great example of this is "SAIL", [MSLK14]. Using dead reckoning methods, based on accelerometer, gyroscope and compass as a complement for a single access point, Wi-Fi location method, it reduces the impact of the Wi-Fi methods problem in a great extent.

Using a single access point, the distance to it is measured using the time-of-flight, which gives only a radius to the origin point of the signal, but is then complemented by the dead reckoning methods, given the orientation of the user's movement.

Another example of this is the solution proposed by [Bal13] where RSS fingerprinting is used as a general location method, being the specific location given the synchronization of that first

estimation with the calculations given by dead reckoning methods. It is mentioned that one of the biggest obstacles was balancing different solution components. But in the end results under the acceptable range were reached.

### 2.3.3 Acoustic Location

Acoustic location methods are based on acoustic signals sent by sources with a known location inside a building that allow the receiving end to measure the intensity of the received signal determining the radial distance to the source.

This method can be used with triangulation methods, needing more than one input to locate the exact position inside a room, or using the previously mentioned fingerprinting method, where a previous mapping of the intensity of the signal received in every position of the room would need to be made.

It is the only indoor location method capable of presenting sub meter results by itself, using no other complimentary methods. It is also important to highlight the fact that this would need to work at a smartphone level, implying that common microphones available in the most common smartphones should be able to detect and perform reasonably well when using this kind of method. No other kind of extra equipment should be needed.

The possibility of working with ultrasounds would allow for it to be inaudible to the human ear and be mostly undisturbed by other sounds. Also, sound presents the same similar behaviour given the same ambient characteristics, meaning that it will travel at the same slow speed and spread at the same rate, if no significant temperature changes happen, as described in [FCC10].

They used ultrasounds to locate mobile phones inside a laboratory equipped with sound detectors in each corner and using an ultrasound emitted from several mobile phones, under different volume configurations, calculating the device's position using the time-of-arrival of the sound emitted to all the different microphones. Although this is made in an opposite direction of what we would eventually try to perform given the amount of devices we would be looking at inside a concert room at the same time, their study of ultrasound capabilities and limitations presents an important and significant work progress. It shows some limitations with distance and the use of high volumes, which would imply using a significant amount of beacons spread around the room, but given the fact that we are using concert rooms this might not be an unsolvable problem.

There is also an Android application developed, RoomSense [Mir13], that aims to provide instantaneous indoor position estimates in two different levels, both rooms and within-rooms position. This is done by using both the microphone and the loud speaker of the same smartphone and would require both to be in the same side of the device for proper results.

A sound is emitted from the smartphone's loud speaker and measuring the acoustic impulse response received in the same smartphone's microphone. High accuracy was reached when identifying inter room location, but for within-rooms location a previous ambient study was important to reach higher accuracy levels and the variance of room conditions proved to be an important factor. Meaning that just changing some furniture's position would mess with the application accuracy

and in an ever changing environment such as a concert room full of people this would need to be diminished somehow.

Another good example of successful indoor use of acoustic location is (Stephen, 2011). Here, the Acoustic Background Spectrum is used to distinguish room level location. This is based under the principle that every room will have specific background noises that when combined may be nearly unique allowing for the identification of the surrounding environment. An iPhone application named "BatPhone" was built by the same author based in these principles [BP].

With "GuoGuo", [Kai13] proposes an acoustic localization solution based on both time-of-arrival and performs non-light-of-sight identification and mitigation, as an error pruning technique. Although it uses a complex beacon network, this is the closest solution to what we want to achieve. It reaches precisions under the 10cm mark in a quiet environment. Even with a non-optimal environment, results under the 25cm mark were achieved which is a real promising figure when it comes to indoor location. The main drawback of this solution is the amount of beacons necessary to achieve the final accuracy, plus the need of a special central beacon that synchronizes every other one by communicating directly and only with secondary beacons and not directly with the user.

All of these examples were successful in their own way but did not fully apply to the wanted use of the acoustic methods.

This seems to be the indoor location method with the most promising results and accuracy when applied correctly to the intended way of usage. Furthermore, although not exactly under the same circumstances, applications have been built on both Android and iPhone devices that used Acoustic methods to some kind of localization extent, which seems to be a good first step showing that it would be possible, with the right amount of effort and study, to apply it in a different environment.

## 2.4 Conclusion

After going over all of these technologies, some of them clearly stand out when compared to their competition, whether it is because they are clearly better developed and supported or because it is a best fit for the actual work and work conditions that the team will have.

It should be mentioned that due to a lack of time, indoor location methods were not applied by the end of this dissertation. It was decided they should still be included as state of the art technologies, for they were an important part of the initial approach and are considered an important future work task.

Bearing that in mind, when it comes to the location technology methods studied in this chapter, acoustic location methods seem to be the best fit for the project in hands. Not only because of the level of precision needed and wanted, but also because, since the project's context already is related to concert rooms, these mentioned concert rooms should be better prepared for this kind of solution and have a large part of the equipment needed already available and nearly ready to use for the purpose wanted.

The framework construction problem, given the mentioned needs for easy adaptability and multi-platform functionality, PureData seems to stand out as an easy and obvious choice given the competition faced. Not only because it was developed for sound and other content, which most of the other visual programming languages are not readily available to perform, but also because libraries already exist which implement the multi-platform support, allowing us a new level of abstraction when developing software for different brands and operating systems.

# Chapter 3

# System Design and Architecture

This chapter presents the system design and architecture used to develop a framework for mobile devices that should help bring some interactivity inside a concert room. This system was mainly implemented for the Android operating system, although it is meant to be ported to iOS as well in the near future.

The first section of this chapter describes the total scope of the framework developed, the idea behind it all and the different phases of its life cycle. After this, a section that describes the necessary components for the functioning of this system is presented. Finally, there is a section describing the location method established.

## 3.1 The Idea - Full Scope

The aim of this dissertation is to bring some interactivity inside a concert room, as previously explained in this document, and allowing a music composer to take a part of the show to the audience, through smartphones brought by the very same audience.

In order to motivate people to use the application during the concert and because there are configurations that can be done previous to the concert day, it was decided that it all should start at the moment the ticket for the concert is bought. This event should trigger the beginning of the user interaction with the application. The installing of the application will be highly incentivized and from the moment it is installed, ideally some days before the concert, its own configuration should begin.

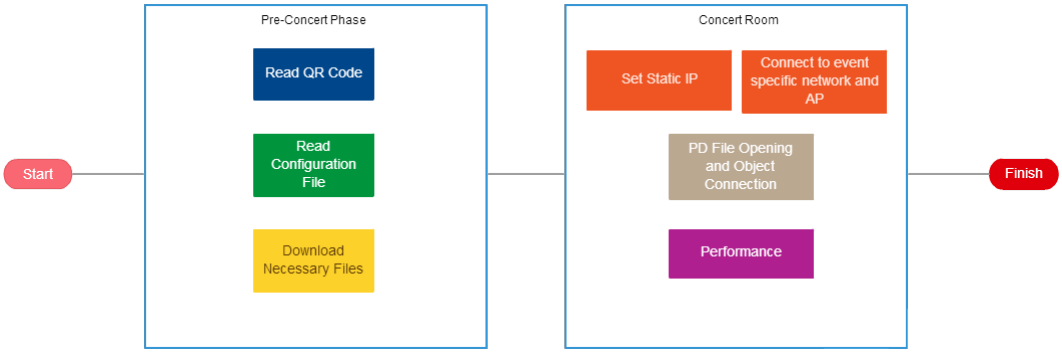


Figure 3.1: Conceptual Division of the Application

Figure 3.1 represents a conceptual map of the application, showing different tasks that need to be performed during different phases of the application life cycle, shown in the picture below 3.2. Colors attributed to each of the items allow for an idea association between both figures.


Figure 3.2: Application Flow Throughout its Life Cycle

A QR Code should be printed or attached to the ticket bought that would give the application some of the information needed for the configuration.

Files that need to be downloaded for the functioning of the application during the concert, should be downloaded during this pre-concert stage of the application.

Basically, everything that has to do with the concert should be readied and configured in a stage previous to the concert room entering. The parts of the solution implemented that refer to this particular part of the system are described in detail in section 4.3.

In the day of the concert, inside the concert room and following the configurations that the application has set during the pre-concert phase, everything should automatically be set up, such as the local network access. These configurations and how they are set up are described in section 4.5.

The connection to the multimedia content, that should have been dynamically downloaded already, at this point in the application flow, will also be realized in this phase. This is described and explained in section 4.4.

As of the concert day, smartphone's location inside the concert room should be known from the QR code read into the application and the audience should be mapped into zones that would play different roles in the show.

Picture 3.3 is an overview of the concert room. The composer and musicians will be on stage, along with the server machine that will be aware of the audience. The people are represented in the figure by the smartphones, mapped into zones, where each zone of the audience will be connected to a different AP. This allows the network component to easily find the people inside a specific zone and target them with specific messages.

The image also allows for a general understanding of what this system may allow an artist to create. By having the audience divided into zones, different content may be reproduced across each zone, sound fading and crossing may be done by playing a sound in one zone and then the next and so on, the creation of a pixelated image, it is up to the artist to invent different contents capable of taking advantage of this concept.

Each of the system's components will be thoroughly described and explained in sections to come.

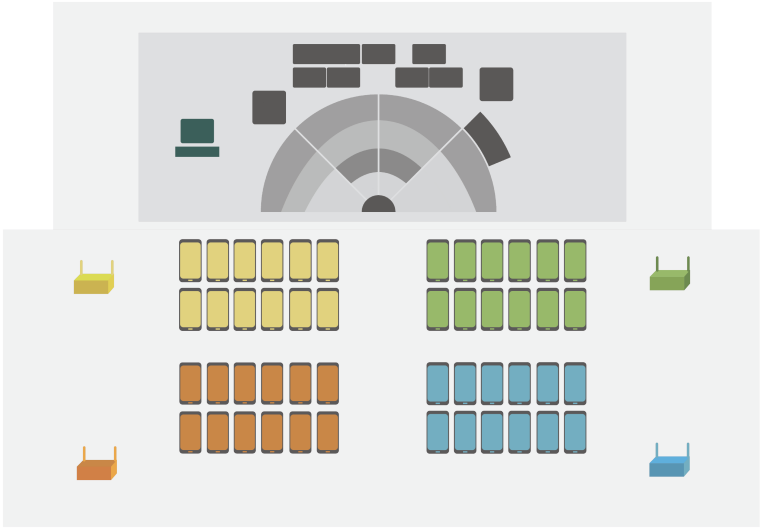

Figure 3.3: Concert Room and System Composition Overview

## 3.2 Architecture

The proposed framework strongly depends on three main different areas that need to interact and perfectly fit into each other, in order for the system to act stable and perform in a reliable way.

This dissertation is not meant to simply be a smartphone application. The idea was always to build more of a framework and less of a common application. This means that it should be built with less restrictions and more degrees of freedom to be changed and molded in the future, able to scale or be taken and used in different contexts.

As previously described in this document, the objective is to bring some more interaction inside a traditional concert room. This is to be achieved using smartphones that nearly everyone

carries these days. But smartphones alone cannot do it. Communication between the stage and the audience is essential, as well as multimedia content that allows for expression, for the interaction to be visible and felt throughout the room.

So, as shown in figure 3.4, the system is composed of a server, a client and a network component that bridges both of them and allows for communication between them.

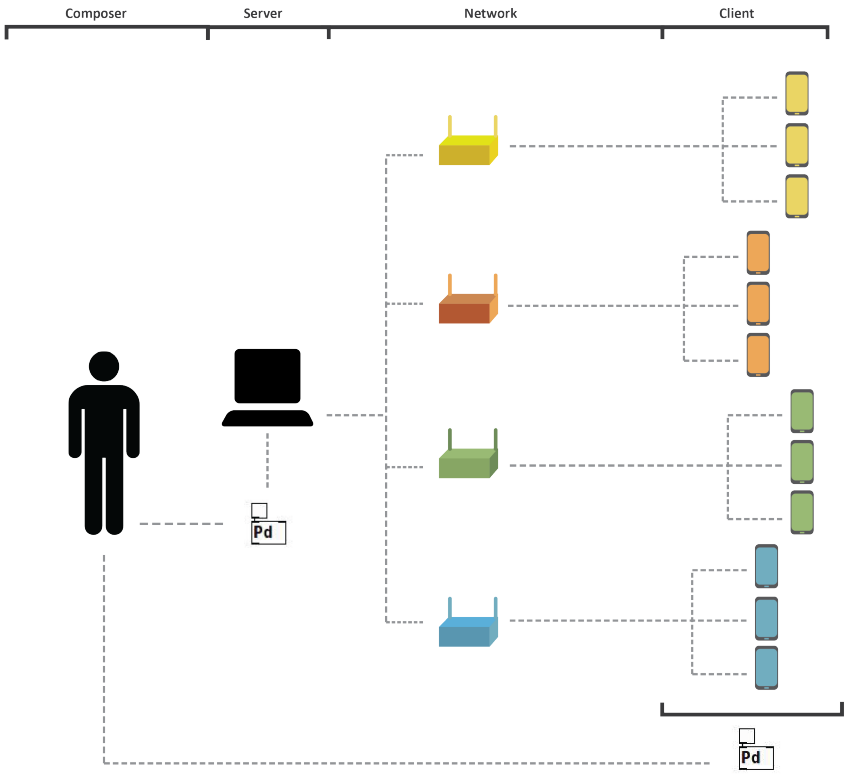


Figure 3.4: System Components

The picture 3.4, illustrates the whole system composition.

It is also understandable through the figure that one unique server will be running and distributing instructions to the network component. The network component will be composed by multiple routers, distributing the messages through all the reachable clients, each configured to handle specific zones of the concert room, distributing the network usage and so making it more available and less likely to become over used and so improving the chances of every message going through it being properly delivered to the target recipient.

It can also be taken from figure 3.4 that every access point will be responsible for a certain amount of clients, located in a specific zone of the concert room.

The network component will be responsible for handling the said distribution and knowing not only where each message is heading to but also how to make it get there.

An essential part of the system, also represented in figure 3.4, are the Pure Data patches present in both sides of the system, the server and the clients.

These patches consist of embedded pieces of Pure Data code blocks. On the server side, the leading patch is responsible for the content and instruction distribution throughout the different zones of the concert room, making it possible for the composer to build a whole show in the stands. It allows for a more dynamic concert, taking some parts of it and reproducing them in the audience's smartphones.

Each of these smartphones will also have a Pure Data patch that is capable of, after receiving messages with instructions from the network, processing them and reproducing the given orders. This patch is not similar to the one placed inside the server, instead it will be an extension of it, like two pieces of a puzzle that would fit together.

Furthermore, not every smartphone in the room will need to implement the same Pure Data patch, so long as the server patch is aware of the different pieces available and where they sit inside the concert room, so it can communicate properly with the right clients.

While keeping an eye out for the basic system architecture, it should mentioned that this dissertation is responsible for the client's side implementation only. Mentions of the other components found in figure 3.4 will be made throughout the rest of this document because, as previously mentioned, this is part of a team effort, where each individual part plays a very significant role for the functioning of the total system.

## 3.3 Localization Methods

Even though a dynamic and more complex localization method would have been useful to detect eventual position changes in people's location inside the concert room, some limitations eventually forced the decision of going with a simpler, more direct method of knowing where each device is.

With that being said, the method chosen, for now, was a seat number location method. Being that as of the moment, the aim of the framework is to be used inside a concert room, the idea is that when a person buys a ticket for the given show where the framework will be used, that very same person would download an application and be able to read the necessary information about the seat location from the acquired ticket into the application. That would be the person's location for the whole show to come.

The concert room would then be mapped and divided into different zones. As an example, let's take figure 3.5. Someone who seats on seat 4a would immediately belong to zone 1, but also super zones 5 and 7. This would mean that that smartphone would receive from the network only instructions that were meant for those respective zones. The message distribution is handled at a network level, as was previously described in subsection 3.2.

The mapping and zoning of the concert room allows for the creation of targeted content, like sounds that could sweep across an entire row or column, fade ins and fade outs and all kinds of movement sensations.

So imagining, for example, that a composer would want to play a part of the piece using the audience's smartphones, but creating some sort of sound effect across the room. A composer could send instructions for zone 7 to play a certain sound and then, following that, send instructions for

zone 8 to play a consecutive sound to the first one. This would create a moving effect in the audience.

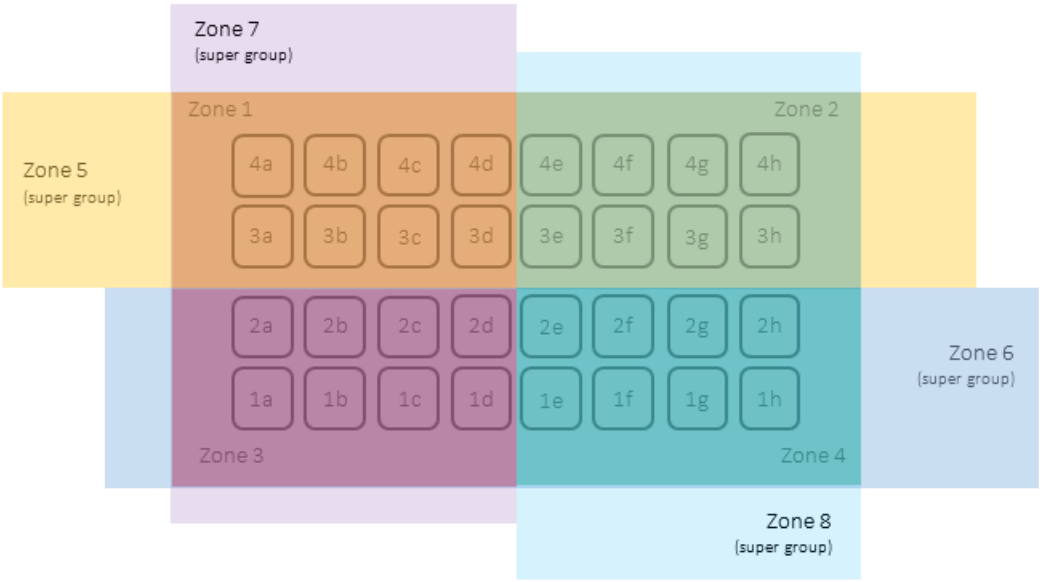


Figure 3.5: Example of zoning in a concert room

Of course this method of location creates some limitations, such as not being able to detect people sitting out of place and possibly creating discrepancies during the planned show, but it is an acceptable solution for the existing problem that can later be developed and worked on to include other methods of location mentioned in this report's State of the Art, chapter 2.

The configuration of the application's attributed place in the concert room is meant to be abstracted from the final user. This is necessary to guarantee the integrity of the system. The more freedom allowed to the users at this level, the more likely it would be that something goes wrong.

# Chapter 4

# Implementation and Final Solution Details

Details about the implementation, decisions made and some low level explanations may be found in this section of the document.

It will start by explaining some decisions about hardware and software usage. Following this, some of the complementary components of the platform and their implementation will be explained. Then it will go on describing some of the network's component decisions and implementations. Posteriorly, it will move on to the PureData patching related areas and finally explain some complementary components implemented and also important for the functioning of the system.

## 4.1 Hardware and Software

There are two different Android versions of the final prototype developed during this dissertation: one for APIs located between 14 and 19, which stands for Android 4.0 and 4.4.4 and another one for APIs higher than that, Android 5.0 and higher.

For now, those are the supported versions of Android. Future changes will have to be taken into consideration, in case there are any major changes in API again.

The ground work laid during this dissertation was therefore developed using the Android environment. Main concerns that surround this kind of environment are the different systems that use it. Different brands build devices with this kind of operating system, which means that phones may be built with completely dissimilar specifications, using different types of hardware for each component. This introduces some unpredictability when it comes to the behaviour of each smartphone individually.

The behaviour is not only dependent of the hardware used by each brand but also of the various versions of the operating system that are available in the market. Brands are able to create their

own specific variations of the original Android versions, besides there being devices that still run rather old versions of the operating system.

During this dissertation, a set of minimum requirements were tested. The minimal Android version started in the 2.4. With improvements made to the platform, a natural need to raise this lower threshold appeared. It was finally set at the 4.0 version of Android, which should be equivalent to API 14.

These settings proved reliable enough to provide a stable solution for the vast majority of Android devices currently in the market.

Although it works in the vast majority of devices, it was detected that in a newer version of the operating system released, Android 5, a new different version of the application was required for it to run in a stable way, due to some significant changes in the network API of the new operating system.

## 4.2 General Android Application Overview

The developed Android application developed consists of the Classes and Activities found in figure 4.1. It intends to show both the class organization and structure, but also somewhat of an application flow. Showing how and when it starts, as well as the reason why.

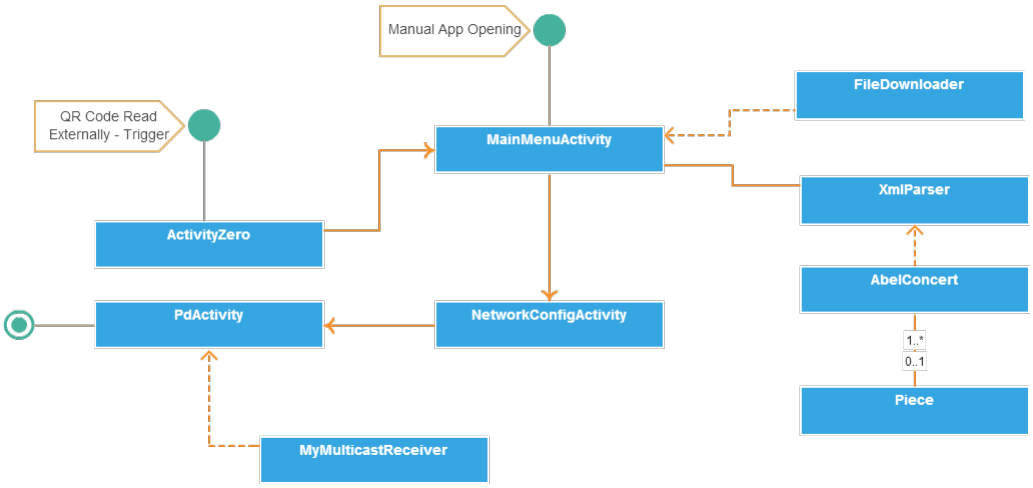


Figure 4.1: Android Application Class Diagram

As may be seen in figure 4.1, there are two possible ways of starting the application. It may be started directly from the reading of a specific QR Code that triggers the application, through **ActivityZero**, which receives the data read from the code and sends it to the right places to start the configuration of the application. It may also be started normally, through **MainMenuActivity**.

This activity exists to represent the pre-concert phase of the application, which consists of various configuration stages. For now it contains options as to what to do, individually, like performing the XML file parsing, **XmlParser**, or downloading the necessary files for the application, **FileDownloader**.

The XML file parsing originates one or more **AbelConcert** objects. This object contains all the information about the concert the application is being configured for and it may contain one or more **Piece** objects. These are correspondent to parts of a concert and contain information needed for the configuration of each piece that requires the use of the application.

The activity **NetworkConfigActivity** is where all the network configurations are applied. It is a transition activity that occurs between the pre-concert phase and the concert phase, leaving the smartphone ready to connect to the local event network.

Finally, the activity that represents the concert phase is **PdActivity**. Here is where all the multimedia content comes to life. The PD components are also embedded in this activity. By running a thread responsible for the message reception, **MyMulticastReceiver**, it is able to get the communication from the server and pass it along to the PD component, for it to be reproduced.

All of the components of this system will be explained below. Starting by the complementary components, as explained in the previous chapter, in section 3.1. Following this, both the Network Component 4.5 and the PureData Component 4.4 will also be explained in more detail.

## 4.3 Complementary Components

Although the main components of this system are the network and the PD patches, there were some developments made in complementary features of the prototype, which will be described in this section, along with the role performed by each of these minor components in the platform's bigger picture.

### 4.3.1 QR Code Reader

With all the dynamic wanted, there was a need for a means of setting up the method of location.

For the time being, a QR Code intent was implemented, which means that when a QR code is read by a QR Code reader application, an event is triggered inside the smartphone, taking the user directly to our application, where data passed inside the QR Code (see figure 4.2) is taken and used to configure the network connection, so it is easily but specifically configured to a specific zone of the concert room.

Figure 4.2: Example of a QR Code

For testing purposes, there is also a manual way of inserting an IP address, but this will not be available in the end user version of the application, where this should be completely abstracted from users, either by the aforementioned QR Code method or, if possibly, an even better solution that uses the previously mentioned Acoustic Location method, which was described in 2.3.3.

The chosen method for location during this dissertation was the attribution of an IP to each seat in a concert room, which would then be mapped into zones that would work as clusters, where certain instructions could be sent. This allows for some instructions to be sent to specific locations inside the concert room.

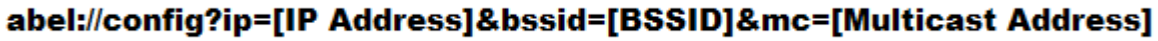


Figure 4.3: Format of QR Codes read into the aplication

This trigger is set up when a QR Code with the format shown in figure 4.3 is read. In place of **[IP Address]** should be inserted the IP address to be attributed to the specific seat where it is placed. The other variable fields will be further explained and discussed in section 4.5.

In a final version of the project the message should be either encrypted, encoded or checksummed to avoid tampering.

When reading the QR Code there are two possibilities:

- The user reads a QR Code with the format mentioned from an external application that is able to read the information contained inside the code and interpret it. This then triggers the application to open and automatically sets up the network components.
- The user opens the application and an option is presented to read a QR Code from inside the application. If this option is selected, the user will be prompted to download and install a specific third party application that is able to read QR Codes and after reading the said QR Code, the result will be retrieved to the application, where it will be processed and automatically set up the network component.

### 4.3.2 XML File Parser and File Downloader

There was a need for decision making on the degrees of freedom that would be conferred for the platform's future usage, given that its aim is to be a framework upon which anyone is able to create a personalized show.

Bearing this in mind, the main concern was making the application the least dependent on specific external files as possible.

To accomplish this objective, it was decided that external files should be able to be downloaded from an external source, such as a web link and preferably, in a dynamic way.

This was done by creating an XML configuration file, in which some of the most important configurations for the application would be passed to it. An XML file parser was created, with some reference tags that were thought to be the most important for the application's self configuration.

In this XML file might also be referred the names of the files necessary for the application's proper functioning, allowing for a complete personalization when it comes to creating a show from scratch with the framework developed.

An example of such a file is presented below and more information may be found in appendix A.

```
<abel_concert>
  <name>Concerto</name>
  <venue>Casa da Musica</venue>
  <location>
    <lat>"41.1621"</lat>
    <lon>"8.6220"</lon>
  </location>
...
  <piece>
    <title>Uma Peca</title>
    <composer>Jose Manuel</composer>
    <callID>UPP</callID>
    <mainPatch>upp.pd</mainPatch>
    <audioFiles>
        <audio>somsonoro.wav</audio>
    </audioFiles>
    <zone>3</zone>
    <totalZones>100</totalZones>
  </piece>
</abel_concert>
```

Listing 4.1: XML Document Example

Files listed should all be downloadable from the same server address, and should differ only in name. If this is true, then the application is ready to dynamically and automatically download all files named in the given XML file. This is another of the complementary components that although not essential, is an important part of this dissertation's final product structure.

Downloaded files are stored in a folder in the device's internal storage and loaded into the application's cache when called upon.

## 4.4 PureData Component

**PureData** is, as explained in chapter 2, a visual programming language. It was chosen for being similar to **MaX** and commonly used among music composers, allowing for an easy adaptation. The existence of **libPD**, a library that allows to easily port PD into any kind of device that is able to run some type of native code, abstracting the hardware input and output specifics, was also a significant part of the decision.

PureData code consists of blocks and connections between them. Each block represents an object and the connections represent data flows between the corresponding objects. PureData files may be placed as a package wrapped inside native code that will abstract the content placed inside it. These files are called **patches**.

It is also possible to extend and create objects in PD by adding external files to the PD libraries. This allows for the creation of specific and personalized objects inside PD, that behave in a personalized way.

This way, a composer who wants to build a concert piece using this framework just needs to develop a central patch, located in the server, and a patch to place inside the smartphones. Messages are then sent from the central patch to the mobile devices' own internal PD patch.

### 4.4.1 Relevant PD Methods

To make the comprehension of the following text easier, some PD methods are now introduced:

- **netsend** - Allows for the sending of messages to a specific address and port.
- **netreceive** - Opposite to the previous method, it allows for the reception of messages directly from the network, from a specific address and port.
- **sendList** - Allows to send PD object a list of objects with a specific type.
- **sendSymbol** - Allows to send PD object a string.

### 4.4.2 Development

Starting with two split pieces, Android and PureData, the objective was to build a compact system capable of supporting and merging both components into a single unit.

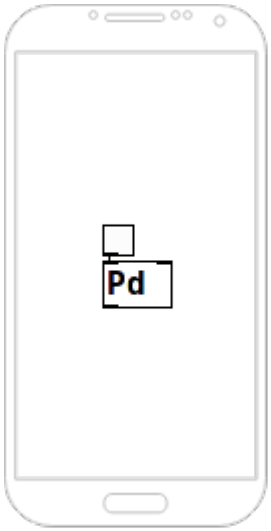


Figure 4.4: Android Running PureData

While allowing for the elasticity of easily including independent PD patching files, the system still must behave in a stable and proper way. This type of problem obviously called for the implementation of a layered system, in which different layers are responsible for different types of tasks.

Abstracting the problem as much as possible, there is a layer of PureData code running inside a layer of Android code. The Android code works as a wrapper for the lower PureData layer and as a visual interface for the application, providing some sort of feedback of what is happening inside it.

The PureData code is responsible for the multimedia content creation and interpretation and the Android code is responsible for providing the PureData code with everything it needs to function properly.

The existence of these different layers inside the application demand for the creation of a communication system between them, since the exchange of messages is essential to ensure that all the components of the application are in the same page and working with the same goal. While there must exist communication, the content of each layer should not be dependable on the each other. The changing of one side should not affect the other. They both should be seen by each other as APIs, meaning that some methods known to each other should be enough to allow the system to completely work while maintaining its independence. The possible replacement of the PD file should go perfectly unnoticed to the Android layer.

The only part of the system that needs to be fully aware of the PureData file contained inside the Android application is the main PureData file, located in the server. These two files need to be totally complete each other, working like two different ends of the same file, but split in physical space. The replacement of one of them, most likely implies the replacement of the other. This means that there is a need for these files to be in constant communication, with messages being sent from the server and interpreted and reproduced in the client.

The implementation of PD communication in Android was done by abstracting the network communication from the PD code and handling it in the native code side, who then passes the important content of the messages received to the PD object, internally.

As seen in figure 4.5, a message is received by the device and after going through some processing, the message's relevant content is sent to the PD code block.

There is also the possibility for the PD object to communicate back to the Android side of the application, which may also be observed in figure 4.5. This was done by implementing a listener in the Android activity that handles the PD object, as explained below, in section 4.4.2.

Two main problems had to be dealt with during the implementation of the system and will be described below: communication between PD patches located in the server and the clients (**1**) and, resulting from the solution found to the first one, communication between Android native code and the embedded PD object (**2**).

1. The need for the Android embedded PD block to communicate with the server side PD block was meant to be handled directly inside the respective blocks. This was supposed to be easily implemented using PD's API methods **netsend** and **netreceive**. These methods are

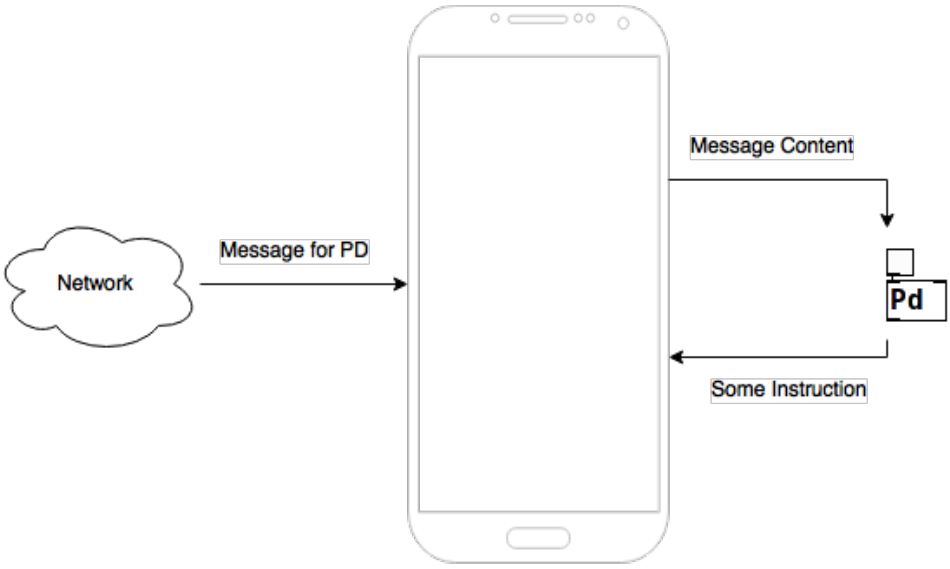


Figure 4.5: Message Exchanging Between Network, Android and PD

implemented inside PD and handle network communication by creating a socket from the sender side to the receiver's known address. It was diagnosed that even though the first one worked fine on the server side, the latter was not able to connect and receive any data directly on the Android side, due to difficulties related to the libPD port of PD. This meant having to handle the network communication in Android native code and internally communicate with the PD object embedded in the application.

With Android handling the network communication, while the messages both originate and are destined to PD objects, they still have to go through other layers of the application in between. On the server side, the message is trimmed to contain only necessary information for the client. On the client side, message content is read and carefully handled before being sent to the PD object present.

The PD object embedded in Android was programmed to receive orders from the server, a pair of values at a time, that would correspond to **Target** and respective **Value**. The Target of the message is the PD object to which it is destined and the Value is the instruction sent for that specific Target. This is handled inside the PD code block, after the reception of every pair.

2. The message received from the server side PD object was of variable size and may contain more than a single of these pairs, **Target Value**. The message is, as mentioned in the previous item, received in the Android side and, after being processed, each of these pairs is sent at a time to the PD object using a method from the PD API described above, **sendList**.

   Due to the necessity of using Android as an intermediary in the network communication between PD objects and the type of characters used to convey the messages, there were some difficulties tracking down errors of interpretation of ASCII characters. These characters

were sent as identifiers from the PD server side. While handling the messages in the Android native code, conversions were inevitable. These conversions damaged the data content and, when sent to the PD client object, would make it crash with little to no warning messages or explanation.

The problem was solved by switching the method used to communicate with PD. Since **sendSymbol** was not effective, because of its input argument type, **sendList** was used. This way, the Android native code converts the data received to integer values and only then does it proceed to send it into the PD object.

Communication from the PD object to the native side of the client's application is also possible.

To handle this type of communication, a listener was implemented. This listener is triggered by a PD object call and, after checking the PD object that triggered the call, it decides on how to act. For now, the only possible action taken is to change the background color of the screen. So when a message is received, it should contain an RGB code that is used to change the color displayed.

Implementing this component was probably the most challenging part of the project, having to deal with PD and its patches and components.

Being a community developed library, libPD presents some limitations. These limitations create problems when it comes to locating errors that occur inside the PD code.

Dealing with those errors was tricky and confusing, due to the difficulty of debugging the embedded PD object. The codes sent out are not quite specific which led to many hours of trying to understand what was going wrong inside PD code, during parts of the development. Debugging and fixing of this type of errors was always done between the Android part of the team and the Multimedia part of the team, which includes the PD experts.

This implied some complex debugging, including inside the PD API itself, to find out the origin of the errors and how to get around them.

More details on the implementation of this system component along with some code excerpts may be found in appendix B.

## 4.5 Network Component

As previously mentioned in this report, communication between a central server and the client's Android applications is essential for the functioning of the system developed.

The method of communication chosen between server and clients was Multicast over UDP. This decision was made based on the fact that feedback on packet reception is not needed for the system to properly work. Multicast allows for a group of users located inside the same zone to all simultaneously connect to the same address, subscribing to all the messages that are broadcast over that channel.

The application connects the device to a specific network and AP, given a BSSID [1] that is

[1] The BSSID is an individual identifier unique to each network distributor device and it is possible to program an Android device to choose a specific AP to which it should connect using this identifier.

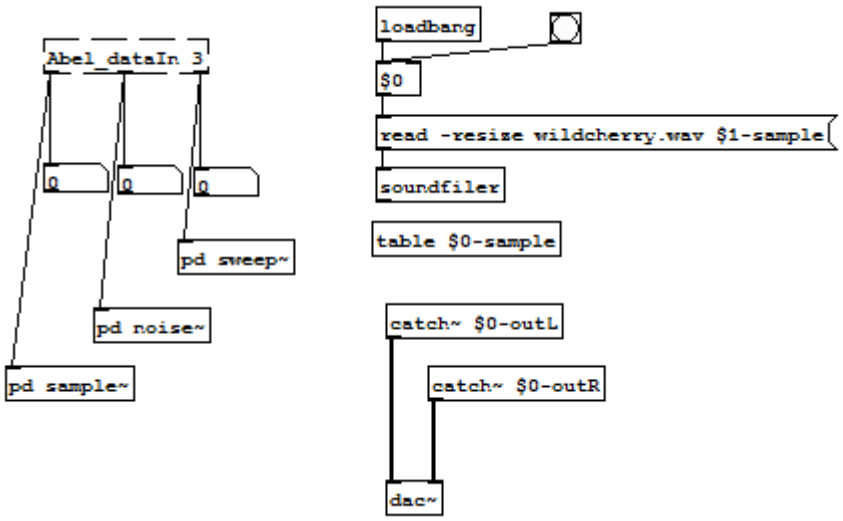


Figure 4.6: PD Code From the Android Side of the Application

passed to the application through the QR Code. It also sets a specific static IP which allows per device identification and localization inside the concert room.

### 4.5.1 Decisions

Even though Multicast over UDP does not guarantee packet delivery or order in packet reception, it provides a way smaller overhead when compared with TCP, due to the fact that it does not have to ensure packet reception. This makes it a better protocol for applications that deal with real time multimedia content, such as this one, since it is better for a packet to be completely lost than for it to arrive a lot later than the others and propagate the timing delay throughout the rest of the communication.

The need for a Multicast type of communication channel also appeals for a UDP type of connection, since ensuring the delivery of every single packet to multiple different targets connected to a single Multicast channel would eventually represent a significant cost in time of arrival of the packets.

### 4.5.2 Network Testing Application

A small secondary application was built to help in the network testing and validation. It was developed as an Android application that would allow to set a static IP configured to the local testing network. Furthermore, it allowed a connection to four simultaneous multicast channels and would provide a total of packets received in each channel, with the possibility of emailing the results to a specific email created for this purpose only, in a CSV file.

### 4.5.3 IP and BSSID set up

As mentioned before in this report, a decision was made to map a concert room into IP addresses, so it would always be known, which IPs were inside which zones, allowing for the definition of certain zones where content could be sent in an isolated way. This meant that the IP configuration must come from the device and not the network AP, as usually it is done.

The way found for the attribution of a static IP address to each device was previously discussed in section 4.3.1. After being read from the QR Code, the IP address is then passed inside the application, to a network configuration activity, which configures the Android device's network interface to connect to the network with that specific IP.

Besides passing the IP to the inner layers of the application, the QR Code also encodes the network's BSSID and the address of the Multicast channel to which the user should connect. This allows for a completely dynamic way of setting up the entire network interface inside the Android device.

It is, as mentioned above, passed with the QR Code and forces the smartphone to not only connect to the specific network where the data for the concert will be broadcast, but also to connect to a specific AP inside that very same network. This is useful in the sense that, since the network layer of the system is responsible for the distribution of the messages across the different zones mapped in the concert room, it needs to be guaranteed that a device that has a specific IP address attributed to itself will be connected to a specific AP inside the concert room, whether it is the one it would automatically connect to or not.

Code relative to the implementation of the Multicast communication and newtork configuration may be found in appendix C.

### 4.5.4 Zones and Messages Exchanged

A zone is therefore message fed by one AP, but this does not mean that one AP feeds solely that one zone. This means that one AP may be broadcasting into more than one Multicast channel. This is why there is also a need to pass a Multicast address through the QR Code read into the application. It is the final component which ensures that a smartphone will be receiving the correct messages sent to the zone corresponding to the user's seat inside the concert room.

After making sure the device is connected to the right network and AP, a Multicast UDP socket is then created in the address given.

**/target [list] /msg [[DestMsg]*]**

Figure 4.7: Message received in the Android device from the network

The message observable in figure 4.7 is a template of a normal message received in the Android devices, from the PD object located in the server. Such a message would be, for example:

- /target 0 1 2 /msg #!

The message is divided in two parts, being the first part destined to the server wrapper around the PD object, so it knows where to send each message going into the network, and the second part meant for the PD object located inside the Android client code. The Android limits itself to isolating the part of the message coming after the **"/msg"** tag, sending it directly into the PD object, which is responsible for interpreting and reproducing the resultant consequences.

So in the specific example that was presented, it would mean that the message was headed for zones 0, 1 and 2 in the audience, and the content of the message to be sent to the PD object would be "**#!**". This last part of the message is always composed by pairs of ASCII characters. The first one represents a target object inside the PD code and the second one, the message for that target. The interpretation is made inside the PD object.

## 4.6 iOS

A preliminary prototype for iOS was started during the time of this dissertation's elaboration, since it would be ideal that in a final state, the project would be cross-platform, as to reach the most possible owners of smartphones across different brands and operating systems.

A project was created that shows PureData is embeddable in such a system, being that it was integrated in the small project created. This project basically consists of a small block of PureData inserted in an iOS program, that reproduces a constant sound. As small as it is, it is enough to prove that integration is possible and with the right amount of time and effort, these kind of smartphones should be able to perform the exact same things that were already implemented in Android devices, so far.

# Chapter 5

# Tests and Results

In this chapter, tests made with different devices and versions of the Android OS will be presented. Another test, representing a very small simulation of the system performing in a real environment was performed and is described later.

## 5.1 Individual Device Test

A variety of devices was tested individually to measure performance and identify individual failure points when running the application under different types of hardware and software. In every test, every single component developed and present in the final prototype was tested:

- QR Code reading, interpreting and consequent application triggering
- XML file parsing
- File Downloading
- Correct network configuration
- Correct PD connection and usability

### 5.1.1 Individual Device Testing

Here are presented the results of tests performed to individual devices, in an isolated way, as a means to test performance variations across different brands, models and Android OS versions.

Test conditions were as follows:

1. Install application APK.
2. Guarantee there is an active internet connection

3. Read QR Code
4. Read XML File
5. Download Files
6. Connect to testing network and apply correct configurations
7. Run testing PD activity

Steps from 4 to 7 are implicitly executed inside the application.

All tests performed were successful across all the different devices tested. Below is presented a list with all the devices and corresponding Android versions.

| Brand | Model | Android OS |
|---|---|---|
| Samsung | Galaxy S III LTE | 4.4.4 |
| HTM | Unknown | 4.2 |
| Samsung | Galaxy Tab 10.1 LTE | 4.4.2 |
| Samsung | Galaxy S IV Mini | 4.2.2 |
| Ainol Novo | 7 Venus | 4.1.1 |
| Jiayu | G3s | 4.2.1 |
| OnePlus | One | 5.0.1 |
| Lazer | Capacitive 10 | 4.0.3 |
| Motorola | Moto G (2nd Gen) | 5.0.2 |
| Asus Google | Nexus 7 | 5.1.1 |
| Samsung | Galaxy S III | 5.1.1 |

Table 5.1: Devices Individually Tested

### 5.1.2 Small Scale System Simulation

A small scale of the whole system was made, using 8 Android devices.

The test was performed at INESC-TEC, during a group meeting, using participant's available Android smartphones.

In total, 8 devices (mid to high range) were made available for testing: 1 One plus One smartphone, 1 Huawei smartphone, 1 Nexus tablet and 3 Samsung smartphones and 2 unbranded smartphones.

The application was installed in all of the devices and some problems immediatly came up. A problem was detected in most devices, when trying to programatically configure the network connection to the local network of the system.

After some experimenting with the mentioned devices it was discovered that it was due to the existence of an active mobile network data connection. This had never been tested before, which is admittedly a testing failure, when it came to testing individual devices. Apparently, when the data connection is on, the Android OS is not able to set a static IP address in a programatic way to connect to a network configured with no DHCP.

After discovering the origin of the errors and turning off the data connection of the devices, the configuration went down at a normal pace and the devices were ready for the system testing.

It should be mentioned that a correction to this problem is easily available, since it is also possible to disable Android mobile network data connections programatically. The only reason this was not done before was a failure in predicting and testing such a situation.

One of the devices, a Nexus 5, was unable to properly function because it was equipped with a ROM that did not support IGMPv4, which is essential for Multicast communication.

Another problem detected was that one, and only one, of the QR code reading applications, would not trigger the default action of opening the application and would try to open the smartphone browser.

The last problem found was that one of the devices simply stopped receiving any packets from the Multicast channel it was connected to. This had happened before during the implementation. It was found to be due to battery life saving specifications applied differently by different brands and models. It had been solved by keeping the screen always on, while the application was running, but apparently that solution is not universal, since it did not work with this specific device.

As a whole system, the test showed that there are some significant limitations to be aware of when applying this system to a real life situation, due to the very broad range of Android devices and specifications out there. The latency was very varied from device to device, which makes it hard to find a standard, even though all the devices run the same OS, just different versions of it.

It was easily observable that contrary to expected, message reception was considerably worse than that observed on the single-device tests.

OnePlus One and Nexus 7 showed the best performances, with good consistency overall and some ocasional message loss, but within the acceptable range.

Some of the weaker devices showed message loss above 50%.

On the devices that performed in an acceptable way, there were some visual glitches happening in the interface that were attributed to the way the parsing of the messages on the Android side of the application is done. It is believed that with some minor improvements in terms of efficiency to the code implemented, this will become less significant.

The test showed that the implemented features are working, but some parts of it need to be worked on and improved in a way that enhances performance.

More detailed results of that part of the testing are not found in this dissertation, for they are specific to other team's areas, which do not concern this document.

# Chapter 6

# Conclusions and Future Work

After explaining and describing the idea behind this dissertation, the concept that originated the project developed, the full scope of the framework meant to be developed and the work done so far, the time comes to take some conclusions from what was already done and what still needs to be done.

First of all, the work described during this dissertation is only a part of the entire project, as it was initially explained. There are different components involved in making all of the final system work and more people working in parallel to make those components work. This dissertation is mostly focused on the Android development part of the project, although some parts may be found where other areas were discussed and presented, since context is essential when talking about the application developed.

It should also be mentioned that the application developed during this time is not a final version of the application that should be used in concerts. It should be seen as a functional prototype.

The work developed so far proves, that although the idea is achievable in a real world environment, it will have far more limitations than would be desired for such a system. Android has the huge problem of being installed in completely different devices and hardware, besides having multiple different versions, which makes it all that much harder to find a common base to work with and setting standards over what each device is or is not capable of. This forces us to completely exclude a low minority of devices, to eliminate some chaos, but it still does not guarantee full functionality and much less synchronism between the vast range of Android devices available in the market, which is one of the most important things when it comes the functioning of this system.

Even though these limitations exist, they can still be worked on and other solutions may be found that minimize the mentioned problems, it just probably will never be reduced to the ideal zero.

Some of the components developed are ready to be used in a final version of the framework and some still need some development, but the work done is considered to be a good ground base

for future development.

This is where it makes sense to start talking about future work. First of all, it should be mentioned that, in order to reach a higher spectrum of people present in every audience, no matter what show, the framework needs to be developed for the iOS environment as well. This should be seen as priority work, from now on, since it represents a big part of the smartphone selling shares worldwide.

Implementing the system in the iOS environment will bring new challenges and limitations, since the OS developed by the Apple company does not allow for the same programatic liberties that the Android does, being an open source operative system. This means that setting a static IP and connecting to a specific Wi-Fi network directly from the code would definitely be a problem that would need solving.

About what was already done and still needs to be improved, the focus should definitely be in finding better ways to synchronize event triggering in the Android side of the application, since that seems to be the hardest part to fix. Proposals such as defining a zero time, triggered by a network message, followed by some synchronizing points during the concert have been discussed by the team and should be tested. This would imply embedding a time stamp in the messages sent from the server to the clients, which would represent the time relatively to the moment zero in which that instruction would be reproduced, instead of just reproducing it as soon as it arrives. This solution would probably mean less chaos, since the Android system should be able to maintain a clock updated similarly, even if it means adding some synchronizing messages during the concert.

On the other hand, a system as integral as iOS, completely specific for the type of devices that run it, with everything built into it, should perform significantly better when it comes to synchronization, if compared to the Android devices. Different platforms, mean different problems, but in this case, this would come as a big advantage, since it is such an important situation.

Localization methods could also be improved. For the time being, the solution found is acceptable and ignoring some minor incidents, such as place switching, it should be functional, but having a localization method implemented, such as one of the methods described in the State of the Art chapter 2 of this report, would add a lot of value to this project. This would mean that changes in position would be noticeable and relocation of the smartphone to another zone of the concert room would be done easily, so it would be sent messages meant for its real location, updated location, instead of messages meant for its original pre-configured location. Even if just used as a complementary method to the already existing one, it would be an important part of the project.

Messages received from the server could also be improved if better structured, making it easier for the devices to know where is the information needed and how to get it directly. This would improve processing speed when it comes to read values off those messages and sending them to PD.

Finally, the implementation of some smartphone sensors readings, such as the accelerometer and the proximity sensor, would help creating a bigger sense of interactivity in the audience. Values read from these sensors could influence the output of the smartphone in some way and

make it seem like each person would play a bigger role in the show. This was already started in the PD part of the project, as it is ready to receive values from the mentioned sensors, but it was not implemented in the final testing version of the prototype.

# References


[APP] Audio plugin player. http://defectiverecords.com/audiopluginplayer/index2.html. Accessed: 2015-01-29.

[Bal13] Kannan Balajee. Predictive indoor navigation using commercial smart-phones. *SAC '13: Proceedings of the 28th Annual ACM Symposium on Applied Computing*, 2013.

[BMSM10] Ivica Ico Bukvic, Thomas Martin, Eric Standley, and Michael Matthews. Introducing l2ork: Linux laptop orchestra. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 170–173, 2010.

[BP] Batphone. http://stevetarzia.com/batphone/. Accessed: 2015-01-29.

[COB] Canofbeats. http://mccormick.cx/projects/CanOfBeats/. Accessed: 2015-01-29.

[DCA+07] Roger B Dannenberg, Sofia Cavaco, Eugene Ang, Igor Avramovic, Barkin Aygun, Jinwook Baek, Erik Barndollar, Daniel Duterte, Jeffrey Grafton, Robert Hunter, et al. The carnegie mellon laptop orchestra. *Research Showcase @ CMU*, 2007.

[FCC10] V. Filonenko, C. Cullen, and J. Carswell. Investigating ultrasonic positioning on mobile phones. *Indoor Positioning and Indoor Navigation (IPIN), 2010 International Conference on*, pages 1–8, 2010.

[GAL] Garageacidlab. http://mccormick.cx/projects/GarageAcidLab/. Accessed: 2015-01-29.

[HABF08] Alex Harker, Angie Atmadjaja, Jethro Bagust, and Ambrose Field. The worldscape laptop orchestra: Creating live, interactive digital music for an ensemble of fifty performers. In *Proceedings of the International Computer Music Conference Belfast, UK*, 2008.

[Her06] Marion Hermersdorf. Indoor positioning with a wlan access point list on a mobile device. In *Proc. Workshop on World-Sensor-Web*, volume 5. Citeseer, 2006.

[Kai13] Liu Kaikai. Guoguo: enabling fine-grained indoor localization via smartphone. *MobiSys '13: Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, 2013.

[lib] libpd. http://libpd.cc/. Accessed: 2015-01-29.

[Mir13] Rossi Mirco. Roomsense: an indoor positioning system for smartphones using active sound probing. *AH '13: Proceedings of the 4th Augmented Human International Conference*, 2013.

## REFERENCES


[MSLK14] Alex T Mariakakis, Souvik Sen, Jeongkeun Lee, and Kyu-Han Kim. Sail: Single access point-based indoor localization. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 315–328. ACM, 2014.

[PD1] Puredata. http://puredata.info/. Accessed: 2015-01-29.

[pMi] pmix. https://cycling74.com/project/project1/. Accessed: 2015-01-29.

[RjD] Rjdj. http://www.rjdj.me/. Accessed: 2015-01-29.

[SCM] Alberto Serra, Davide Carboni, and Valentina Marotto. Indoor pedestrian navigation system using a modern smartphone. In *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services*, pages 397–398. ACM.

[SLKC] Souvik Sen, Jeongkeun Lee, Kyu-Han Kim, and Paul Congdon. Avoiding multipath to revive inbuilding wifi localization. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 249–262. ACM.

[TCSW06] Daniel Trueman, Perry Cook, Scott Smallwood, and Ge Wang. Plork: the princeton laptop orchestra, year 1. In *Proceedings of the international computer music conference*, pages 443–450, 2006.

[TNT] Tapntempo. http://jnote.org/2882. Accessed: 2015-01-29.

[WBOH09] Ge Wang, Nicholas Bryan, Jieun Oh, and Rob Hamilton. *Stanford laptop orchestra (slork)*. Ann Arbor, MI: MPublishing, University of Michigan Library, 2009.

[WEP08] Ge Wang, Georg Essl, and Henri Penttinen. Do mobile phones dream of electric orchestras. In *Proceedings of the International Computer Music Conference*, 2008.

[XJ13] Jie Xiong and Kyle Jamieson. Arraytrack: A fine-grained indoor location system. In *NSDI*, pages 71–84, 2013.

[YA05] Moustafa Youssef and Ashok Agrawala. The horus wlan location determination system. In *Proceedings of the 3rd international conference on Mobile systems, applications, and services*, pages 205–218. ACM, 2005.

# Appendix A

# Appendix 1: XML Configuration Files

## A.1 XML File Example

```
<abel_concert>
  <name>INESC 30 anos</name>
  <venue>Casa da Musica</venue>
  <location>
    <lat>"41.1621"</lat>
    <lon>"8.6220"</lon>
  </location>
  <time>2015-10-26T21:30:00.00</time>
  <seat>H27</seat>
  <ipAddress>204.11.3.4</ipAddress>
  <subnetMask>255.255.255.255</subnetMask>
  <piece>
    <title>pendulum</title>
    <composer>Rui Penha</composer>
    <callID>PNDL</callID>[a]
    <mainPatch>pendulum.pd</mainPatch>
    <otherPatches>[b]
      <patch>addpendulum1.pd</patch>
    </otherPatches>
    <audioFiles>
      <audio>pendulum.wav</audio>
    </audioFiles>
    <qlistFiles>
      <list>eventos1.txt</list>
      <list>eventos2.txt</list>
    </qlistFiles>
    <zone>3</zone>
    <totalZones>6</totalZones>[c]
  </piece>
  <piece>
    <title>Ai o caraaaaa</title>
```

```
32      <composer>Carlos Guedes</composer>
33      <callID>CRAA</callID>
34      <mainPatch>craa.pd</mainPatch>
35      <audioFiles>
36          <audio>freeze.wav</audio>
37      </audioFiles>
38      <zone>3</zone>
39      <totalZones>100</totalZones>
40    </piece>
41  </abel_concert>
```

Listing A.1: XML Document Example

## A.2 Tags Implemented

- **abel_concert** - Marks the beginning and ending of the XML file with configurations for the application.
- **name** - Name of the concert
- **venue** - Venue where the concert will happen
- **location** - Introduces the specification of location coordinates
- **lat** - Latitude
- **lon** - Longitude
- **seat** - Seat associated with the device
- **ipAddress** - IP Address associated with the seat
- **subnetMask** - Other network configuration value
- **piece** - Introduces a piece inside the concert with all the information and files necessary
- **title** - Title of the piece which contains it
- **composer** - Composer of the piece
- **callID** - ID that identifies the piece
- **mainPatch** - Main patch file to be used with the piece
- **otherPatches** - Other complementary patch files that may be necessary
- **patch** - Introduces the name of a file
- **audioFiles** - Complementary audio files necessary for the piece

- **qlistFiles** - Complementary files that may be necessary
- **zone** - Zone in which the device is located
- **totalZones** - Total amount of zones

Not all these tags must be present in every file, being that only the **abel_concert** tag is mandatory to be present as opening and closing tag, for the time being.

Other tags may still be implemented if necessary with easiness.

Files to be downloaded are read from this XML file. Their names are taken from it and then a download is tried from a fix server address with the given file name.

Appendix 1: XML Configuration Files

# Appendix B

# Appendix 2: PureData Implementation Details

All the code related to the connection between Android and the PureData code is located inside the file called **PdActivity.java**. This file is an Android Activity.

It has a method called **onCreate** that is run whenever the Activity is created, initializing all the data necessary for the functioning on the Activity.

```
1  protected void onCreate(android.os.Bundle savedInstanceState) {
2    super.onCreate(savedInstanceState);
3    AudioParameters.init(this);
4    PdPreferences.initPreferences(getApplicationContext());
5    PreferenceManager.getDefaultSharedPreferences(getApplicationContext()).
         registerOnSharedPreferenceChangeListener(this);
6    initGui();
7    bindService(new Intent(this, PdService.class), pdConnection, BIND_AUTO_CREATE);
8
9    Bundle extras = getIntent().getExtras();
10       if(extras != null)
11       {
12         mcChannel = extras.getString("MC");
13       }
14 };
```

Inside the **onCreate** method are observable calls to many functions related to the initialization of PureData.

```
1 AudioParameters.init(this);
```

This method allows for the configuration of parameters that enable low-latency features. It basically gets the Android OS version running in the device.

```
PdPreferences.initPreferences(getApplicationContext());
```

This method initializes preferences either set by the previous method or default ones.

```
bindService(new Intent(this, PdService.class), pdConnection, BIND_AUTO_CREATE);
```

This binds the Android to the PD side, by creating a **pdConnection**, method shown below.

```
private final ServiceConnection pdConnection = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
      pdService = ((PdService.PdBinder)service).getService();
      initPd();
    }

    @Override
    public void onServiceDisconnected(ComponentName name) {
      // this method will never be called
    }
  };
```

Here, a pdConnection is created. It creates a service binder to connect to PD and calls **initPd**, which handles all the remaining necessary configurations.

```
private void initPd() {
    Resources res = getResources();
    File patchFile = null;
    try {
      PdBase.setReceiver(receiver);
      PdBase.subscribe("android");
      PdBase.setReceiver(dispatcher);
      dispatcher.addListener("ab_rgb", pdListener);

      InputStream in = openFileInput("test.pd");
      patchFile = IoUtils.extractResource(in, "test.pd", getCacheDir());
      IoUtils.extractResource(res.openRawResource(R.raw.wildcherry), "wildcherry.
          wav", getCacheDir());
      PdBase.openPatch(patchFile);
    } catch (IOException e) {
      Log.e(TAG, e.toString());
      finish();
    } finally {
      if (patchFile != null) patchFile.delete();
    }
  }
```

In this code snippet, **initPd** is shown. Here is well the magic happens. PdBase is a project that comes with the lipPD for Android package. It is the object that bridges Android and PD and allows for messages to be sent either way.

Files are explicitly open inside this method as well and loaded into cache memory to be used during the execution of PD code.

After the execution of all these configuration methods, the PD service is started by clicking a button that triggers a call to the following method:

```
1  private void startAudio() {
2      String name = getResources().getString(R.string.app_name);
3      try {
4        pdService.initAudio(-1, -1, -1, -1);   // negative values will be replaced
             with defaults/preferences
5        pdService.startAudio(new Intent(this, PdActivity.class), R.drawable.icon,
             name, "Return to " + name + ".");
6      } catch (IOException e) {
7        toast(e.toString());
8      }
9    }
```

This method starts the audio component of PD, which is the main focus of the PD side of the application, playing audio content.

Finally, the method that is triggered whenever a new message incomes from the network. This method is called asynchronously from the thread that handles network communication.

```
1  Runnable updateTextMessage = new Runnable() {
2        public void run() {
3
4          String dest = "ab_dataMsg";
5
6          if(myMultiCastReceiver == null) return;
7
8          DatagramPacket packet = myMultiCastReceiver.lastPacket;
9
10         byte[] buffer = new byte[140];
11
12         String newMessage = new String(buffer, 0, packet.getLength());
13
14         for(int i=15; i<newMessage.length()-2; i=i+2){
15           PdBase.sendList(dest, ((int) (buffer[i] & 0xFF)) - 127, ((int) (buffer[i
                 +1] & 0xFF)) - 127);
16         }
17       }
18   };
```

The essential part is the last instruction:

```
PdBase.sendList(dest, ((int) (buffer[i] & 0xFF)) - 127, ((int) (buffer[i+1] & 0xFF)
    ) - 127);
```

As mentioned in section 4.4, the method **sendList** is used to send pairs of Target Value instructions to the PD code running. **dest** stands for the object inside PD that expects these pairs of values.

To handle calls from inside the PD code to the Android side of the application, a listener was implemented as follows:

```
private final PdListener pdListener = new PdListener.Adapter() {
    @Override
    public void receiveSymbol(String source, String symbol) {
      System.out.println(symbol);
      if(source.equals("ab_rgb"))
        changeColor("#" + symbol);
    }
  };
```

This method is very simply built, it only checks the origin of the message and if it corresponds to the PD object **ab_rgb**, then it is handled as a message to change the screen colour.

# Appendix C

# Appendix 3: Network Component Implementation Details

## C.1 Multicast Code

All the code relative to the implementation of the thread responsible for handling the multicast communication is found below.

```
1  class MyMulticastReceiver extends Thread {
2
3    final int MAX_UDP_DATAGRAM_LEN=1000;
4    final int MC_PORT1=2500;
5    final int MC_PORT2=2501;
6    final int MC_PORT3=2502;
7    final int MC_PORT4=2503;
8    final String ADDRESS1 = "224.0.0.5";
9    final String ADDRESS2 = "224.0.0.6";
10   final String ADDRESS3 = "224.0.0.7";
11   final String ADDRESS4 = "224.0.0.8";
12
13     private boolean bKeepRunning = true;
14     private String lastMessage = "";
15     private byte[] lastBuffer = new byte[140];
16     Runnable uiHandler;
17   PdTest mainUI;
18   MulticastSocket socket;
19   InetAddress address;
20   int mcChannel = 0;
21
22   public MyMulticastReceiver(PdTest pdTest) {
23     super();
24     mainUI = pdTest;
25   }
26
```

Appendix 3: Network Component Implementation Details

```
27  public void run()
28  {
29
30     WifiManager wifi = (WifiManager) mainUI.getSystemService(Context.WIFI_SERVICE)
           ;
31       MulticastLock mLock = wifi.createMulticastLock("mylock");
32       mLock.acquire();
33
34      String message = "";
35        byte[] buffer = new byte[140];
36
37            while (bKeepRunning)
38            {
39              if(mcChannel != 0)
40              {
41                    try {
42                      DatagramPacket packet = null;
43                      switch(mcChannel)
44                      {
45                        case 1:
46                          packet = new DatagramPacket(buffer, buffer.length,
                                address, MC_PORT1);
47                          break;
48                        case 2:
49                          packet = new DatagramPacket(buffer, buffer.length,
                                address, MC_PORT2);
50                          break;
51                        case 3:
52                          packet = new DatagramPacket(buffer, buffer.length,
                                address, MC_PORT3);
53                          break;
54                        case 4:
55                          packet = new DatagramPacket(buffer, buffer.length,
                                address, MC_PORT4);
56                          break;
57                        default: break;
58                      }
59                      socket.receive(packet);
60                      message = new String(buffer, 0, packet.getLength());
61                        lastMessage = message;
62                        lastBuffer = buffer;
63                        mainUI.runOnUiThread(mainUI.updateTextMessage);
64
65                    } catch (UnknownHostException ue) {
66                    } catch(IOException ez){
67                    }
68              }
69            }
70  }
```

```
71
72  public void kill() {
73        bKeepRunning = false;
74    }
75
76    public String getLastMessage() {
77        return lastMessage;
78    }
79
80    public byte[] getLastBuffer(){
81      return lastBuffer;
82    }
83
84    //SWITCH BEWTEEN MC CHANNELS ON THE RUN
85
86    public void switchChannels(int channel){
87      switch(channel){
88      case 1:
89        try {
90        multiCastChannelSwitcher(MC_PORT1, ADDRESS1);
91        mcChannel = 1;
92      } catch (IOException e) {
93        e.printStackTrace();
94      }
95        break;
96      case 2:
97        try {
98        multiCastChannelSwitcher(MC_PORT2, ADDRESS2);
99        mcChannel = 2;
100     } catch (IOException e) {
101       e.printStackTrace();
102     }
103       break;
104     case 3:
105       try {
106       multiCastChannelSwitcher(MC_PORT3, ADDRESS3);
107       mcChannel = 3;
108     } catch (IOException e) {
109       e.printStackTrace();
110     }
111       break;
112     case 4:
113       try {
114       multiCastChannelSwitcher(MC_PORT4, ADDRESS4);
115       mcChannel = 4;
116     } catch (IOException e) {
117       e.printStackTrace();
118     }
119       break;
```

```
120       default:
121         break;
122       }
123     }
124
125   private void multiCastChannelSwitcher(int port, String add) throws IOException,
126       UnknownHostException, SocketException {
127     socket = new MulticastSocket(port);
128     address = InetAddress.getByName(add);
129     socket.joinGroup(address);
130     socket.setSoTimeout(2000);
131   }
132 }
```

## C.2 Network Configuration Code for Android Versions Lower than 5.0

All the code relative to the configuration of IP Address and connection to network with specific name and BSSID as described in section 4.5, for Android versions below 5.0.

```
1  public class IPConfigActivity extends Activity {
2
3    private final static String DEFAULT_GATEWAY = "192.168.1.1";
4    private final static String DEFAULT_DNS = "127.0.0.1";
5    private final static String NETWORK_NAME = "AbelNetwork";
6    private final static String BSSID = "C4:6E:1F:BD:48:94";
7
8    EditText seatNumber;
9      Button btn;
10
11   @Override
12   protected void onCreate(Bundle savedInstanceState) {
13     super.onCreate(savedInstanceState);
14     setContentView(R.layout.activity_ipconfig);
15
16     seatNumber = (EditText) findViewById(R.id.seatNumber);
17         btn = (Button) findViewById(R.id.InsertBtn);
18
19         Bundle extras = getIntent().getExtras();
20         if(extras != null)
21         {
22           connectAbelNetwork();
23           String ip = extras.getString("IP_ADDRESS");
24           setWiFiParameters(ip);
25
26           Intent intent = new Intent(this, PdTest.class);
```

```
27          startActivity(intent);
28          finish();
29        }
30
31  }
32
33  private void connectAbelNetwork() {
34
35    WifiManager wifiManager = (WifiManager)getSystemService(Context.WIFI_SERVICE);
36
37    WifiInfo currConf = wifiManager.getConnectionInfo();
38
39    WifiConfiguration wifiConf = new WifiConfiguration();
40    wifiConf.SSID = "\"" + NETWORK_NAME + "\"";
41    wifiConf.allowedKeyManagement.set(WifiConfiguration.KeyMgmt.NONE);
42    wifiConf.BSSID = BSSID;
43
44    if(!currConf.getBSSID().equals(wifiConf.BSSID))
45    {
46      wifiManager.addNetwork(wifiConf);
47
48      List<WifiConfiguration> list = wifiManager.getConfiguredNetworks();
49      for( WifiConfiguration i : list ) {
50          if(i.SSID != null && i.SSID.equals("\"" + NETWORK_NAME + "\"")) {
51               wifiManager.disconnect();
52               wifiManager.enableNetwork(i.networkId, true);
53               wifiManager.reconnect();
54               break;
55          }
56       }
57    }
58  }
59
60  @Override
61  public boolean onCreateOptionsMenu(Menu menu) {
62    // Inflate the menu; this adds items to the action bar if it is present.
63    getMenuInflater().inflate(R.menu.ipconfig, menu);
64    return true;
65  }
66
67  @Override
68  public boolean onOptionsItemSelected(MenuItem item) {
69    // Handle action bar item clicks here. The action bar will
70    // automatically handle clicks on the Home/Up button, so long
71    // as you specify a parent activity in AndroidManifest.xml.
72    int id = item.getItemId();
73    if (id == R.id.action_settings) {
74      return true;
75    }
```

```
76      return super.onOptionsItemSelected(item);
77    }
78
79    public static void setIpAssignment(String assign, WifiConfiguration wifiConf)
80              throws SecurityException, IllegalArgumentException,
                    NoSuchFieldException, IllegalAccessException,
                    InvocationTargetException, NoSuchMethodException {
81              setEnumField(wifiConf, assign, "ipAssignment");
82      //    }
83      }
84
85      public static void setIpAddress(InetAddress addr, int prefixLength,
            WifiConfiguration wifiConf)
86              throws SecurityException, IllegalArgumentException,
                    NoSuchFieldException, IllegalAccessException,
87              NoSuchMethodException, ClassNotFoundException, InstantiationException,
                    InvocationTargetException
88      {
89          Object linkProperties = getField(wifiConf, "linkProperties");
90          if(linkProperties == null)return;
91          Class laClass = Class.forName("android.net.LinkAddress");
92          Constructor laConstructor = laClass.getConstructor(new Class[]{InetAddress.
                class, int.class});
93          Object linkAddress = laConstructor.newInstance(addr, prefixLength);
94
95          ArrayList mLinkAddresses = (ArrayList)getDeclaredField(linkProperties, "
                mLinkAddresses");
96          mLinkAddresses.clear();
97          mLinkAddresses.add(linkAddress);
98      }
99
100     public static void setGateway(InetAddress gateway, WifiConfiguration wifiConf)
101             throws SecurityException, IllegalArgumentException,
                    NoSuchFieldException, IllegalAccessException,
102             ClassNotFoundException, NoSuchMethodException, InstantiationException,
                    InvocationTargetException{
103         Object linkProperties = getField(wifiConf, "linkProperties");
104         if(linkProperties == null)return;
105         Class routeInfoClass = Class.forName("android.net.RouteInfo");
106         Constructor routeInfoConstructor = routeInfoClass.getConstructor(new Class
                []{InetAddress.class});
107         Object routeInfo = routeInfoConstructor.newInstance(gateway);
108
109         ArrayList mRoutes = (ArrayList)getDeclaredField(linkProperties, "mRoutes");
110         mRoutes.clear();
111         mRoutes.add(routeInfo);
112     }
113
114     public static void setDNS(InetAddress dns, WifiConfiguration wifiConf)
```

```
115             throws SecurityException, IllegalArgumentException,
                    NoSuchFieldException, IllegalAccessException{
116         Object linkProperties = getField(wifiConf, "linkProperties");
117         if(linkProperties == null)return;
118 
119         ArrayList<InetAddress> mDnses = (ArrayList<InetAddress>)getDeclaredField(
                linkProperties, "mDnses");
120         mDnses.clear(); //or add a new dns address , here I just want to replace
                DNS1
121         mDnses.add(dns);
122     }
123 
124     public static Object getField(Object obj, String name)
125             throws SecurityException, NoSuchFieldException,
                    IllegalArgumentException, IllegalAccessException{
126         Field f = obj.getClass().getField(name);
127         Object out = f.get(obj);
128         return out;
129     }
130 
131     public static Object getDeclaredField(Object obj, String name)
132             throws SecurityException, NoSuchFieldException,
133             IllegalArgumentException, IllegalAccessException {
134         Field f = obj.getClass().getDeclaredField(name);
135         f.setAccessible(true);
136         Object out = f.get(obj);
137         return out;
138     }
139 
140     public static void setEnumField(Object obj, String value, String name)
141             throws SecurityException, NoSuchFieldException,
                    IllegalArgumentException, IllegalAccessException{
142         Field f = obj.getClass().getField(name);
143         f.set(obj, Enum.valueOf((Class<Enum>) f.getType(), value));
144     }
145 
146 
147 
148     public void buttonClick(View v){
149       if( seatNumber != null &&
150         Integer.parseInt(seatNumber.getText().toString()) >=20 &&
151         Integer.parseInt(seatNumber.getText().toString()) <= 254)
152         setWiFiParameters(null);
153       else
154       {
155         Toast toast = Toast.makeText(this, "Please fill in with an acceptable
                address (20 - 254)", Toast.LENGTH_SHORT);
156         toast.show();
157       }
```

```
158     }
159
160     private void setWiFiParameters(String ip){
161       String ipAdd = "192.168.1.";
162
163         WifiConfiguration wifiConf = null;
164
165         WifiManager wifiManager = (WifiManager)getSystemService(Context.
              WIFI_SERVICE);
166         WifiInfo connectionInfo = wifiManager.getConnectionInfo();
167         List<WifiConfiguration> configuredNetworks = wifiManager.
              getConfiguredNetworks();
168
169         for (WifiConfiguration conf : configuredNetworks){
170             if (conf.networkId == connectionInfo.getNetworkId()){
171                 wifiConf = conf;
172                 break;
173             }
174         }
175
176
177         try{
178
179           if(ip!=null)
180           {
181             ipAdd = ip;
182           }
183           else
184           {
185             ipAdd += seatNumber.getText().toString();
186               Toast toast = Toast.makeText(this, "IP set to " + ipAdd, Toast.
                    LENGTH_SHORT);
187               toast.show();
188
189           }
190
191             seatNumber.setKeyListener(null);
192
193             setIpAssignment("STATIC", wifiConf); //or "DHCP" for dynamic setting
194             setIpAddress(InetAddress.getByName(ipAdd), 24, wifiConf);
195             setGateway(InetAddress.getByName(DEFAULT_GATEWAY), wifiConf);
196             setDNS(InetAddress.getByName(DEFAULT_DNS), wifiConf);
197             wifiManager.updateNetwork(wifiConf); //apply the setting
198             wifiManager.saveConfiguration(); //Save it
199
200         }catch(Exception e){
201             e.printStackTrace();
202         }
203     }
```

```
204 }
```

## C.3 Network Configuration Code for Android Versions Higher than 5.0

All the code relative to the configuration of IP Address and connection to network with specific name and BSSID as described in section 4.5, for Android versions above 5.0.

```
1  public class NetworkConfigActivity extends Activity {
2
3    private final static String DEFAULT_GATEWAY = "192.168.1.1";
4    private final static String DEFAULT_DNS = "127.0.0.1";
5    private final static String NETWORK_NAME = "AbelNetwork";
6    private final static String BSSID = "F8:D1:11:64:6F:72";
7
8    EditText seatNumber;
9      Button btn;
10
11   @Override
12   protected void onCreate(Bundle savedInstanceState) {
13     super.onCreate(savedInstanceState);
14     setContentView(R.layout.activity_ipconfig);
15
16     seatNumber = (EditText) findViewById(R.id.seatNumber);
17         btn = (Button) findViewById(R.id.InsertBtn);
18
19         Bundle extras = getIntent().getExtras();
20         if(extras != null)
21         {
22           String ip = extras.getString("IP_ADDRESS");
23           String bssid = extras.getString("BSSID");
24           String mc = extras.getString("MC");
25           connectAbelNetwork(bssid);
26
27             //Force IP - taking some tries to fix
28           tryIP(ip);
29
30           Intent intent = new Intent(this, PdTestActivity.class);
31           intent.putExtra("MC", mc);
32           startActivity(intent);
33           finish();
34         }
35         else connectAbelNetwork(BSSID);
36
37   }
38
```

```
39  private void connectAbelNetwork(String bssid2) {
40
41    WifiManager wifiManager = (WifiManager)getSystemService(Context.WIFI_SERVICE);
42
43    WifiInfo currConf = wifiManager.getConnectionInfo();
44
45    WifiConfiguration wifiConf = new WifiConfiguration();
46    wifiConf.SSID = NETWORK_NAME;
47    wifiConf.allowedKeyManagement.set(WifiConfiguration.KeyMgmt.NONE);
48
49    wifiConf.BSSID = bssid2;
50
51
52    if(currConf.getBSSID() == null || !currConf.getBSSID().equals(wifiConf.BSSID))
53    {
54      wifiManager.addNetwork(wifiConf);
55
56      List<WifiConfiguration> list = wifiManager.getConfiguredNetworks();
57      for( WifiConfiguration i : list ) {
58        System.out.println(i.SSID);
59          if(i.SSID != null && i.SSID.equals("\"" + NETWORK_NAME + "\"")) {
60               wifiManager.disconnect();
61               wifiManager.enableNetwork(i.networkId, true);
62               wifiManager.reconnect();
63               break;
64          }
65       }
66    }
67  }
68
69  private void tryIP(String ip) {
70
71    ConnectivityManager connManager = (ConnectivityManager) getSystemService(
          Context.CONNECTIVITY_SERVICE);
72    NetworkInfo mWifi = connManager.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
73
74    int counter = 0;
75
76    while(!mWifi.isConnected() && counter < 100)
77    {
78      setWiFiParameters(ip);
79      counter++;
80    }
81  }
82
83  public void buttonClick(View v){
84      if( seatNumber != null &&
85        Integer.parseInt(seatNumber.getText().toString()) >=20 &&
86        Integer.parseInt(seatNumber.getText().toString()) <= 254)
```

```
87          setWiFiParameters(null);
88        else
89        {
90          Toast toast = Toast.makeText(this, "Please fill in with an acceptable
                address (20 - 254)", Toast.LENGTH_SHORT);
91          toast.show();
92        }
93      }
94
95    @Override
96    public boolean onCreateOptionsMenu(Menu menu) {
97      // Inflate the menu; this adds items to the action bar if it is present.
98      getMenuInflater().inflate(R.menu.ipconfig, menu);
99      return true;
100   }
101
102   @Override
103   public boolean onOptionsItemSelected(MenuItem item) {
104     // Handle action bar item clicks here. The action bar will
105     // automatically handle clicks on the Home/Up button, so long
106     // as you specify a parent activity in AndroidManifest.xml.
107     int id = item.getItemId();
108     if (id == R.id.action_settings) {
109       return true;
110     }
111     return super.onOptionsItemSelected(item);
112   }
113
114   private static Object newInstance(String className) throws ClassNotFoundException
        , InstantiationException, IllegalAccessException, NoSuchMethodException,
        IllegalArgumentException, InvocationTargetException
115     {
116         return newInstance(className, new Class<?>[0], new Object[0]);
117     }
118
119     private static Object newInstance(String className, Class<?>[] parameterClasses
          , Object[] parameterValues) throws NoSuchMethodException,
          InstantiationException, IllegalAccessException, IllegalArgumentException,
          InvocationTargetException, ClassNotFoundException
120     {
121         Class<?> clz = Class.forName(className);
122         Constructor<?> constructor = clz.getConstructor(parameterClasses);
123         return constructor.newInstance(parameterValues);
124     }
125
126     @SuppressWarnings({ "unchecked", "rawtypes" })
127     private static Object getEnumValue(String enumClassName, String enumValue)
          throws ClassNotFoundException
128     {
```

```
129         Class<Enum> enumClz = (Class<Enum>)Class.forName(enumClassName);
130         return Enum.valueOf(enumClz, enumValue);
131     }
132
133     private static void setField(Object object, String fieldName, Object value)
            throws IllegalAccessException, IllegalArgumentException,
            NoSuchFieldException
134     {
135         Field field = object.getClass().getDeclaredField(fieldName);
136         field.set(object, value);
137     }
138
139     private static <T> T getField(Object object, String fieldName, Class<T> type)
            throws IllegalAccessException, IllegalArgumentException,
            NoSuchFieldException
140     {
141         Field field = object.getClass().getDeclaredField(fieldName);
142         return type.cast(field.get(object));
143     }
144
145     private static void callMethod(Object object, String methodName, String[]
            parameterTypes, Object[] parameterValues) throws ClassNotFoundException,
            IllegalAccessException, IllegalArgumentException, InvocationTargetException
            , NoSuchMethodException
146     {
147         Class<?>[] parameterClasses = new Class<?>[parameterTypes.length];
148         for (int i = 0; i < parameterTypes.length; i++)
149             parameterClasses[i] = Class.forName(parameterTypes[i]);
150
151         Method method = object.getClass().getDeclaredMethod(methodName,
                parameterClasses);
152         method.invoke(object, parameterValues);
153     }
154
155     @SuppressWarnings("unchecked")
156     private static void setStaticIpConfiguration(WifiManager manager,
            WifiConfiguration config, InetAddress ipAddress, int prefixLength,
            InetAddress gateway, InetAddress dns)
157             throws ClassNotFoundException, IllegalAccessException,
                    IllegalArgumentException, InvocationTargetException,
                    NoSuchMethodException, NoSuchFieldException, InstantiationException
158     {
159         // First set up IpAssignment to STATIC.
160         Object ipAssignment = getEnumValue("android.net.
                IpConfiguration$IpAssignment", "STATIC");
161         callMethod(config, "setIpAssignment", new String[] { "android.net.
                IpConfiguration$IpAssignment" }, new Object[] { ipAssignment });
162
163         // Then set properties in StaticIpConfiguration.
```

```
164         Object staticIpConfig = newInstance("android.net.StaticIpConfiguration");
165         Object linkAddress = newInstance("android.net.LinkAddress", new Class<?>[]
                { InetAddress.class, int.class }, new Object[] { ipAddress,
                prefixLength });
166
167         setField(staticIpConfig, "ipAddress", linkAddress);
168         setField(staticIpConfig, "gateway", gateway);
169         getField(staticIpConfig, "dnsServers", ArrayList.class).clear();
170         getField(staticIpConfig, "dnsServers", ArrayList.class).add(dns);
171
172         callMethod(config, "setStaticIpConfiguration", new String[] { "android.net.
                StaticIpConfiguration" }, new Object[] { staticIpConfig });
173         manager.updateNetwork(config);
174         manager.saveConfiguration();
175     }
176
177     public void setWiFiParameters(String ip)
178     {
179       String ipAdd = "192.168.1.";
180
181         WifiManager manager = (WifiManager)getSystemService(Context.WIFI_SERVICE);
182         WifiInfo connectionInfo = manager.getConnectionInfo();
183         WifiConfiguration wifiConf = null;
184
185         List<WifiConfiguration> configuredNetworks = manager.getConfiguredNetworks
                ();
186
187         for (WifiConfiguration conf : configuredNetworks){
188             if (conf.networkId == connectionInfo.getNetworkId()){
189                 wifiConf = conf;
190                 break;
191             }
192         }
193
194         if(ip!=null){
195           ipAdd = ip;
196         }
197
198         else
199         {
200           if(seatNumber != null && Integer.parseInt(seatNumber.getText().toString()
                ) >=20 && Integer.parseInt(seatNumber.getText().toString()) <= 254)
201               ipAdd += seatNumber.getText().toString();
202           else
203               ipAdd += "20";
204
205           Toast toast = Toast.makeText(this, "IP set to " + ipAdd, Toast.
                LENGTH_SHORT);
206           toast.show();
```

```
207         }
208
209         seatNumber.setKeyListener(null);
210
211         if (wifiConf != null)
212         {
213             try
214             {
215                 setStaticIpConfiguration(manager, wifiConf,
216                         InetAddress.getByName(ipAdd), 24,
217                         InetAddress.getByName(DEFAULT_GATEWAY),
218                         InetAddress.getByName(DEFAULT_DNS));
219             }
220             catch (Exception e)
221             {
222                 e.printStackTrace();
223             }
224         }
225     }
226 }
```