Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

MASTER THESIS

Development of an instant VoIP audio content provider

Studies: Telecommunications Engineering

Author: Miquel Angel Corbella

Director: Pavel Bezpalec

Co-Director: Marcel Fernández

Year: January 2016

Table of Contents

Acknowledgments

First of all I would like to thank all the people that have been close to me in my development as an engineer and in particular, as a person. Especially I would like to thank to my parents *Joan Corbella Roqueta* and *Mercè Mateu Sans* and my girlfriend *Elisabet Mendez* all the efforts that have done to give me the opportunity to be who I am. This project would not have been able to be developed in a great part without them.

Moreover, I consider that it cannot be forgotten to thank all engineers who did some effort to improve the world developing code for the community without profit. With this project I understand the important of the free and open source community that with a great motivation could create a product better than some companies. In addition, I would like to thank all the volunteers who offered their voice for audio descriptions in different languages and make this project more feasible.

Finally, but not less important, I would like to thank to both my home and host institutions *Universitat Politècnica de Catalunya* and *Czech Technical University in Prague*, respectively, for the basis, concepts and knowledge that they brought to me. At the same time I want to thank my tutor *Pavel Bezpalec* and my co-tutor *Marcel Fernández* for the opportunity they gave me to develop this project.

Abstract (Catalan)

[Catalan]

Avui en dia, obtenir informació instantània sobre el que hom té davant comença a ser molt necessari. Aquest projecte intenta resoldre aquesta necessitat desenvolupant un codi utilitzant plataformes ben establertes de codi lliure.

Utilitzant una aplicació creada per telèfons intel·ligents i mecanismes per identificar la posició del usuari, la aplicació proporcionarà informació rellevant a través de VoIP (audio descripció sobre IP).

L'objectiu principal és implementar un servidor proveïdor de informació. Això es farà a través de una estructura PBX (Private Branch Extension) de codi lliure com Asterisk, y serà contestada per una Raspberry Pi cap al client amb un telèfon intel·ligent.

Abstract (Spanish)

[Spanish]

Hoy en día, obtener información instantánea sobre lo que uno tiene delante empieza a ser muy necesario. Este proyecto intenta solventar esta necesidad desarrollando un código usando plataformas bien establecidas de código libre.

Usando una aplicación creada para teléfonos inteligentes y mecanismos para identificar la posición del usuario, la aplicación proporcionará información relevante a través de VoIP (audio descripción sobre IP).

El objetivo principal es implementar un servidor proveedor de información. Esto se hará a través de una estructura PBX (Private Branch Extension) de código libre y abierto como Asterisk, y será contestada por una Raspberry Pi hacia el cliente con teléfono inteligente.

Abstract (English)

[English]

Nowadays, to obtain instant information from what is in front of oneself is increasingly necessary. This project tries to fulfil this need by developing a code using well-established open source platforms.

Using a self-developed smartphone application and mechanisms for identifying the position of an individual, the application will provide related information through VOIP (audio description over IP).

The main goal is to implement an information provider server. This will be done over a free and open source PBX (Private Branch Extension) framework like Asterisk, and will be replied using a Raspberry Pi to a smartphone client.

1 Introduction

In this chapter it is exposed the main reasons to develop this project and the targets that needs to achieve.

1.1 Context of the project

Nowadays, to obtain instant information from what is in front of oneself is increasingly necessary. Not only people with disabilities need audio descriptions, in most situations in this stressed world it is better than read it. Currently there are different solutions for getting audio descriptions to smartphone clients. However, most of solutions need expensive infrastructure to identify the position of an individual or they just download the audio description taking up too much bandwidth.

Fortunately, it exist a non-expensive solution which can reach many people. This project will try to obtain audio descriptions with a new improved and optimized method.

1.2 Objectives

This project will achieve these main objectives described below.

- To achieve the best solution with least expense.
- Must provide audio descriptions in different languages.
- To be used in an environment enabled for it using Wi-Fi connection and low noise.
- Must be user friendly application.
- To be a non-intrusive application. Should not modify any user parameter from the smartphone.

1.3 Structure of the documentation

This documentation is structured in three blocks. *Previous analysis* chapter, where exposed a description of what to develop. *Development* chapter, where there is explained the steps for getting the desired goal. And finally, *Verifications* chapter, where will test all the parts.

2 Previous Analysis

In this chapter it is exposed how to get the desired goals focused in different areas of analysis. In addition, it is exposed the reasons to develop by this method and its different parts. This chapter is of great importance to be able to create a structured and well-developed project.

2.1 Environment Scheme

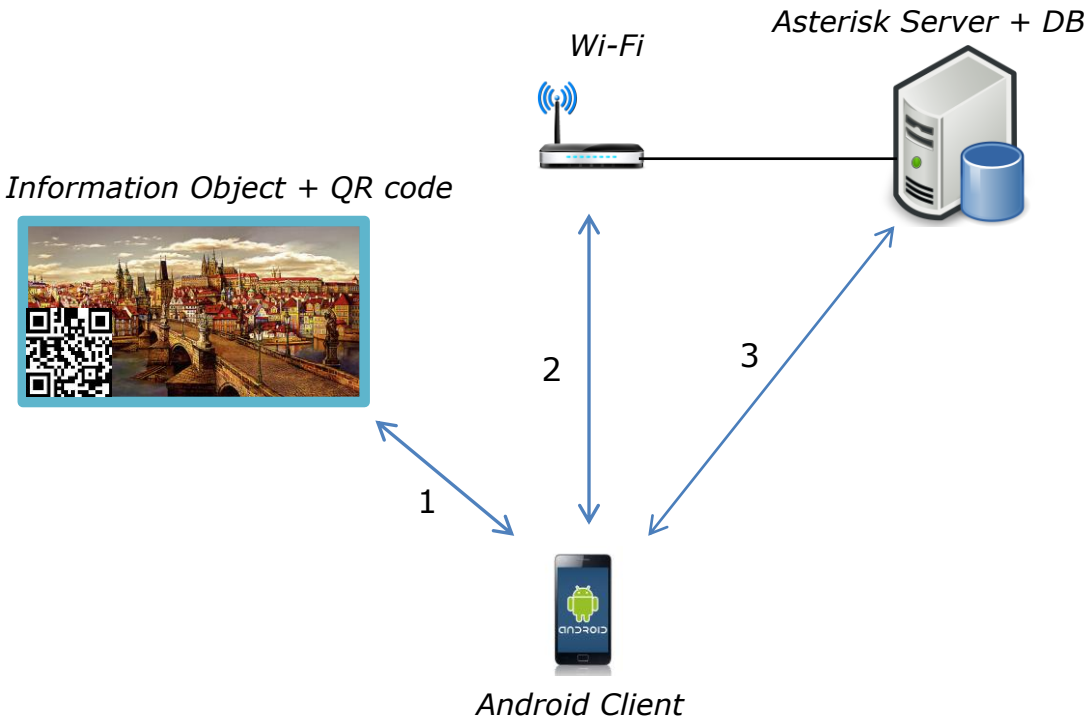

Figure 2-1 Environment Scheme

This scheme shows how the environment should be to solve the main aim, which is to get the audio-description about the *Information Object* to the client. Considering the client has already installed the application on their smartphone. The process is separated in 3 steps according to the previous scheme (see Figure 2-1):

1. Get specific information from QR code on the *Information Object* using a QR Reader in the developed Android app.

2. The *Android Client App* will connect to specific *Wi-Fi Network* determined by the QR information if it is not connected yet.

3. When the Client wants, make a one-way call to *Asterisk Server* to get the audio-description in the previously selected language.

2.2 Analysis of QR Code

QR code system was invented in 1994 by Denso Wave. Its purpose was track vehicles during manufacture; it was designed to allow high-speed component scanning. Although initially used for tracking parts in vehicle manufacturing, QR code now are used in a much broader context, including both commercial tracking applications and convenience-oriented applications aimed at mobile-phone users (termed mobile tagging).

QR codes may be used to display text to the user, to add a vCard contact to the user's device, to open a Uniform Resource Identifier (URI), or to compose an e-mail or text message. Users can generate and print their own QR codes for others to scan and use by visiting one of several paid and free QR code generating sites or apps. The technology has since become one of the most-used types of two-dimensional barcode.[3]

2.2.1 Distinguishing features

The main reason to use QR code to set and get information is because it is well-established and has a good historical trajectory. Compared to different 2D-codes, QR code has better features. According to the standard *ISO/IEC 18004:2015*[1], see reference [2], the QR Code provides the following features:

[1] *ISO/IEC 18004:2015* (published in 2015-02-01) defines the requirements for the symbology known as QR Code. It specifies the QR Code symbology characteristics, data character encoding methods, symbol formats, dimensional characteristics, error

- **High capability encoding of data:**

  While conventional bar codes are capable of storing a maximum of approximately 20 digits, QR Code is capable of handling from several dozen to hundreds times more information. QR Code is capable of handling all types of data such as numeric and alphabetic characters, Kanji, Kana, Hiragana, symbols, binary, and control codes. Up to 7,089 characters can be encoded in one symbol.

- **Small printout size:**

  Since QR Code carries information both horizontally and vertically, QR Code is capable of encoding the same amount of data in approximately one-tenth the space of a traditional barcode.

- **Kanji and Kana capability:**

  As a symbology developed in Japan, QR Code is capable of encoding JIS Level 1 and Level 2 kanji character set. In case of Japanese, one full-width Kana or Kanji character is efficiently encoded in 13 bits, allowing QR Code to hold more than 20% data than other 2D symbologies.

- **Dirt and Damage Resistant:**

  QR Code has error correction capability. Data can be restored even if the symbol is partially dirty or damaged. A maximum 30% of codewords[2] can be restored.

- **Readable from any direction in 360º:**

  QR Code is capable of 360 degree (omni-directional), high speed reading. QR Code accomplishes this task through position detection patterns located at the three corners of the

correction rules, reference decoding algorithm, production quality requirements, and user-selectable application parameters.

[2] A codeword is a unit that constructs the data area. In the case of QR Code, one codeword is equal to 8 bits.

symbol. These position detection patterns guarantee stable high-speed reading, circumventing the negative effects of background interference.

- **Structured appending features:**

  QR Code can be divided into multiple data areas. Conversely, information stored in multiple QR Code symbols can be reconstructed as a single data symbol. One data symbol can be divided into up to 16 symbols, allowing printing in a narrow area.

Knowing these characteristics, it is relevant to use QR code for this project. It is an excellent way to store and obtain information from a little space and save in cost. QR code can provide enough space to store the necessary content to develop this project. In addition, QR code has a good error correction, appropiate for different type of cameras on different android client versions.

2.3 Reasons to choose Android OS

The main reason to develop the application in Android operating system is because it is of utmost relevant in the market. It is the most used operating system in smartphones. Therefore, a development in this OS (Operating System) could reach bigger portion of smartphone users.

According to the global market share, see following Figure 2-2 Global market share, in the 2nd quarter 2015 Android OS is used in 82,2% of the smartphones in the world. Moreover, the historical increasing trajectory of Android OS is something to consider as a well-established work environment where many help resources are easily available.

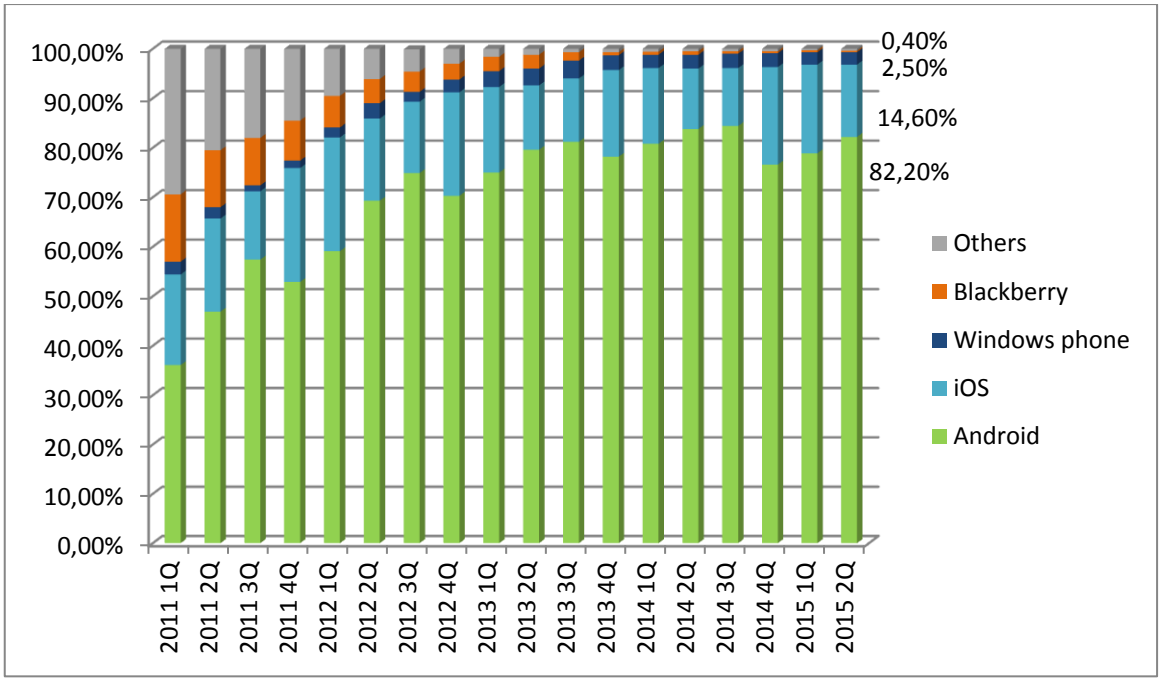

Figure 2-2 Global market share held by the leading smartphone operating systems from 1st quarter 2011 to 2nd quarter 2015. Resource: ©*Statista*

2.4 Analysis of Server

2.4.1 What is Asterisk?

*Asterisk is a free and open source software implementation of a telephone private branch exchange (PBX). It allows attached telephones to make calls to one another, and to connect to other telephone services, such as the Public Switched Telephone Network (PSTN) and Voice over Internet Protocol (VoIP) services. Its name comes from the asterisk symbol, *. It is an open source framework for building communications applications. It can turn an ordinary computer into a communications server.*

Asterisk can support IP PBX systems, VoIP gateways, conference servers and other custom solutions. It is used by small businesses, large businesses, call centers, carriers and government agencies, worldwide.

Asterisk was created in 1999 by Mark Spencer of Digium[9]. Originally designed for Linux, Asterisk runs on a variety of operating systems,

including NetBSD, OpenBSD, FreeBSD, Mac OS X, and Solaris. Asterisk is small enough to run in an embedded environment such as Customer-premises equipment-hardware running OpenWrt OS. [4]

The Asterisk project started in 1999 when Mark Spencer released the initial code under the GPL open source license. Since that time, it has been enhanced and tested by a global community of thousands. Today, Asterisk is maintained by the combined efforts of Digium and the Asterisk community.

Today, there are more than one million Asterisk-based communications systems in use, in more than 170 countries. Asterisk is used by almost the entire Fortune 1000 list of customers. Most often deployed by system integrators and developers, Asterisk can become the basis for a complete business phone system, or used to enhance or extend an existing system, or to bridge a gap between systems. [9]

2.4.2 Reasons to choose Asterisk server

There are plenty of reasons to choose Asterisk as a PBX server instead of others. The main reason is that it is an open source solution without any extra license costs and it can work on any Linux OS computer. This means that it can be used with a considerable cost reduction. Asterisk server should fulfil one of the main goals, *to achieve the best solution with least expense* (see Chapter 1.2).

The principal features of Asterisk are:

• Supports SIP connections	**• Worldwide connection**
• Can save all statistics to MySQL DB	**• Allows interconnection with multiple Asterisk servers**
• Allows conference calling	**• Allows call transfer**
• Allows voice menus	**• Allows call recording**
• Allows voice mails	**• Allows call without registration**
• Manages queues	**• Uses transcoder to calls**
• Allows user to create dial plans	

2.4.3 The SIP connection

The Session Initiation Protocol (SIP) is a communications protocol to signalling and controlling multimedia communications sessions. The protocol defines the messages that are sent between endpoints, which govern establishment, termination and other essential element of a call. SIP connection is commonly used in IP phones and software phones (softphones or software applications simulating phones).

SIP is independent from the underlying transport protocol. It runs on the Transmission Control Protocol (TCP), the User Datagram Protocol (UDP) or the Stream Control Transmission Protocol (SCTP). SIP can be used for two-party (unicast) or multiparty (multicast) sessions. The URI scheme used for SIP is sip and a typical SIP URI is of the form:

sip:username:password@host:port

Figure 2-3 SIP URI scheme

SIP clients typically use TCP or UDP on port numbers 5060 or 5061 to connect to SIP servers and other SIP endpoints. Port 5060 is commonly used for non-encrypted signalling traffic whereas port 5061 is typically used for traffic encrypted with Transport Layer Security (TLS).

A motivating goal for SIP was to provide a signalling and call setup protocol for IP-based communications that can support a superset of the call processing functions and features present in the public switched telephone network (PSTN). SIP by itself does not define these features; rather, its focus is call-setup and signalling.

The features that permit familiar telephone-like operations (i.e. dialling a number, causing a phone to ring, hearing ringback tones or a busy signal) are performed by proxy servers and user agents. Implementation

and terminology are different in the SIP world but to the end-user, the behaviour is similar.

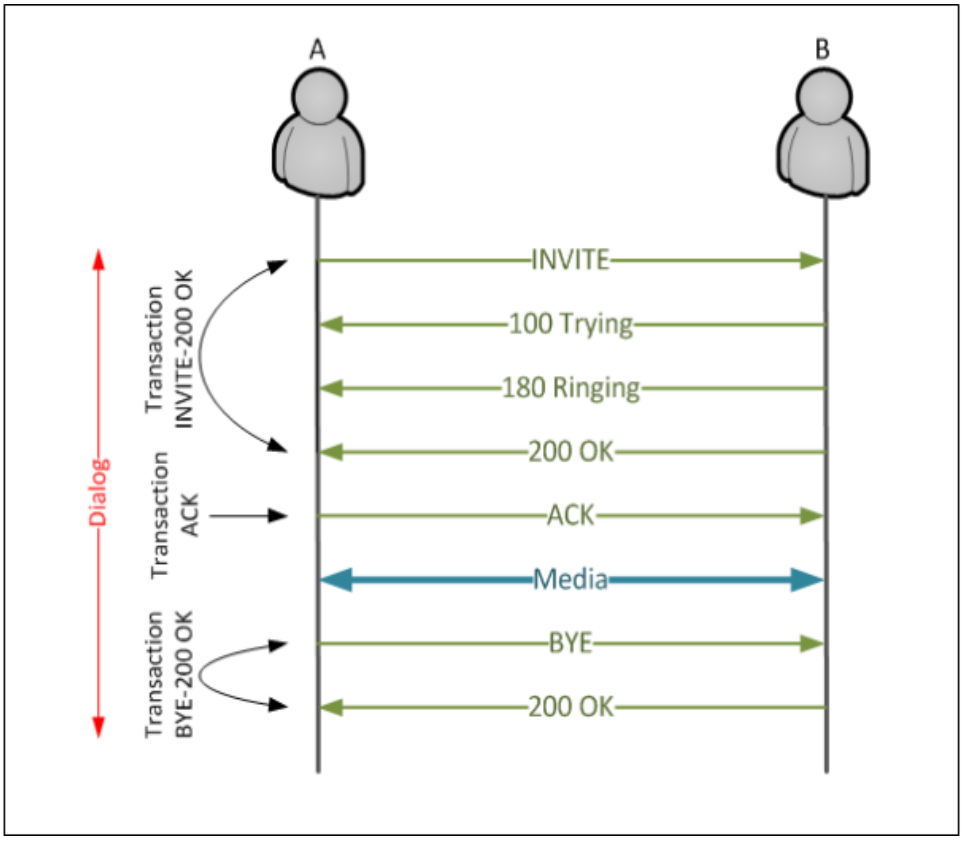

Figure 2-4: Scheme of SIP transaction. Resource: telconotes.wordpress.com

Until now, it exist multiple signalling protocols such as ITU H.323, Cisco SCCP or MGCP. However, it seems that gradually is winning the battle SIP standard. Cisco is gradually adopting SIP as a protocol on their IP telephony systems to the detriment of H.323 and SCCP.

Microsoft has elected SIP as the protocol for its new OCS (office Communication Server), and operators (mobile and fixed phones) are also implementing SIP in its convergence strategy, thereby taking advantage of the scalability and interoperability that provides the SIP protocol. [5]

2.4.4 The Hardware: Raspberry Pi

Raspberry Pi is a series of credit card–sized single-board computers developed in the United Kingdom by the *Raspberry Pi Foundation* with the intention of promoting the teaching of basic computer science in schools and developing countries. The last version and most powerful is Raspberry Pi 2 model B. It has an ARMv7 quad-core processor running at 900 MHz with 1GB of RAM memory.

It costs around 35€ and can run a specific version of Debian O.S. called Raspbian. This Operating System provides a good environment to install and run Asterisk software. This small computer perfectly meets the project objectives because it has a very low cost and enough power to become a prototype solution.

3 Development

3.1 Generating the QR code

In this chapter it is exposed the necessary features that QR code needs for this project. The QR code has to fulfil the following features:

- To have the correct size. Not too big, not to small.
- To be limited to 300 characters to avoid complexity.
- Different content, must be separated by " **;** "
- To include all the content described below.

The content has to fulfil the following pattern according this order:

- Object ID (2 numerical digits)
- Title of object (English)
- Title of object (Czech)
- Title of object (Spanish)
- Brief description (English)
- Brief description (Czech)
- Brief description (Spanish)
- SSID of Wi-Fi Network
- Wi-Fi Password
- Asterisk server IP address (last 2 bytes)

01;Charles Bridge;Karlův most;Puente Carlos;The most famous bridge that cross the Vltava river in Prague, Czech Republic.;Nejslavnější most na řece Vltavě v Praze, v České republice.;El puente más famoso que cruza el rio Moldava en Praga, Republica Checa.;Miquel-VOIP;12345678;10.2

Figure 3-1 Example of QR code and its pattern

The *Figure 3-1* is an example of how the pattern of QR code must look like. It is created by online QR generator website called *https://www.the-qrcode-generator.com/*. The main reason to follow this pattern is because the Android application needs all of these fields for proper performance. The following chapter (see Chapter 3.2) will emphasize on it.

3.2 Android Application

3.2.1 Analysis of core application

In this chapter it is exposed how the different parts of the Android application will be. The application has to be separated in 3 modules or APIs[3] (Application Programing Interface) to accommodate all the functionalities. [7]

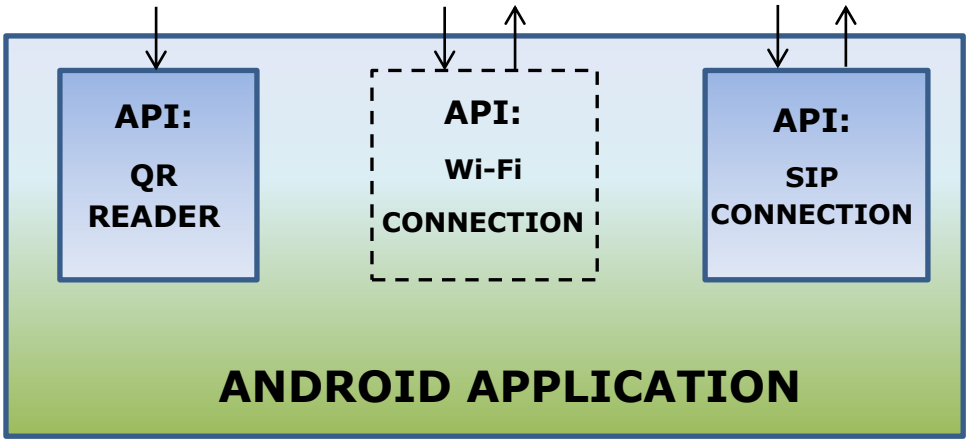

Figure 3-2 Different Blocs or APIs of the Android Application

QR reader API:

This API is responsible to obtain all the content of a specific QR code. It has to fulfil the following features:

[3] In computer programming, an application programming interface (API) is a set of routines, protocols, and tools for building software applications. An API expresses a software component in terms of its operations, inputs, outputs, and underlying types. An API defines functionalities that are independent of their respective implementations, which allows definitions and implementations to vary without compromising the interface. A good API makes it easier to develop a program by providing all the building blocks. The programmers then put the blocks together.

- This API has to be able to take a photo of the QR code and decode it as a string parameter (plain text).
- Has to be able to take the photo in a dark environment.
- Has to check if the string obtained has sense or not. If it has not, it has to advise to de user to repeat the process.
- The application will parse all the string content and separate it in different variables according the QR pattern (see Chapter 3.1)
- Possible API candidates: ZXing API, Barcode API…

The application will use ZXing API. The main reason for using it is because it is an open source project where anyone can obtain the source code or contribute on it. It has a good historical trajectory, is well documented and most of Android developers have used it.

Wi-Fi Connection API:

This API is an integrated part of Android OS and their API. Its responsibility is to establish the network to the android application and has to fulfil the following features:

- Enable Wi-Fi connection if it is disabled.
- Check if the application is connected to the specific Wi-Fi network determined by the QR code scanned before.
- If it is not connected to the specific Wi-Fi network, this API should connect using the SSID and Wi-Fi password provided by QR code.
- As it is a non-intrusive application, it must remove the mentioned network from the phone's Wi-Fi list.

SIP connection API:

This API has to make a call to the Asterisk server to obtain the audio description. This API has to fulfil the following features:

- Check if it is in the same network of the Asterisk server and register on it.

- If it is on the same network, has to enable the possibility to make a call to a specific extension of the Asterisk server.
- Possible API candidates: Integred Android SIP API, JAIN, MjSIP, PjSIP, Doubango…

The extension to call will be determined by the Object ID field of QR code, and a prefix determined by the language chosen. The extension will follow the next pattern:

Figure 3-3 Fields of an extension call

The field of the Prefix will be as following pattern:

Figure 3-4 Possible language chosen and their prefix

According to that pattern an example could be:

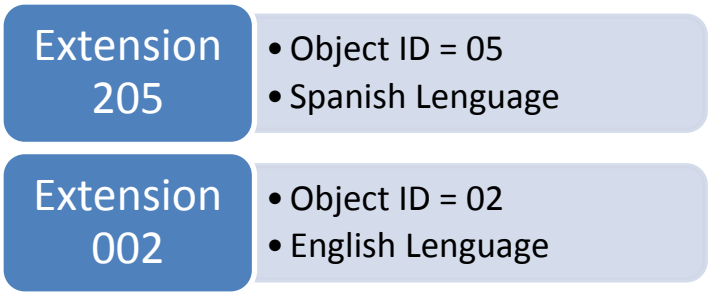

Figure 3-5 Extension examples

3.2.2 Screen navigation

This chapter exposes the behaviour of the Android application and its different activities. The application will follow the next base diagram:

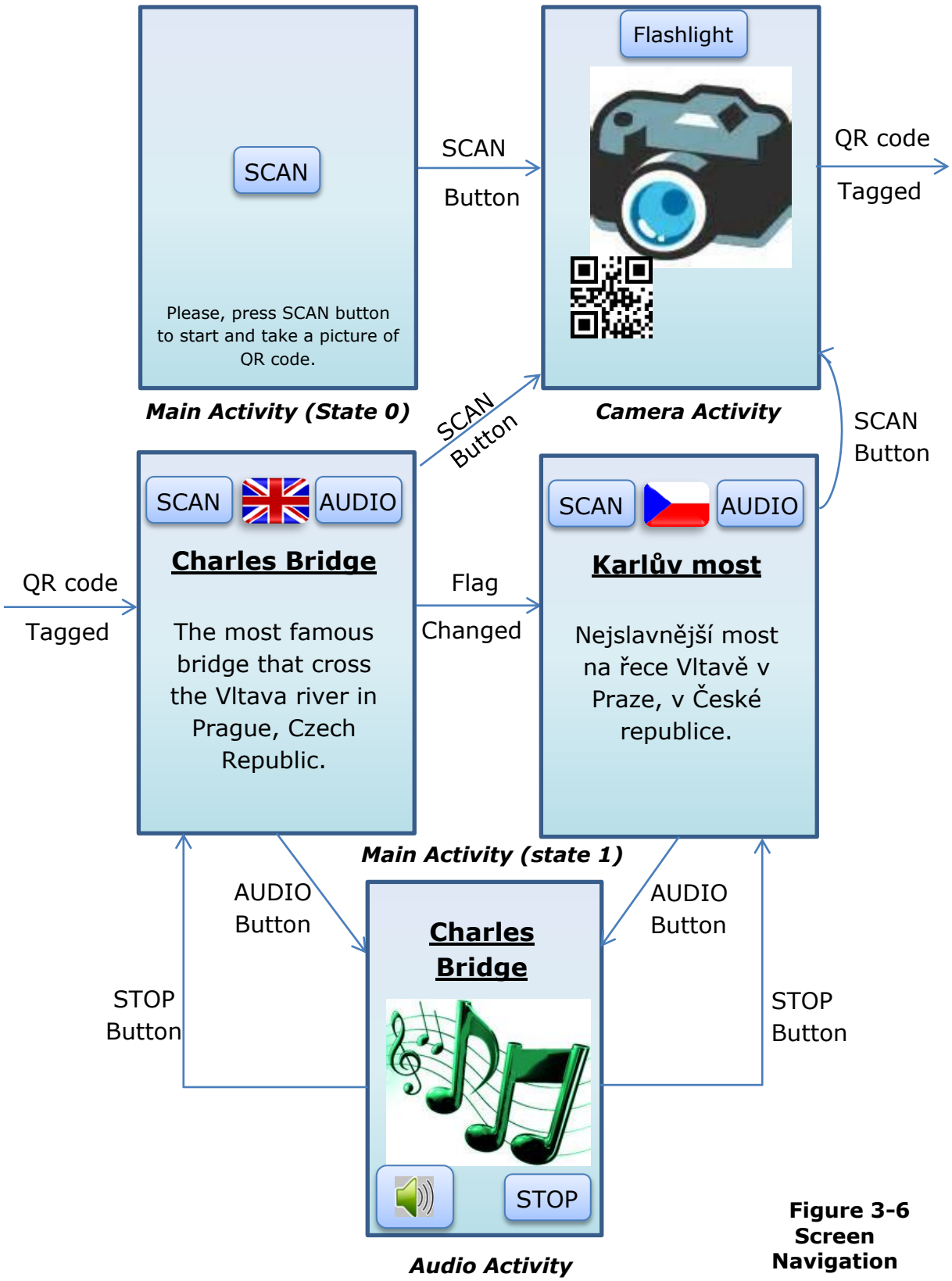

Figure 3-6 Screen Navigation

This preview scheme shows how the user will obtain the audio description step by step. It shows the different Android activities, their basic visual characteristics and the relations between other activities. It is noteworthy that it is a preliminary outline. The following Chapter, see Verification Chapter 4, will expose how the final activities and layout will look like.

To achieve a well-developed application it is important to know the relations between applications and all the possible transitions with them. Next transition diagram, see Figure 3-7, made with *StarUML*™ program has to fulfil this point.

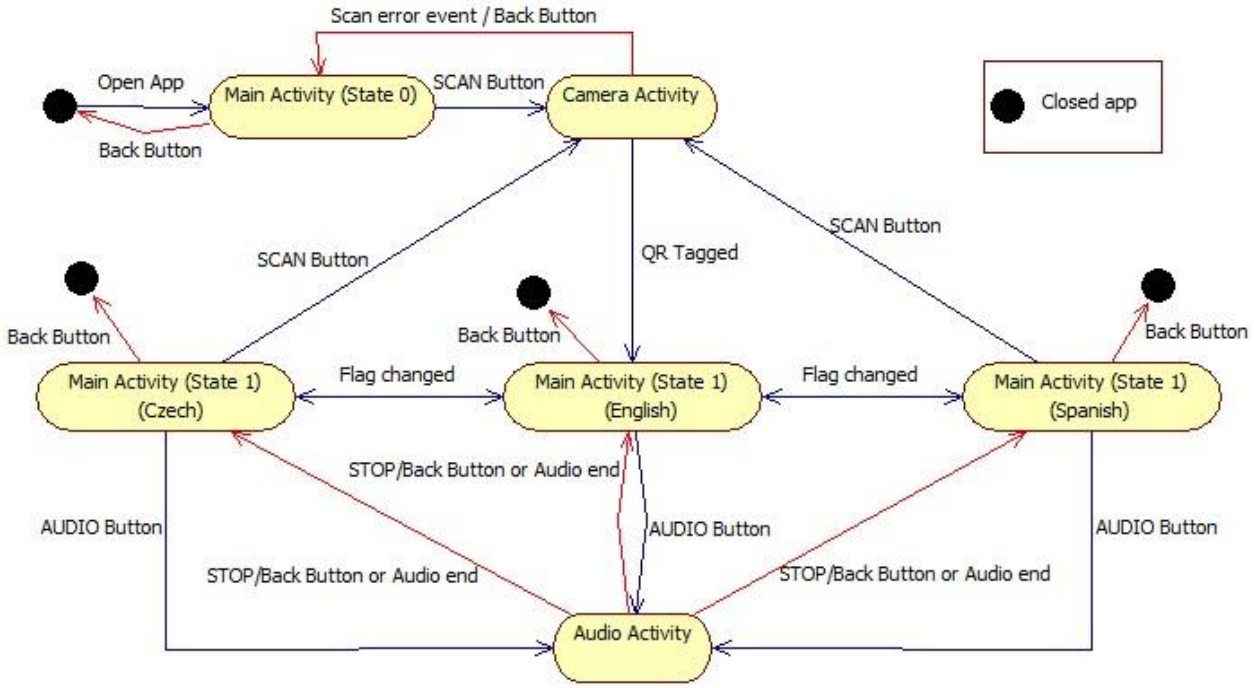

Figure 3-7 Diagram of activities transitions. Program: *StarUML*™

3.2.3 Create an Android Studio project

Once all design has been exposed, the application needs to be developed. Choosing Android Studio as a work environment to create the application, it could be guaranteed a correct behaviour because it is the official program for Android developers and has Google support. In

addition Android Studio has included all the needs including virtual devices for testing.

Firstly, to create an Android Project, it must be specified the minimum Android SDK that the application will run. Lower SDK levels will target more devices. Targeting *API 8: Android 2.2 (Froyo)*, the app will run on approximately 100% of the devices that are active on the Google Play Store. It is possible to modify it later. Finally, choosing to create a Blank Activity will be enough to start developing.

One important point to highlight is that Android Studio has the possibility to work with subversion repository. Subversion is a software versioning and revision control system distributed as free software under the apache license. Software developers use Subversion to maintain current and historical versions of files such as source code, web pages, and documentation. It is extremely recommended to use this tool to avoid problems and have a historical control of all the developing process.

3.2.4 QR reader API – ZXing

Once created an Android project it is necessary to include the different APIs. In this chapter it will be explained the API QR reader.

The code needed is in Github ZXing Official repository[4]. However, to integrate the library and use it by intents, the ZXing Official does not have an easily method. The code is not ready for integration and usage as a library in the same application. For that reason it is used ZXing Android Embedded[5] from Github repository by *rkistner* developer.

ZXing Android Embedded is a barcode scanning library. It is not affiliated with the official ZXing project but is loosely based on it. The main features to choose this specific repository are:

[4] ZXing Project in Github repository: https://github.com/zxing/
[5] ZXing Android Embedded: https://github.com/journeyapps/zxing-android-embedded

- Can be used via Intents.
- Can be embedded in an Activity for advanced customization of UI and logic.
- Scanning can be performed in landscape or portrait mode.
- Camera is managed in a background thread, for fast start-up time.

This API is ready for integration in the application and can be used as a library via Intents. This library achieve the requisites commented in Chapter 3.2.1.

The integration of ZXing API requires to increase the Android minimum SDK up to *API 11: Android 3.0 (Honycomb)* that will run in approximately 94% of the Android devices because it uses some functionalities that were integrated form Android 3.0 onwards.

The image below (Figure 3-8) shows the code necessary in *Main Activity*; it makes the Intent to start the *Camera Activity* and waits for the result. This is a common functionality of Android called *IntentResult* that provides a mechanism to start another activity only to obtain some information and close it. The function *scanCustomScanner* is the function used when SCAN button is pressed and the function onActivityResult is the function that capture the result, in this case the string of the QR code.

```
private String ScannedInfo="0";

public void scanCustomScanner(View view) {
    new IntentIntegrator(this).setCaptureActivity(CustomScannerActivity.class).initiateScan();
}
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    IntentResult result = IntentIntegrator.parseActivityResult(requestCode, resultCode, data);
    if(result != null) {
        if(result.getContents() == null) {
            Log.d("MainActivity", "Cancelled scan");
            Toast.makeText(this, "Cancelled", Toast.LENGTH_LONG).show();
            ScannedInfo="0";
        } else {
            Log.d("MainActivity", "Scanned");
           // Toast.makeText(this, "Scanned: " + result.getContents(), Toast.LENGTH_LONG).show();
            ScannedInfo=result.getContents(); //SAVE SCANNED INFO AS A STRING
            ParseMessageScanned();
        }
    } else {
        Log.d("MainActivity", "Weird");
        // This is important, otherwise the result will not be passed to the fragment
        super.onActivityResult(requestCode, resultCode, data);
    }
}
```

Figure 3-8: Code to start the Camera Activity using *IntentResult* in *Main Activity*.

The *ParseMessageScanned* is a function which parses the *ScannedInfo* string variable, that contains all the QR code, and if it is different of 0 it will put all the content to different global variables as described in Chapter 3.1. In addition, if the QR code is well scanned, the *Main activity* changes their state with the function *ActivityState2*. It modifies the visual characteristics as the buttons positions and makes visible the title and subtitle.

Furthermore, the function *ShowScannedResults* shows the information title to the *title bar* and the brief description to *subtitle bar* in *Main Layout*. After that, it calls the function *ConnectWifi* to connect to the Wi-Fi, it will be more explained in next chapter.

```
private void ParseMessageScanned ()
{
    if (!ScannedInfo.equals("0")) //not equals
    {
        String[] InfoParsed = ScannedInfo.split("[;]");
        if (InfoParsed.length==10)
        {
            ObjectID=InfoParsed[0];
            TitleEN=InfoParsed[1];
            TitleCZ=InfoParsed[2];
            TitleES=InfoParsed[3];
            DescripEN=InfoParsed[4];
            DescripCZ = InfoParsed[5];
            DescripES=InfoParsed[6];
            NetworkSSID=InfoParsed[7];
            NetworkPASS=InfoParsed[8];
            ServerIP=InfoParsed[9];
            ExtensionToCall=PREFIX+ObjectID;//PREFIX + ObjectID = Extension to CALL
            ReportMessage("WELL SCAN!");
            ActivityState2();
            ShowScannedResults();
            ConnectWifi();
        }
        else
            ReportMessage("WRONG SCAN, please repeat it.");
    }
    else
        ReportMessage("WRONG SCAN, please repeat it.");
}
```

Figure 3-9: *ParseMessageScanned* function in *Main Activity* class.

According to the requirements, see Chapter 3.2.1, *ParseMessageScanned* function is the responsible of advice the user if the message scanned is not correct. *ReportMessage* is a simple function created to avoid an extensive line of code; it contains the Toast functionality to show a message pop-up to the user. Although, the QR code has error correction capabilities and if the code scanned does not have 10 fields or are not well captured, it will advise the user to repeat the scan.

To fulfil the requirement of be able to capture QR code in a dark environment, it is necessary introduce a button to control the flashlight of the smartphone in *Camera Layout*. The next Figure 3-9 shows the code to integrate this functionality in *Camera Activity* class.

```
//Check if the device's camera has a Flashlight. Returns true or false.
private boolean hasFlash() {
    return getApplicationContext().getPackageManager()
            .hasSystemFeature(PackageManager.FEATURE_CAMERA_FLASH);
}

public void switchFlashlight(View view) {
    if (getString(R.string.turn_on_flashlight).equals(switchFlashlightButton.getText())) {
        barcodeScannerView.setTorchOn();
    } else {
        barcodeScannerView.setTorchOff();
    }
}

@Override
public void onTorchOn() {
    switchFlashlightButton.setText(R.string.turn_off_flashlight);
}

@Override
public void onTorchOff() {
    switchFlashlightButton.setText(R.string.turn_on_flashlight);
}
```

Figure 3-10: Code to turn on/off the flashlight in *Camera Activity*.

The function *hasFlash* checks if the smartphone have flash for the camera. This function is called when the *Camera Activity* starts. If it has flashlight, it returns true and the flashlight button in *Camera Layout* will be visible, otherwise if not. The other function, *switchFlashlight* is the function used when the Flashlight button is pressed, it changes the name of the button and it set the flashlight on or off.

3.2.5 Wi-Fi connection API

In this chapter it is exposed the functions of API Wi-Fi and their integration with the rest of source code. As mentioned in Chapter 3.2.1 the API Wi-Fi is already integrated in Android functionalities, uniquely needs to apply the correct functions in the correct moment. To achieve this, it is good to separate in other class all the hard functions to avoid complexity in *Main Activity* class; it is created *WifiConnection* class.

The next Figure (Figure 3-11) shows the functions in *Main activity* to connect the specific Wi-Fi determined by the global variables

NetworkSSID and *NetworkPASS.* The function *ConnectWifi* is the responsible to check if it is already connected or needs to connect.

```
public api.wifi.WifiConnection WifiC;

public void ConnectWifi ()
{
        Context context = getApplicationContext();
        if (WifiC.CheckWifiNetwork(context, NetworkSSID))
            ReportMessage("WiFi was connected before !");
        else {
            ReportMessage("Starting WiFi connection...");
            StartConnectionWifi();
        }
}

public void StartConnectionWifi()
{
    try {
        Context context = getApplicationContext();
        if (NetworkSSID != null && NetworkPASS != null) {
            WifiC.ConnectToWifi(context, NetworkSSID, NetworkPASS);
            Thread.sleep(600); //few time to wait for connect to Wi-Fi
            if (WifiC.CheckWifiNetwork(context, NetworkSSID))
                ReportMessage("WiFi well Connected !");
            else
                ReportMessage("WIFI ERROR - not connected");
        } else
            ReportMessage("NetworkSSID or NetworkPass ERROR, please RESCAN");
    }catch (Exception ex)
    {
        ReportMessage(ex.toString());
    }
}
```

Figure 3-11: Application of Wi-Fi connection in *Main Activity* class.

To avoid complexity in *ConnectWifi* function it is necessary to separate it in two functions, *ConnectWifi* and *StartConnectionWifi*. According the statistics of *Source Monitor* program, this code (in the same function) increases the complexity and needs to be separated; in Chapter 4.3 will explained the *Source Monitor* program. For that reason, *ConnectWifi* function decides connect to the network or not, checking if it is already connected in the specific network. If it is not connected yet, it calls the function *StartConnectionWifi* which has the responsibility to connect the specific network, and check it again.

```
public static void ConnectToWifi (Context context, String networkSSID, String networkPass)
{
    WifiConfiguration conf = new WifiConfiguration();
    conf.SSID = "\"" + networkSSID + "\"";   // String should contain ssid in quotes
    conf.preSharedKey = "\""+ networkPass +"\"";

    WifiManager wifiManager = (WifiManager)context.getSystemService(Context.WIFI_SERVICE);
    wifiManager.setWifiEnabled(true); //ENABLE WIFI CONNECTION (ANTENA MODE ON)

    List<WifiConfiguration> list = wifiManager.getConfiguredNetworks();

    //If the network wasn't in the phone WiFi list, then added an remake the list.
    // If doesn't check, it will add the same repeated network
    if(!CheckIfWifiIsIn_PhoneWifiList(context, networkSSID, networkPass)) {
        wifiManager.addNetwork(conf);
        list = wifiManager.getConfiguredNetworks();
    }

    for( WifiConfiguration i : list ) {
        if(i.SSID != null && i.SSID.equals("\"" + networkSSID + "\"")) {
            wifiManager.disconnect();
            wifiManager.enableNetwork(i.networkId, true);
            wifiManager.reconnect();
            break;
        }
    }
}
```

Figure 3-12: *ConnectToWifi* function in *WifiConnection* class

The most important function in *WifiConnection* class is *ConnectToWifi* (see Figure 3-12). This function turns on the Wi-Fi antenna and makes a request if the network is already on the phone list using the function *CheckIfWifiIsIn_PhoneWifiList*. It is really important to check it before aggregate any Wi-Fi SSID, because the API Android functions does not take care in that and can aggregate the same Wi-Fi address although it already exist, creating a duplicate network SSID. After check it, it aggregates the Wi-Fi to the phone's Wi-Fi list, as needed. Then it disconnects any possible connection and reconnects to the specific one.

Another important functionality to meet the requirement is to be a non-intrusive application with *DisconnectWifiAndRemoveIt* function in *WifiConnection* class (see Figure 3-13). This function is called before the application is closed and it is the responsible to find the specific network in the phone's Wi-Fi list, remove it and turn off the Wi-Fi antenna to keep the mobile in its original state.

```
public static void DisconnectWifiAndRemoveIt(Context context, String networkSSID, String networkPass)
{
    if (CheckIfWifiIsIn_PhoneWifiList(context,networkSSID,networkPass))
    {
        WifiConfiguration conf = new WifiConfiguration();
        conf.SSID = "\"" + networkSSID + "\"";   //String should contain ssid in quotes
        conf.preSharedKey = "\""+ networkPass +"\"";

        WifiManager wifiManager = (WifiManager)context.getSystemService(Context.WIFI_SERVICE);
        List<WifiConfiguration> list = wifiManager.getConfiguredNetworks();

        for( WifiConfiguration i : list ) {
            if (i.SSID != null && i.SSID.equals("\"" + networkSSID + "\"")) {
                wifiManager.setWifiEnabled(false); //DISABLE WIFI CONNECTION (ANTENA MODE OFF)
                wifiManager.removeNetwork(i.networkId); //FORGET NETWORK from phone wifi list
            }
        }
    }
}
```

Figure 3-13: *DisconnectWifiAndRemoveIt* function from *WifiConnection* class.

3.2.6 SIP connection API - Doubango

One of the most difficult part of the project is to integrate the API SIP connection. The problem of this kind of APIs is that most of them are not ready to integrate in other projects and all functionalities must be known, spending too much time trying to understand all the documentation and correcting integration errors.

One possible candidate could be the own *Android SIP API*[6], because it has the SIP functionality already integrated. However, the problem is that even though the specification says the SIP API is available for Android devices from Android 2.3 version, it is not. The Android SIP API is not supported on all devices. Although, it has been integrated with the rest of the APIs and tested, it does not run. In most cases, the API will be supported but not VOIP. For that reason this API is discarded.

On the one hand, the most used API for most of developers is PJSIP[7]. This API is free and open source multimedia communication library written in C language. It implements SIP protocol with rich multimedia

[6] Android SIP API→ https://developer.android.com/guide/topics/connectivity/sip.html
[7] PJSIP library official website → http://www.pjsip.org/

framework. It supports audio, video, presence, instant messaging and has extensive documentation. However, PJSIP API has a low-level and requires to have a good knowledge of the SIP protocol. The developer must know what headers, what messages, what requests to send at the right time and have a well control of the sockets. There is nothing like calling a simple *register* method to help handle SIP server registration. Another problem of this API is that it is not written in Java and is not going to be well optimized. Although, it has been attempted to integrate with the rest of the application the problems that it has are too many. For this reasons this API is also discarded.

On the other hand, there is Doubango API. This one can achieve all the requirements established in Chapter 3.2.1, for two reasons:

- Doubango offers a high level API that makes SIP development much easier and faster.
- Doubango has an Android library that can be integrated easily. The hard part of the integration is already done.

For these two reasons *Doubango's* API is the one chosen. The source code integrated in the application is from the Android version library which is in GitHub resource called *IMSDroid*[8]. IMSDroid is an Android application that uses the Doubango Android library.

According to this, the library package *androidngnstack* is integrated from *IMSDroid* resource code. Once it is well integrated, it is necessary to start the library engine and sip services in Main Activity.

The image bellow, see Figure 3-14, shows the function *SipRegister* in *Main Activity* class. This function is the responsible to start the *Doubango* engine and start a sip service that allows making calls, sending and listening to SIP dialog. It is important to know all the functions that control the SIP API must be in the same class, because when the sip engine starts it opens a socket in the context of the class, it is called to establish the SIP communication and the rest of the functions interact with this socket.

[8] IMSDroid from Doubango → https://github.com/DoubangoTelecom/imsdroid

```
public void SipRegister()
{
    try {
        // Get engines
        mEngine = NgnEngine.getInstance();
        mSipService = mEngine.getSipService();

        SipConfigurationDetails(sip_username, "192.168."+ServerIP, 5060, SipPassword);

        // Register broadcast receivers
        regBroadcastReceiver = new RegistrationBroadcastReceiver();
        final IntentFilter intentFilter = new IntentFilter();
        intentFilter.addAction(NgnRegistrationEventArgs.ACTION_REGISTRATION_EVENT);
        registerReceiver(regBroadcastReceiver, intentFilter);
        // Incoming call broadcast receiver
        final IntentFilter intentRFilter = new IntentFilter();
        callStateReceiver = new CallStateReceiver();
        intentRFilter.addAction(NgnInviteEventArgs.ACTION_INVITE_EVENT);
        registerReceiver(callStateReceiver, intentRFilter);

        initializeManager();
    }
    catch(Exception ex)
    {
        ReportMessage(ex.toString());
    }
}
```

Figure 3-14: *SipRegister* function in *Main Activity* class.

When *initializeManager* function is called the application sends an *INVITE* packet according the SIP dialog, see Figure 2-4, with all the parameters previously configured with *SipConfigurationDetails* function. SipConfigurationDetails have as parameters:

- String sip_username = "1000";
- "192.168." + ServerIP as a complete local server IP address.
- 5060 as a Port to communicate in the Asterisk server.
- String SipPassword = "app12345"

With this function, these parameters are included in the engine of the library for easy treatment.

The next relevant function that needs this API is RegistrationBrodcastReceiver, see Figure 3-15. This function is the responsible to get the SIP dialog replies.

```
public class RegistrationBroadcastReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        final String action = intent.getAction();
        // Registration Event
        if(NgnRegistrationEventArgs.ACTION_REGISTRATION_EVENT.equals(action)){
            NgnRegistrationEventArgs args = intent.getParcelableExtra(NgnEventArgs.EXTRA_EMBEDDED);
            if(args == null){
                ReportMessage("Invalid event args");
                Log.d("DEBUG", "Invalid event args");
                return;
            }
            SwitchBroadcastReceiver (args); //refactoring for avoid complexity
        }
    }
}
```

Figure 3-15: *RegistrationBrodcastReceiver* function in *Main Activity* class.

Once it has the replies from server in SIP dialog (saved in local variable *args*), *SwitchBrodcastReceiver* function checks the SIP argument and identify if it is well connected or not. If so, they call the function *MakeCall* for establish a call.

To avoid complexity it is necessary to separate BrodcastReceiver in two classes. RegistrationBrodcastReceiver, see Figure 3-15, and SwitchBrodcastReceiver, see Figure 3-16.

The image below shows the function *SwitchBrodcastReceiver* which according to message received it applies specific content. As you can see, when it receives a *REGISTRATION_OK* event message it starts the function *MakeCall* to make the SIP call with the global variable *ExtensionToCall* as the dial number.

```
//This function is for avoid complexity in RegistrationBroadcastReceiver class
public void SwitchBroadcastReceiver (NgnRegistrationEventArgs args)
{
    switch(args.getEventType()){
        case REGISTRATION_NOK:
            ReportMessage("Failed to register :(");
            Log.d("DEBUG", "Failed to register :(");
            break;
        case UNREGISTRATION_OK:
            ReportMessage("You are now UN-registered");
            Log.d("DEBUG", "You are now unregistered :)");
            break;
        case REGISTRATION_OK:
            ReportMessage("You are now registered :)");
            Log.d("DEBUG", "You are now registered :)");
            MakeCall(ExtensionToCall); /////MAKE CALL
            break;
        case REGISTRATION_INPROGRESS:
            ReportMessage("Trying to register...");
            Log.d("DEBUG", "Trying to register...");
            break;
        case UNREGISTRATION_INPROGRESS:
            ReportMessage("Trying to unregister...");
            Log.d("DEBUG", "Trying to unregister...");
            break;
        case UNREGISTRATION_NOK:
            ReportMessage("Failed to unregister :(");
            Log.d("DEBUG", "Failed to unregister :(");
            break;
    }
```

Figure 3-16: *SwitchBrodcastReceiver* function in *Main Activity* class.

To make the call it is necessary the function MakeCall, see Figure 3-18. This function creates an outgoing session and establishes the SIP call with the format:

```
sip:phoneNumber@"192.168."+ServerIP
```

Figure 3-17: Format to call used in MakeCall function in Main Activity class

```
public void MakeCall(String phoneNumber)
{
    ReportMessage("CALL: "+phoneNumber);
    final String validUri = NgnUriUtils.makeValidSipUri(String.format("sip:%s@%s",
                                                    phoneNumber, "192.168."+ServerIP));

    NgnAVSession avSession = NgnAVSession.createOutgoingSession(mSipService.getSipStack(),
                                                        NgnMediaType.Audio);

    Intent i = new Intent();
    i.setClass(this, CallActivity.class);
    i.putExtra("SipSession", avSession.getId());
    startActivity(i);

    avSession.makeCall(validUri);
}
```

Figure 3-18: MakeCall function in Main Activity class.

Once the call is established the Call Activity in *CallActivity* class starts. This class shows another activity layout where all the audio description is directly received from the Asterisk server. It allows the user to stop receiving content by *FINISH button* or enable or disable the speaker by *Speaker image button*.

3.2.7 Integration of all source code

The integration of all this source code is one of the most difficult part of this project. Although the libraries were ready for integration, it is necessary to well know all Android platform and their particularities. To achieve this point, it was necessary to test different APIs to know well which had the easiest integration.

One of the most important file to consider in Android is the *Android Manifest*. This file is written in XML language and it defines the behaviour of the application and their different class.

The image below, see Figure 3-19, shows how the application and the activities are defined in *Android Manifest* file. This file defines the screen orientation and other important parameters. All activities are defined as fixed portrait mode to avoid complexity. Other important parameter is to

define which activity is the main and has to be visible in the launcher as an executable application. In this project it is *MainActivity*.

```
<application
    android:allowBackup="true"
    android:description="This app can get you an instant VoIP audio content usin..."
    android:icon="@mipmap/ic_launcher"
    android:label="Miquel-VoIP"
    android:theme="@style/AppTheme"
    tools:replace="android:icon" >
    <activity
        android:name=".MainActivity"
        android:label="Miquel-VoIP"
        android:screenOrientation="portrait" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity
        android:name="api.sip.CallActivity"
        android:label="CallActivity"
        android:screenOrientation="portrait" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
        </intent-filter>
    </activity>
    <activity
        android:name="api.zxing.CustomScannerActivity"
        android:label="CustomScannerActivity"
        android:screenOrientation="portrait" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
        </intent-filter>
    </activity>
</application>
```

Figure 3-19: Application part of *Android Manifest* file.

Without going into details of design patterns, the structure of all source code is showed in Figure 3-20. As commented in previous chapters, the two libraries are integrated as two different modules, *androidngnstack* as SIP connection API, and *zxing-android-embedded* as QR reader API. Between the two library modules there is the core of the application, called *app* module. This module contains all the classes, activities and layouts used by the application.

To avoid complexity, the structure of the classes does not have a design pattern, but is separated in different folders for easy understanding. The *java* folder contains all the core classes. It is separated in two folders, *miquelcorbella.miquel_voip* folder and *api* folder. The *miquelcorbella.miquel_voip* folder contains the *MainActivity* class which controls the rest of the application. In addition, *api* folder contains the three different *apis* with each specific class inside.

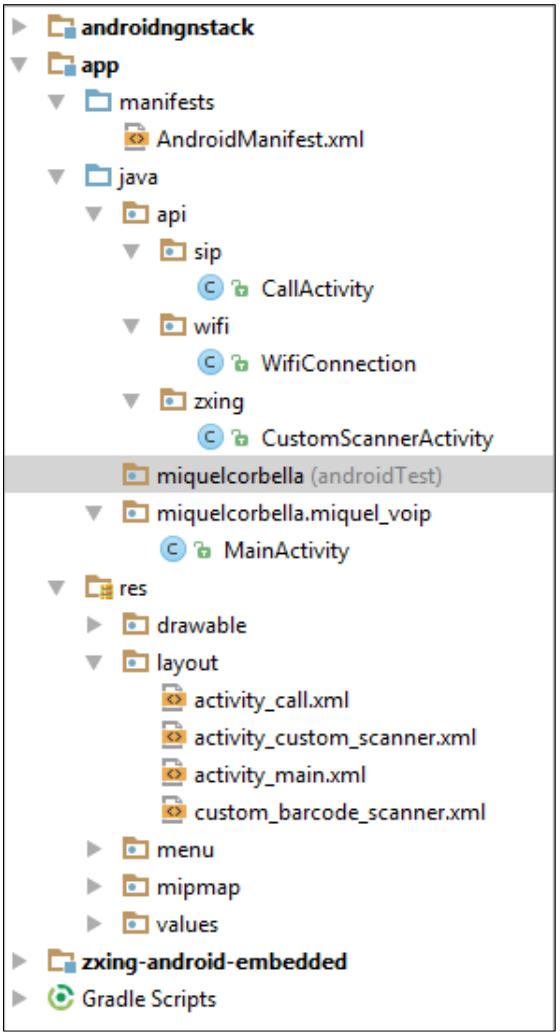

Figure 3-20: Structure of all source code.

According to the Chapter 3.2.3, all the source code is integrated in a subversion repository[9] to avoid problems and have a historical control for all the development process.

[9] Own project SVN Repository → svn://mikynas.noip.me/PFC-Android

3.3 Asterisk Configuration

In this chapter it is exposed the installation and configuration of the server. As explained in Chapter 2.4, the chosen server is Asterisk software in a Raspberry Pi computer.

3.3.1 Installation

It is important to separate the installation it in two steps. The installation and preparation of Raspberry Pi and the installation of the program software Asterisk.

Preparation of Raspberry Pi:

To run and execute an Operating System in Raspberry Pi it first has to be installed in a MicroSD card. The selection of the type and model of the MicroSD is really important for the speed access to the memory. It is recommended to choose the fastest possible. It is recommended to select the simplest and fastest possible Operating System, for that reason Raspbian O.S. could provide a good environment to run Asterisk. Raspbian is a specific operating system based on Debian O.S. especially designed and optimized for Raspberry Pi hardware. Once the O.S. is installed and the Raspberry Pi is running it is recommended to update to the last version using next command:

```
sudo apt-get update && apt-get upgrade
```

The only configuration that Raspberry Pi needs for this project is to configure its IP address as static because in the QR code it is fixed. For that, it requires to modify the *interface* file adding the specific address. This file is located in */etc/network* directory. One example is in Figure 3-21.

```
auto lo
iface lo inet loopback
iface eth0 inet static

address 192.168.10.2
netmask 255.255.255.0
network 192.168.10.0
broadcast 192.168.1.255
gateway 192.168.1.254

auto wlan0
allow-hotplug wlan0
iface wlan0 inet manual
wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
```

Figure 3-21: *Interface* file modifications to establish a static IP in Raspberry Pi.

Asterisk installation:

To install Asterisk in Raspbian it is necessary to follow few steps. Firstly, download the libraries necessary for Asterisk typing the command below. It is possible Raspberry Pi already has some libraries and can be avoided.

```
sudo apt-get install build-essential libncurses5-dev wget libssl-dev libnewt-dev libxml2-dev linux-headers-$(unaname -r) libsqlite3-dev uuid-dev libjasson-dev
```

Then, download the Asterisk package file and extract it. Using the minimal Asterisk version 1.4 is enough for this project.

```
sudo wget http://downloads.asterisk.org/pub/telephony/asterisk/old-releases/asterisk-1.4.44.tar.gz
sudo tar xvf asterisk-1.4.44.tar.gz
cd asterisk-1.4.44
```

Once it is done, it needs to compile the extracted configuration files using next command and waiting for a while.

```
sudo ./configure
```

Next step is to type *make menuconfig*. This command allows the user to choose which modules are going to be installed. For this project it does not need an extra module, the default installation is enough.

```
sudo make menuconfig
```

The only special step Raspberry Pi needs differently than when installing in other computer is that it works in ARM architecture. For that reason it is important to modify *makeopts* file (in the build directory) and specify the arm architecture. To modify the line below is enough.

```
#PROC=armv61
PROC=arm
```

Once this file is modified, it can start the global compilation of all Asterisk modules typing *make* command. It will take 30-40 minutes to compile everything.

```
sudo make
```

Finally, the last step is to install all the compiled Asterisk modules typing *sudo make install* and execute it with *sudo service asterisk start*.

```
sudo make install
sudo service asterisk start
```

3.3.2 Dial Plan and extensions

In this chapter it is exposed the configuration needed in Asterisk once it is already installed. According the book *Asterisk: The Future of Telephony[1]*, there are two important files that need to be modified. These files are: *sip.conf* and *extensions.conf*

Sip.conf

This file determines the characteristics of the sip terminals that will be connected in Asterisk server. To configure it, it is recommended to delete all default configurations, to avoid security problems, and introduce the configuration needed. To specify one sip address with specific configuration is necessary in this project, see Figure 3-22.

```
[1000]
type=peer
context=app
host=dynamic
allowguest=no
secret=app12345
```

Figure 3-22: *Sip.conf* file

Creating a SIP address *1000* with the password *app12345* (secret field) will restrict the access only for the Android application SIP clients. All other configuration will reject the registration. The parameter *type* as a *friend* allows this extension to make calls and receive them. To use *type* parameter as a *peer* will be better to allow only the clients to make calls rather than receive them. However, in order to test the application the parameter *type* is used as *friend*. The parameter *host* specifies the hostname or IP address of a SIP peer or user. In this project the parameter *host* is defined as *dynamic* because the clients will not always have the same IP address.

One important parameter that must be in this extension is *allowguest*. This parameter allows other users to connect in this extension without authentication. This parameter is set as *no* to avoid security problems; all users have to use extension name and password authentications.

extensions.conf

This file determines the Dial Plan. It specifies to Asterisk the steps to deliver the different audio descriptions to its clients. To avoid security problems it is recommended to delete all default configurations and

introduce the needed configuration. The figure below shows the *extensions.conf* file, see Figure 3-23.

```
[app]
;prefix 0=EN ; 1=CZ ; 2=ES
;----------Charles Bridge--------
exten => 001,1,Answer()
exten => 001,2,PlayBack(/home/audio/charles-bridge-en)
exten => 101,1,Answer()
exten => 101,2,PlayBack(/home/audio/charles-bridge-cz)
exten => 201,1,Answer()
exten => 201,2,PlayBack(/home/audio/charles-bridge-es)
;--------Astronomical Clock--------
exten => 002,1,Answer()
exten => 002,2,PlayBack(/home/audio/astronomical-clock-en)
exten => 102,1,Answer()
exten => 102,2,PlayBack(/home/audio/astronomical-clock-cz)
exten => 202,1,Answer()
exten => 202,2,PlayBack(/home/audio/astronomical-clock-es)
;--------Sagrada Familia---------
exten => 003,1,Answer()
exten => 003,2,PlayBack(/home/audio/sagrada-familia-en)
exten => 103,1,Answer()
exten => 103,2,PlayBack(/home/audio/sagrada-familia-cz)
exten => 203,1,Answer()
exten => 203,2,PlayBack(/home/audio/sagrada-familia-es)
;----------------------
exten => s,3,Hangup()
```

Figure 3-23: *Extensions.conf* file in Asterisk server.

This file is composed by different steps when an extension is dialled. According to the Asterisk documentation this file is composed by:

```
exten => Name,Priority,Application()
```

Name determines the name or number of the extension dialled, *Priority* specifies the order of the steps, and *Application()* determines the function to apply.

The function *Answer()* allows Asterisk to accept the call. *PlayBack()* function is used to reproduce a recorded voice previously saved with specific characteristics[10]. The *Hangup()* function is to finish the call previously established.

According to this, in Figure 3-23, there are 3 examples exposed; Charles Bridge, Astronomical Clock and Sagrada Familia audio descriptions. Each one with 3 different languages: English, Czech and Spanish.

[10] Playback function only accepts audio WAV MONO files in 8KHz.

4 Verifications

In this chapter it is exposed different tests and analysis after development to verify the correct behaviour of the application. This chapter is really important to make a correct development and validate it.

4.1 Testing Features

Screen navigation:

The first step to test features is the screen navigation. It must be the same behaviour as previous defined in Chapter 3.2.2.

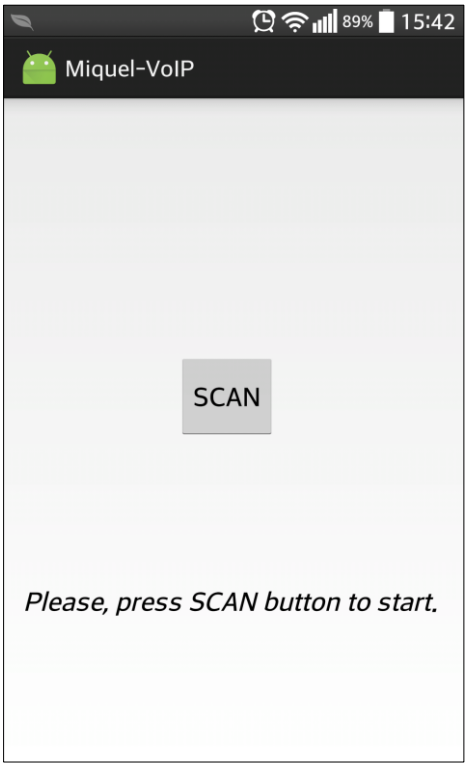

Figure 4-1: Main Activity (state 0)

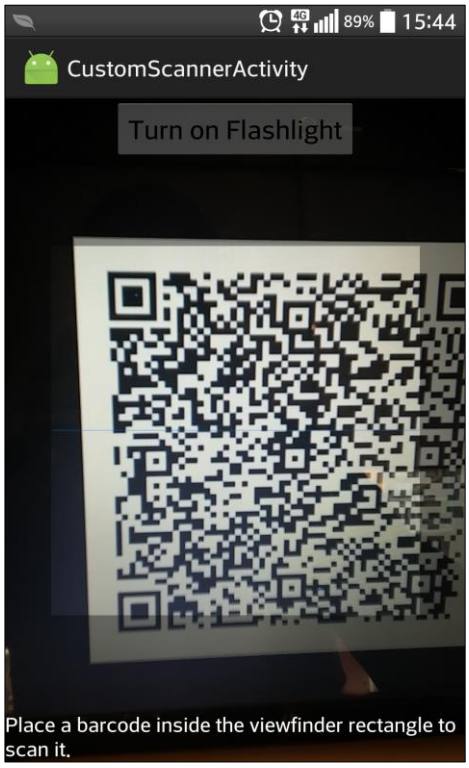

Figure 4-2: Camera Activity

As showed in the preview figures, the behaviour in these two activities is correct. When SCAN button is pressed, the Camera activity starts and it

tries to find a QR code. The Camera Activity allows the user to turn on the Flashlight if needed.

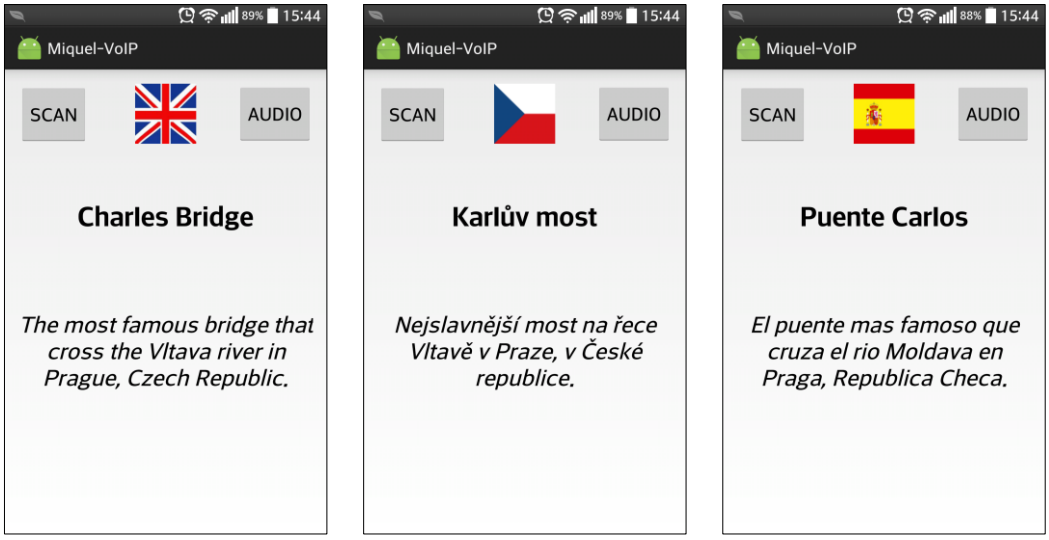

Figure 4-3: Main Activity (state 1)

When the correct QR code is captured it starts the Main Activity (state 1) in English language and connects to the specific Wi-Fi. When the user press the flag button the title and the description switch the language. It allows the user to press the button AUDIO to obtain the audio description or SCAN button to rescan another QR code.

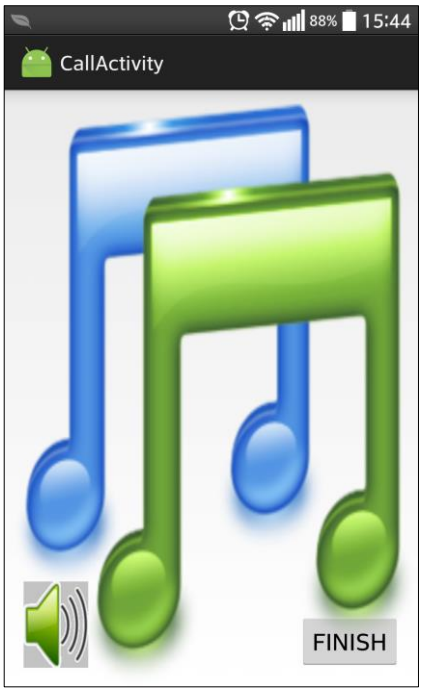

Figure 4-4: Audio Activity

When the user presses the *AUDIO* button it starts sending a registration to Asterisk and obtains the audio description with the language previously established. *Audio Activity* allows the user to enable the speaker phone or finish the call with different buttons. Once the Activities and its features have been tested in a real environment it can be determined that is well developed.

Asterisk features:

To verify the rest of the system it is important to analyse the behaviour of the Asterisk. For that reason, the Raspberry Pi needs to be connected with SSH connection with the next command:

```
sudo asterisk -r
```

This command shows the CLI[11] interface of Asterisk. Then, typing the command *"sip show peers"*, see Figure Figure 4-5, it shows how many SIP clients are connected to the server.

```
raspberrypi*CLI> sip show peers
Name/username              Host                                    Dyn Forcerport ACL Port     Status
1000/1000                  192.168.10.4                             D   N             57465    Unmonitored
1 sip peers [Monitored: 0 online, 0 offline Unmonitored: 1 online, 0 offline]
raspberrypi*CLI>
raspberrypi*CLI>
```

Figure 4-5: *"sip show peers"* command in Asterisk CLI

Once the user is well connected, it is the moment to try different extension calls. The image below, see Figure 4-6, shows the registration of the SIP Android client. In addition, there are two different test calls until the end; the Charles Bridge in English and the Astronomical clock in Spanish. Then, it is shown the un-registration when the user had closed the application.

To verify the correct functionality, all examples are also successfully tested with all possible languages.

[11] CLI: Command Line Interface. Is the console for Asterisk administrators.

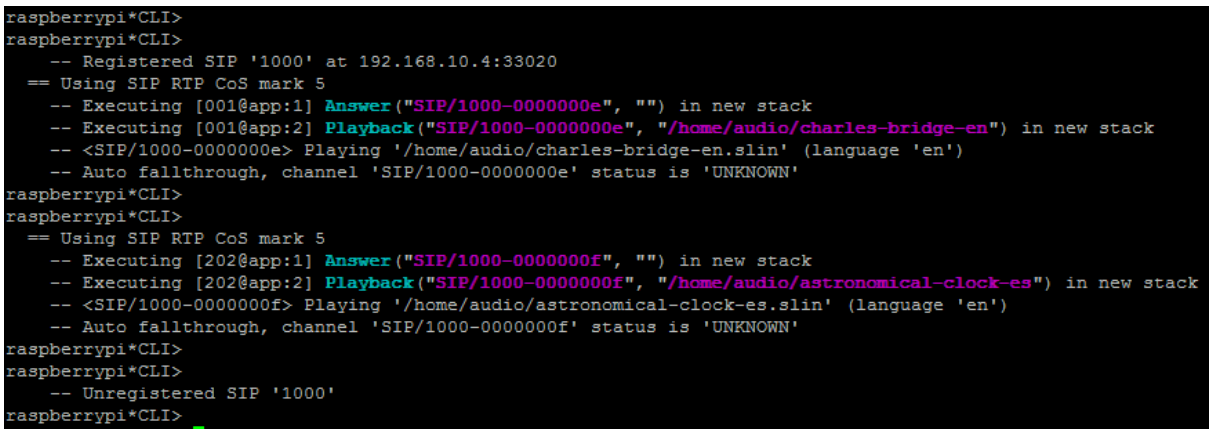

Figure 4-6: *"core set verbose 5"* command in Asterisk CLI.

Looking at these tests, it can be determined that Asterisk successfully runs according to the previous development.

4.2 Optimization of source code with Source Monitor

To optimize the source code of Android Application it is used *Source Monitor* program. This is a software metric program; it obtains statistics from source code and gives to the user relevant information about the optimization and quality of source code. Develop source code with this tool can reduce complexity, avoid problems and optimize the code.

Source Monitor analyses lots of parameters for each class. This program should not become an obsession to fulfil all its requirements because sometimes there are parameters that are not possible to correct. It depends on the source code. However, it is an appropriate tool that helps for a good development of codes. The different parameters are described below.

- ***Methods per class***. Quantifies the relation between methods and classes. Highlight that it is good to separate the methods in separated classes to avoid complexity.
- ***Average Statements per Method.*** Useful for a general feeling of how big each method is.

- ***Function Complexity***. Quantifies how complicated is the code according the cyclomatic complexity.
- ***Level of depth***. Quantifies how deep is the functions in methods, for example in chained functions.
- ***% comments***. It determines if there are enough comments in the code for a proper interpretation.

According to these parameters, Source Monitor creates a table with the results of analysis. The results below, see Figure 4-7, show the analysis of the classes developed in this project. There is no analysis for the integrated libraries (APIs). It is checked the MainActivity.java, CallActivity.java, WifiConnection.java and CustomScannerActivity.java.

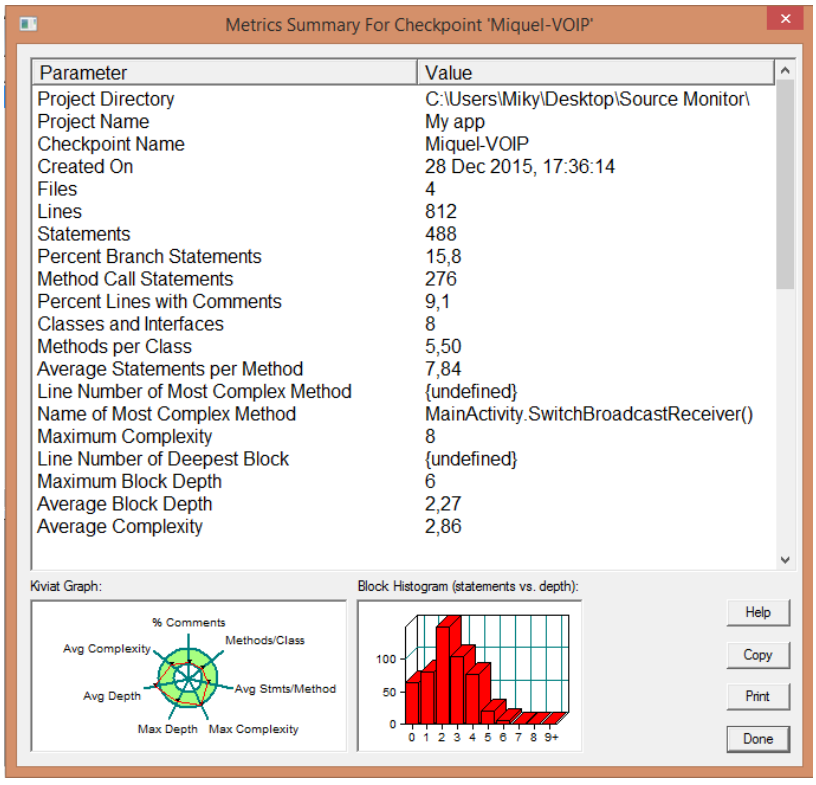

Figure 4-7: Metrics of source code using *Source Monitor program.*

Source Monitor can also generate a Kiviat graph with the analysis of Java source code in the Android application. See Figure 4-8.

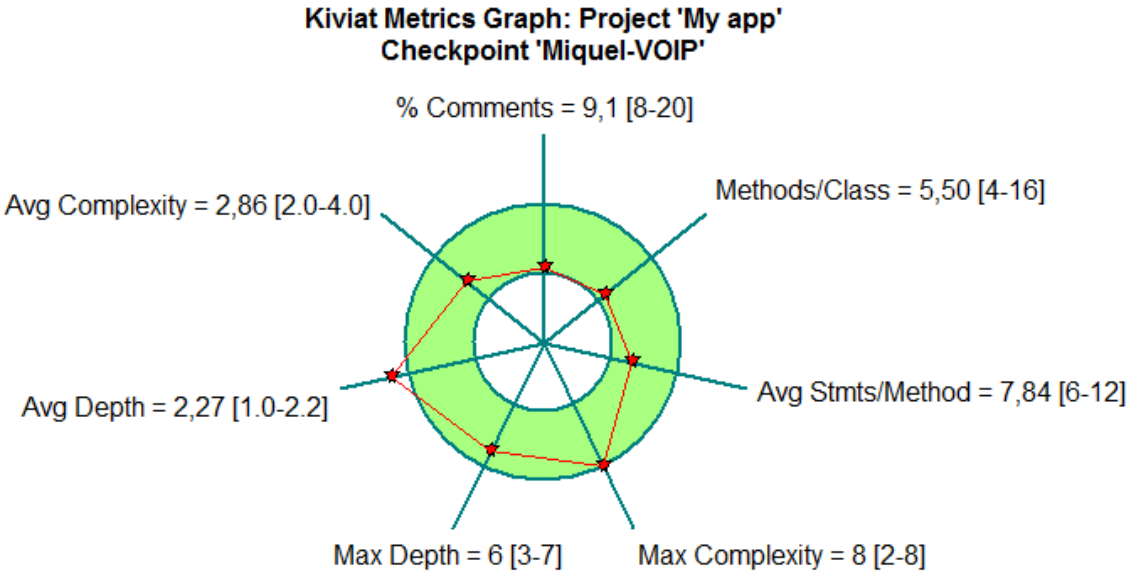

Figure 4-8: Kiviat graph from *Source Monitor* analysis.

The preview image shows that the developed code of the application (without the integrated libraries) is well developed because almost all the parameters are inside the green circle. The parameter "Avg Depth" is out of range in 0.07 points, which means a deviation of 5.8% and therefore it is not relevant for further discussion. In addition, this parameter is difficult to correct because it implies to modify all the methods in all the classes that has a depth code. According to these results, the source code from Android application is well developed with a good optimization.

4.3 Wireshark analysis

In this project it is important to analyse the communication between Android client and server. Wireshark is the most famous program to analyse the traffic in a network. It is used for network troubleshooting, analysis, software and communications protocol development, and education. It is the perfect program to analyse the communication of this project.

To capture the packets of the communication in the Asterisk server, it must be installed in the Raspberry Pi the program *tshark* and execute it with the following command:

```
sudo tshark -i eth0 -w /home/Test1.pcap -a duration:120
```

Using this command the Raspberry Pi will capture all packets in the interface *eth0* for 120 seconds and will save in /home/Test1.pcap. Once *tshark* is running, the Android user must register to Asterisk, make an extension call to obtain the audio description and close the application to un-register.

The three different extensions (three different languages) of Charles Bridge example are tested. The packets filtered are in the image below, see Figure 4-9.

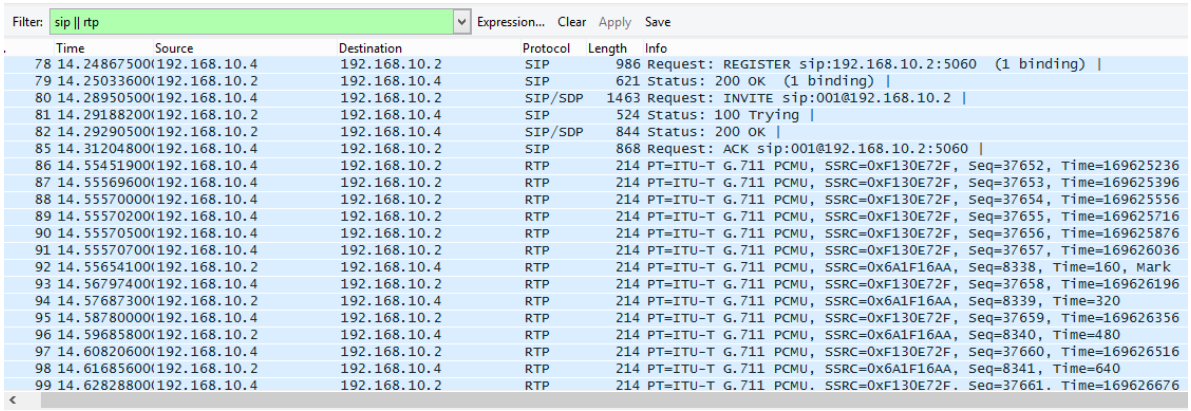

Figure 4-9: Filtered packets of Sip Dialog

It is used the filter "sip || rtp" protocols because these are the packets of the SIP dialog explained in Chapter 2.4.3. The great point of Wireshark is that have special functionalities to analyse VoIP and SIP connections.

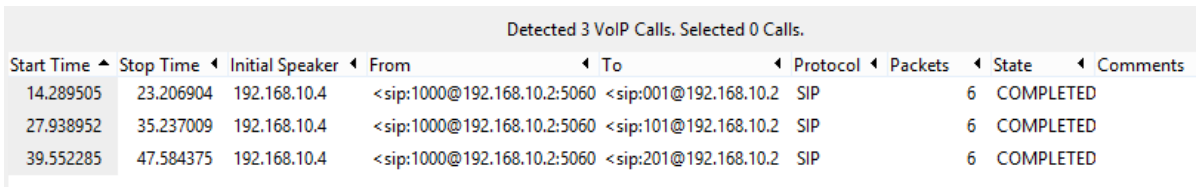

Figure 4-10: Three different calls identified by Wireshark.

Figure 4-10 shows one Wireshark functionality, it allows the user to identify the different VoIP calls in all the packets. Choosing the first call a little graph of the dialog will be shown, see Figure 4-11.

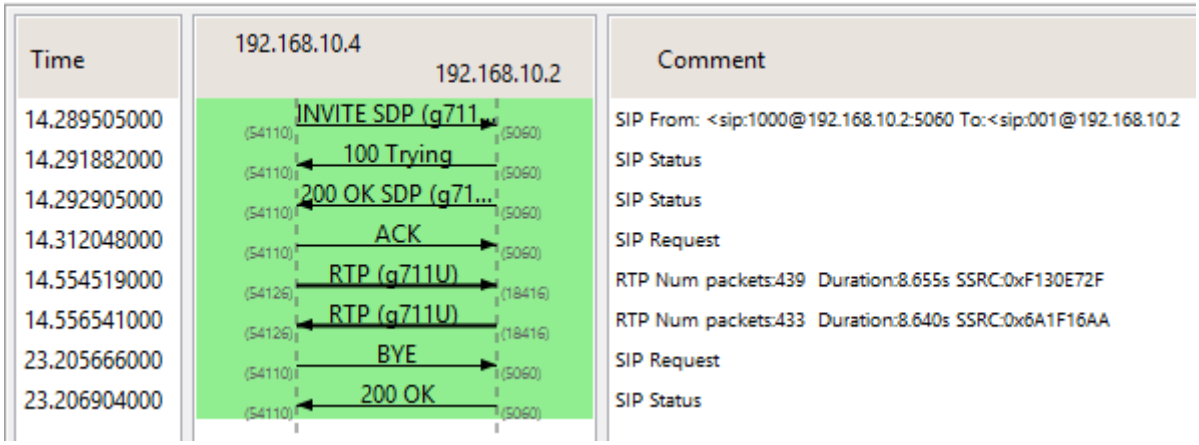

Figure 4-11: Real SIP dialog captured with Wireshark.

As Figure 4-11 shows, the user with IP 192.168.10.4 starts the dialog and makes the extension call to 192.168.10.2 which is the Asterisk server. Another functionality from Wireshark is the capacity of decode the RTP packets with the audio content.

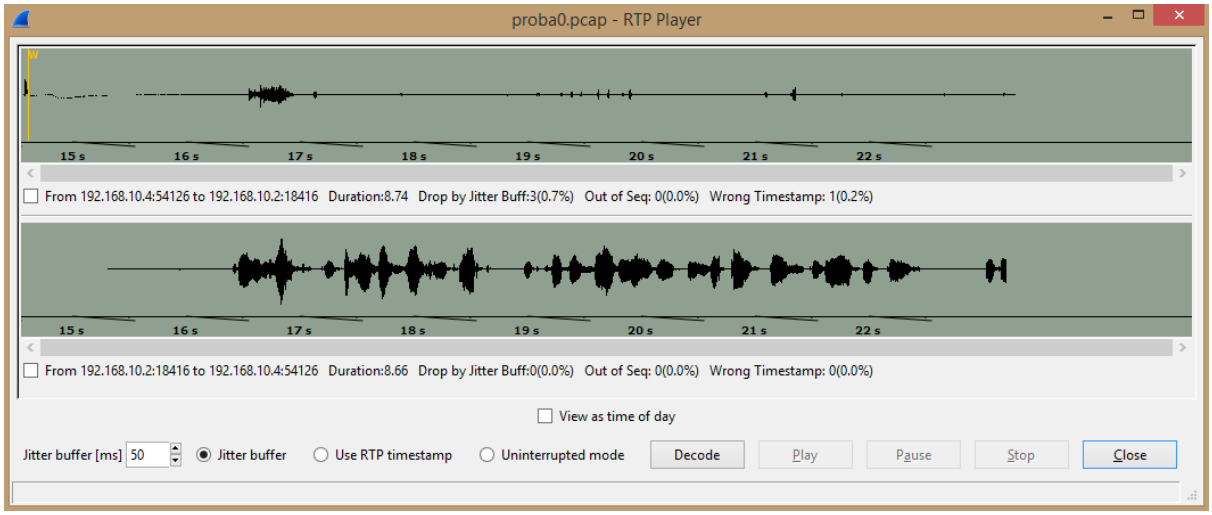

Figure 4-12: Decoded RTP packets with Wireshark.

The Figure 4-12 shows the decoded packets with the audio content and the user is able to reproduce it and listen to it. The problem observed is that the call established is bidirectional. The decoded packets shows two different audios, the audio description from server to client and the audio

recorded from user to server, as a normal call. However, the Asterisk server does not interpret the RTP packets received from the client, but this problem should be documented.

This problem implies that the double of bandwidth for each call is filled. Therefore, this problem will reduce the simultaneous user's capacity in the same network and increase power consumption of the smartphones.

This problem could be difficult to solve because it needs to modify the code from the *Doubango*'s library. It implies to code in low level and modify it enough to create a SIP dialog with only one way RTP call packets.

However, Asterisk server does not interpret the RTP packets from Android smartphones. It involves that the behaviour for the final user does not have any problems and it works as expected.

5 Conclusions

Once all development is finished and well tested, it can be concluded that this project achieves all the main objectives. Moreover, to obtain audio descriptions over VoIP has proven to be a cheap and good solution.

However, it is important to know how difficult the integration of a library is when the developer does not prepare the code for it. Fortunately, there is a lot of information and developer's communities on the internet to help and it is increasing, but sometimes it is not enough. Nevertheless, the technology changes really fast and in few years it may be some improved method to obtain audio descriptions.

Future lines:

It is relevant to highlight that this is a prototype solution that works fine, but has some limitations to establish it in a real environment. It should be adapted for more user capacity with a better hardware, and optimize the occupancy of bandwidth modifying the *Doubango*'s library. Other point to improve is the limitation that offers the 3 digits of *Extension to call*. One digit of *Prefix* and two digits of *Object ID* offer only ten different languages for one hundred possible audio-descriptions. This limitation could be solved increasing the digits of *Extension to call* and introducing more language characters in QR code until 7089 characters or a DB address where obtain all the languages. Moreover, there are a lot of users which use Bluetooth earphones and could be a new functionality to be introduced.

In Addition, according to Chapter 3.2.4 it is possible that few Android versions (almost 6% of Android devices) could not run this application because ZXing API use some functionalities that were integrated from Android 3.0 version onwards. To solve this it is recommended to update to the latest version or change the APIs to a low level APIs.

To conclude, nowadays the world needs audio description in many situations and it seems that this trend will not be reduced but increasing. For that reason this project opens a range of possibilities to obtain audio descriptions in an easy and cheap method to help people.

6 Bibliography

[1] Jim V. M., Leif M. and Jared S. *Asterisk: The future of Telephony*. Published by O'Reilly Media (2005). ISBN: 978-0-596-00962-5

[2] *ISO/IEC.* Article *18004:2015* 3th edition. Published in 2015-02-01.

[3] *Denso ADC. QR Code Essentials.* 2011. (accessed October 2015) http://www.nacs.org/LinkClick.aspx?fileticket=D1FpVAvvJuo%3D&tabid=1426&mid=4802

[4] *Asterisk.* 2015. http://www.voip-info.org/wiki/view/Asterisk

[5] Henning S. Session Initiate Protocol (SIP). May 2001. http://www.cs.columbia.edu/~hgs/teaching/ais/slides/2003/sip_long.pdf

[6] Global market share held by the leading smartphone operating systems by Statista. (accessed October 2015) http://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/

[7] Joshua B. *How to Design a Good API and Why it Matters.* Published 2007. http://lcsd05.cs.tamu.edu/slides/keynote.pdf

[8] Official guide for Android developers. *Session Initiation Protocol* https://developer.android.com/guide/topics/connectivity/sip.html (accessed October 2015)

[9] Official website of Asterisk. http://www.asterisk.org/ (accessed September 2015)

[10] Opeyemi O. *SIP on Android.* 4 June 2014 (accessed September 2015) http://obem.be/2014/06/04/sip-on-android.html

[11] Wahid G. *Integrating ZXing in your Android App as standalone scanner.* 21 February 2013 (accessed September 2015) http://blog.dihaw.com/integrating-zxing-in-your-android-app-as-standalone-scanner/

7 Annex

One Compact Disc with all source code of Android application and the different APIs with their documentation will be attached. In addition, it will be attached the configuration files of Asterisk server, the audio description files and the file from captured packets of network analysis.

The files are distributed according to the following scheme:

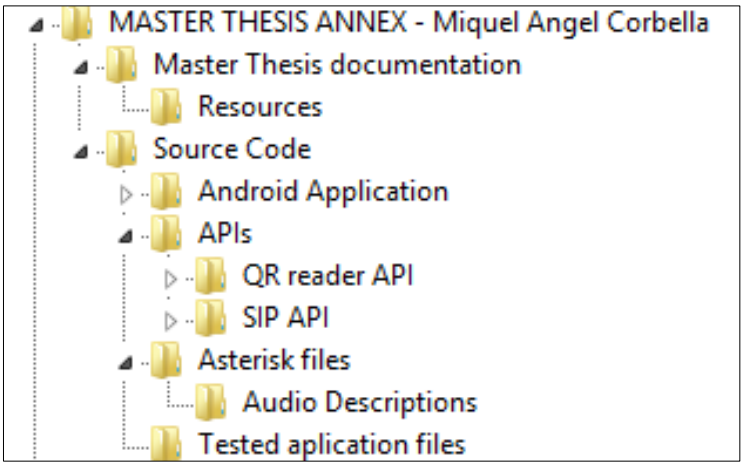

Figure 7-1: Structure of files attached.

Master Thesis documentation folder contains this documentation; inside this folder there is *Resources* folder which contains the book *Asterisk: The future of Telephony* [1]. In addition, in *Source Code* folder there are all the files needed to develop this project. *Android Application* folder contains all the files developed with Android Studio and *Miquel-VOIP.apk* application file ready to install in Android devices.

Moreover, *APIs* folder contains the two different APIs integrated in the Android application; *QR reader API* and *SIP API*. Furthermore, *Asterisk files* folder contains the configuration files *sip.conf* and *extensions.conf*, there are also all the audio descriptions tested for this project. The last folder is Tested application files which contains the Wireshark packets captured explained in Chapter 4.3.

In addition, the next images show the three different examples used for this prototype with the QR code and their descriptions.

01;Charles Bridge;Karlův most;Puente Carlos;The most famous bridge that cross the Vltava river in Prague, Czech Republic.;Nejslavnější most na řece Vltavě v Praze, v České republice.;El puente más famoso que cruza el rio Moldava en Praga, Republica Checa.;Miquel-VOIP;12345678;10.2

02;Astronomical Clock;Staroměstský orloj;Reloj Astronómico; Is a medieval astronomical clock located in Prague, the capital of the Czech Republic.; Je to středověký orloj v Praze, hlavním městě české republiky.;Es un reloj astronómico medieval localizado en Praga, la capital de la República Checa.;Miquel-VOIP;12345678;10.2

03;Sagrada Familia;Sagrada Familia;Sagrada Familia; Is a large Roman Catholic church in Barcelona, Spain.;Je to velký románský kostel v Barceloně ve španělsku.;Es una gran iglesia católica romana en Barcelona, España.;Miquel-VOIP;12345678;10.2