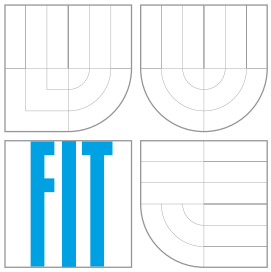
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

AUDIO STREAMING FOR ANDROID DEVICES USING UPNP

SDÍLENÍ AUDIA POMOCÍ UPNP PRE ANDROID

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — GABRIEL LEHOCKÝ
AUTHOR

VEDOUCÍ PRÁCE — Ing. VÍTĚZSLAV BERAN, Ph.D.
SUPERVISOR

BRNO 2014

Abstrakt

Obrovské množství lídí v současnosti používa své smartfóny jako mobilní hudební přehrávač, kde si ukladá svou oblíbenou hudbu. Chceme-li při poslechu zvýšit hudební žážitek a chceme-li použít kvalitní externí soustavu reproduktorů, použitím wireless streamovacího protokolu UPnP a RTSP můžeme to jednodušše docílit. Tyhle protokoly umožní jednoduché sdílení hudby, a jsou již implementovány v převažne většine audio-video zařízení. Táto práce si klade za cíl popis tychto technologii, na základe čeho popisuje i vývoj a realizaci jednoduché aplikace na sdílení audia.

Abstract

There is a huge number of people using their smartphones as portable music-players, with their favourite music stored on the device. To improve the music listening experience, a louder and better quality loudspeaker is needed. Using UPnP and RTSP streaming, wireless streaming of the music becomes possible. These protocols are implemented in high variety of devices, thus the user can share the music track to many devices easily. This thesis describes the technologies and possibilities in this kind of audio sharing and also describes a simple application that provides this kind of audio streaming.

Klíčová slova

Android, UPnP, aplikácia, audio, hudba, sdílení, streamování, multimédiá, mobilní telefón, smartfón, RTSP

Keywords

Android, UPnP, mobile application, audio, music, streaming, sharing, multimedia, mobile phone, smartphone, RTSP

Citace

Gabriel Lehocký: Audio streaming for Android devices using UPnP, bakalářská práce, Brno, FIT VUT v Brně, 2014

Audio streaming for Android devices using UPnP

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Vítězslava Berana, Ph.D.

.....
Gabriel Lehocký
July 20, 2014

Contents

Chapter 1

Introduction

Portable devices are very popular nowadays. One of the most used ones are smartphones with the Android operating system [14]. These phones are well equipped with advanced features, so except of making phone calls, users can use them for high variety of tasks, e.g., web browsing, photographing, gaming or listening to music [31].

Android offers an online store (Google Play) to download free applications, and with that, to extend the possibilities provided by the device. On the actual market there are hundreds of advanced music players for Android with millions of downloads together, but one can use the built-in music player which is equipped with many functions. Therefore, even this default music player is used by high percentage of users. Listening to music on a smartphone is very popular and many Android users do it every day. Users have their favourite tracks stored in the device's storage, and have playlists created by their taste of music [3].

People usually use their device to listen to music alone, with headphones on. For this purpose every available music player is suitable. To listen to the music in better quality, and louder, smartphone speakers or a headphones are not an optimal choice for a true music lover [16]. To connect the device to a better quality audio system, the easiest and most common way is the 3.5mm jack cable. Apart from that, there are plenty of wireless technologies also (section 2.1) that provide more free connectivity.

The Universal Plug and Play (UPnP) protocol is implemented in high range of network capable media-related devices. UPnP is one of the most used technologies for media streaming in local network [24].

There are already applications on the Google Play store solving the problem of streaming our favourite music to an audio system using Universal Plug and Play protocol (section 2.6). However, this solution tries to do it in a different way, where the user doesn't need to learn to work with a new music player application, but can use any media player to play the music. This solution also tries to make possible to stream other audio streams, like phone call, game sounds, audio of a video and other sources.

Chapter 2

Theory

This chapter describes the technologies and protocols that are used within this thesis. First, it discusses some technologies which can be used for audio streaming. Furthermore it describes the Universal Plug and Play protocol and its functionality in more details. In this section there are also information about Android and Real-time Transfer Protocol which are used for audio streaming on the network by the created application 4.5. The last section of this chapter describes some existing applications that provide audio sharing using the UPnP protocol.

2.1 Wireless streaming technologies

There are some technologies and standards which make wireless media streaming possible. The four most common media steaming technologies are Bluetooth, AirPlay, Miracast and DLNA [6, 19, 24].The following subsections describe the basics of these technologies.

Bluetooth

Only a few technologies exists that succeeded to unificate the whole market, but Bluetooth did it. Bluetooth was created by Ericsson in 1994 to make a RS-232 like communication wireless [5]. The most common use of this technology today is telephoning and audio listening.

Setting up a Bluetooth communication is relatively easy. To connect two devices, only pairing them is needed. Usually this pairing process requires a four-digit password to create the communication channel [6]. The data transfer between the devices is beamed directly without the need of a router or other intermediary.

Bluetooth uses protocols, called profiles, that interpret different kinds of communications. There is a big number of Bluetooth profiles describing many different types of communication. From these profiles, two provide audio streaming. These two are the Headset Profile and the Advanced Audio Distribution Profile [6]. To create a functioning communication, both end-point-devices must have the same profile implemented.

The Headset Profile (HSP) provides support for mobile devices to connect with a hands-free device [6]. The audio is encoded in 64 kbit/s CVSD (Continuously variable slope delta modulation) or PCM (Pulse-code modulation). It also implements minimal controls like the ability to ring, to adjust the volume or to pick up the call and to hang up.

For audio streaming an A2DP (Advanced Audio Distribution Profile) is used that relies on the GAVDP (Generic Audio/Video Distribution Profile). This profile creates a one-way stereo-quality pipeline between a source (smartphone, tablet, media player, laptop, etc.) and sink (wireless speaker system, AV receiver, etc.) [6]. The data channel of the A2DP profile is 721 kbps wide and it supports many different audio codecs like mp3, wma, aac, and others [30].

Figure 2.1: Music streaming from a smartphone to the headphones over Bluetooth.

Todays most preferred codec for audio streaming over Bluetooth is aptX, which can deliver near-CD-quality streaming [6]. The support of the aptX codec is good in the new devices however still not all of them supports this codec. The big advantage of the aptX, is the high tolerance to bit errors, and the quick re-establishing feature after a small dropout in the communication. The end-to-end latency of the aptX is 32ms, while the latency of the standard Bluetooth stereo is around 150ms.

AirPlay

AirPlay (originally called AirTunes, when it only supported audio) is *Apple*'s wireless display standard. It allows to stream audio and video from an *iPhone*, *iPad*, or *Mac* to an *Apple TV* device [6, 19].

AirPlay can work two different ways. It is able to use it in a display and audio mirroring mode, just like in a standard media streaming mode. *Apple*'s technology is good enough to stream and share the right content the user wants to [19].

In the most common implementation, the AirPlay-capable devices are connected to the local network via WiFi or Ethernet [6]. Therefore the quality of the streaming from the *iOS* or *Mac* device, depends on the quality of the network. AirPlay streams the music

in lossless ALAC (Apple Lossless Audo Codec) format. This means that the compressed audio is streamed without any further loss, but the better quality audio is down-sampled to 44.1 kHz.

AirPlay protocol stack uses UDP for streaming media, and is based on the RTSP (specified in chapter 2.5) network control protocol [12]. AES encryption is used for the two-channel audio. This encryption requires the receiver to access the appropriate private key to decrypt the streams. The stream is buffered for approximately 2 seconds before playback begins, resulting in a small delay before media is played [2].

To prevent sound quality deterioriaton due to reduction in bit depth and sound quality, AirPlay protocol uses metadata packets that contain the final output volume [32]. The audio data is streamed unprocessed at its original full volume, and the volume metadata is used by the end-point to set the volume. AirPlay also makes possible to stream one source to different end-points, each with its own volume control.

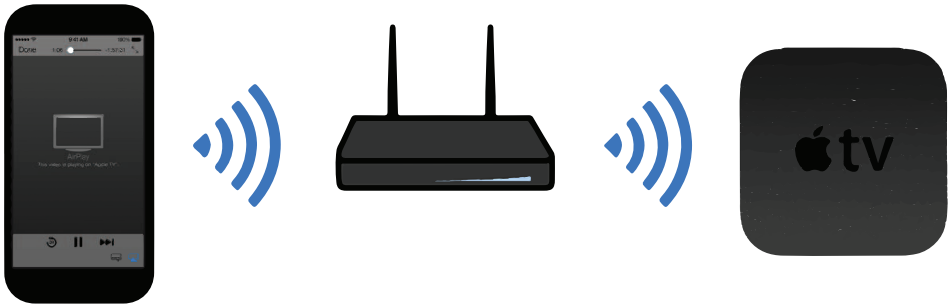


Figure 2.2: Video played from iPhone on an AppleTV using AirPlay.

As also view by the users, we can consider AirPlay a good functioning standard for media streaming, but it also has one big disadvantage. It only works with *Apple* devices [19]. However, more and more television and AVR receiver manufacturers implement this standard in their newly released devices for the *Apple* device users, so they can use this type of media streaming.

WiDi/Miracast

WiDi (Wireless Display) is a standard created by *Intel* [19]. With this technology Intel tries to compete against Apple's AirPlay.

To share media content using WiDi, there is no need for a connection to a local network such in case of AirPlay or UPnP [24]. The connection is direct between the two devices, which use WiFi Direct connection. This technology is integrated to all new *Intel*-based PCs and into TVs from manufacturers as *Samsung*, *LG*, and *Toshiba*. WiDi, just like AirPlay, allows screen and audio mirroring, and also individual media sharing. It also allows to show different content on the PC and TV display at the same time.

WiDi supports up to FullHD video streaming with 60fps in H.264 that is hardware encoded [26]. As for audio, it is possible to stream a 6 channel (5.1) in 16-bit/48 kHz LPCM in AAC or AC3 format. The latency of the streaming is less then 150ms.

Miracast is considered to be *Android*'s answer to *Apple*'s Airplay, and it uses WiFi Direct connection just like WiDi does [24]. All smartphones and tablets with Android 4.2 and later are Miracast-enabled. The newest version of WiDi and Miracast are compatible with each other, so the two technologies are merged together.

Figure 2.3: Screen mirroring from an Android device to TV using Miracast.

The Miracast was created to be an industry-wide standard, but in practice it didn't worked out well [19]. The biggest problem is, that for the manufacturers it is not mandatory to brand the products with the „Miracast" brand. As a result different manufacturers call their implementation of Miracast differently. For example *Sony* calls it „*screen mirroring*", *Samsung* calls it „*AllShare Cast*", *Panasonic* calls it „*display mirroring*", and *LG* calls their Miracast support „*SmartShare*". The manufacturers not only name the technology differently, but there are also differences in the implementations. These differences make theoretically compatible Miracast devices incompatible with each other, so the standard seems to have collapsed in practice.

UPnP/DLNA

DLNA (Digital Living Network Alliance) is a non-profit company that was crated by *Sony* to standardize media transmission and streaming on a local network [24]. The DLNA standard uses UPnP (described in chapter 2.2) as it's communication protocol. It is considered to be the most common way of media streaming over the local network [6]. This standard allows to share media in an IP-based network, just like in case of the AirPlay. All DLNA certified products that are connected to the same network are compatible with each other. There are many types of devices with various roles that can be part of the DLNA topology. The three main components of the communication are the media server, the media player and the media renderer. (A more detailed specification is in chapter 2.3, part Device Classification.)

Playback quality depends on many things: the quality of the source files, the quality of the video processing within the player, and the speed and reliability of the network. The positive side of the DLNA is the open architecture, that allows to stream high variety of media formats even in lossless quality [6]. The playback ability of the given format depends on the player itself, and whether it supports such formats. File compatibility varies per manufacturer of the player. Some choose to support a wide variety of file types while others only support the basics required by DLNA. (More in chapter 2.3, part Media Formats)

The biggest difference between DLNA and AirPlay/Miracast is, that DLNA is not a wireless

display solution [19]. It does not allow content sharing the way AirPlay and Miracast does. It is designed to stream local media files from the media server to the player. The standard also implements media description and control function like *play*, *pause*, *next*, *previous*, *stop*, etc.. The playback control is possible on the media player side of the communication.

On the actual market there are millions of DLNA certified devices, so this technology is easily accessible [6]. Nearly every network capable multi-media device (TV, Blu-Ray, AVR, etc.) has the DLNA certification. The UpnP protocol allows to create the connection between the devices easily. The devices automatically advertise themselves on the local network. This makes them easy to access.

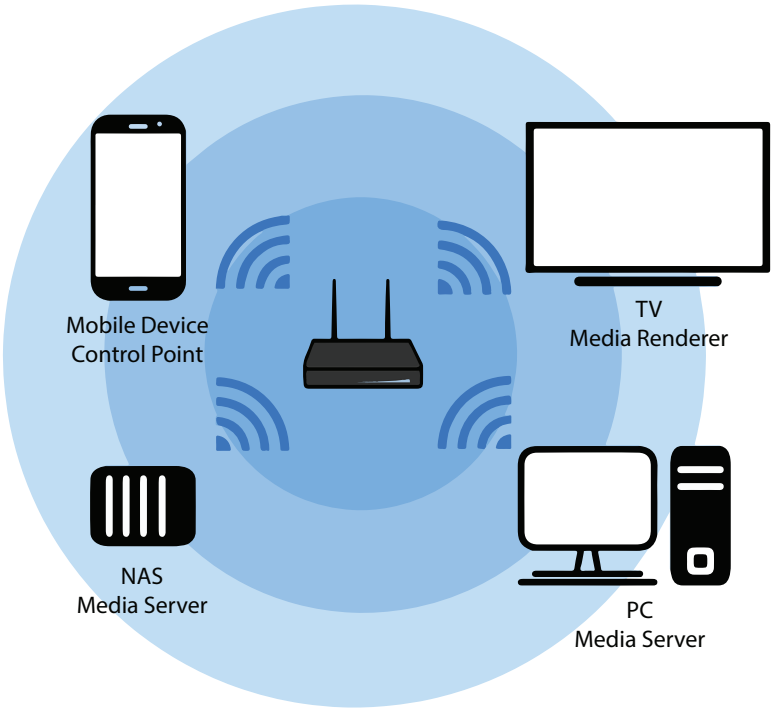


Figure 2.4: Different devices with various roles in one network.

2.2 UPnP Device Architecture

UPnP is a technology, that defines an architecture of a peer-to-peer network connection for intelligent appliances, wireless devices, and PCs of all forms [4]. The protocol was designed to be an easy-to-use and flexible standard connectivity in any kinds of networks, whether in the home, in a small business, or in public spaces. UPnP provides open networking and distributed architecture for TCP/IP networks that allows data transfer and transfer control functionality.

UPnP has a zero-configuration feature with automatic discovery of various devices on the same network. The UPnP devices can dynamically connect to the network, advertise their capabilities and ask for presence and capabilities of other devices. The devices can also

disconnect from the network easily and smoothly without leaving any unwanted state behind [4].

The UPnP Device Architecture contains protocols for communication between controllers, or control points and devices. For discovery, description, control, eventing, and presentation, the UPnP Device Architecture uses the protocol stack showed in Figure 2.5.

The UPnP Architecture uses both TCP and UDP transport protocols [4]. UDP is used for SSDP (Simple Service Discovery Protocol) and for multicast events. TCP is used for HTTP communication which delivers SOAP (Simple Object Access Protocol) and GENA (General Event Notification Architecture) messages.

The UPnP architecture defines two general classifications of devices: controlled devices (or simply „devices“), and control points [4]. The controlled device functions in a role of a server, that responds to the requests from the control points. Both control points and controlled devices can be implemented on a variety of platforms including personal computers and embedded systems. On a single network, multiple devices and control points can be operational at the same time.

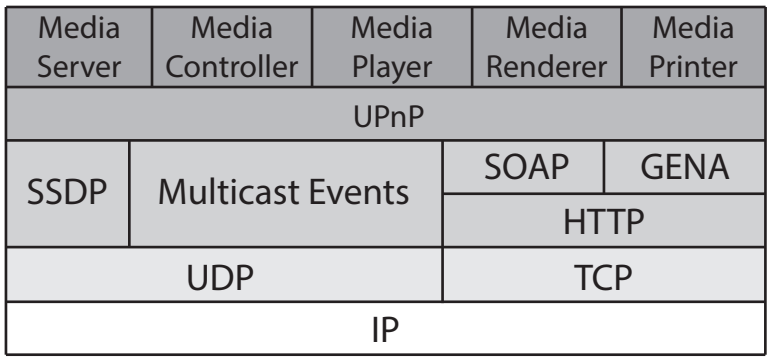


Figure 2.5: UPnP protocol stack.

Discovery

Discovery is the first step in UPnP networking [4]. The UPnP discovery protocol allows the device to advertise itself to other devices on thelocal network when the device is connected to it. This discovery protocol also allows the device to search for devices on the network. The discovery message contains some essential information about the services provided by the device. It also contains information such as the type of the device, the universal unique identifier, the pointer to more detailed informations and it also can contain the identification of the current state of the device.

Description

After a control point has discovered a device, the control point still knows only the basic provided information about the device [4]. The control point retrieves additional information from the URL that is provided in the discover message, to learn more features of the other device. The description of the device has two logical parts. The first part describes

the physical and the logical containers of the device, while the second describes the service capabilities of the it. The UPnP device description also includes some vendor-specific information about the device. These information are for example the model name and number of the device, the serial number, the manufacturers name and the URL of the website of the device.

Control

After knowing all the information about the device, the control point can ask to invoke actions and receive responses to the actions from the device [4]. Invoking actions on the device works like a remote procedure call. The control point sends the action to the device. After the action was completed or failed, the device returns the result of the action.

The control messages are done by sending a SOAP request to the „control URL“ of the control point [18]. This control URL was specified in the description of the device. The performed action to the request depends on the profile of the device. For every device profile, different kinds of control requests can be sent, different methods can be called. These methods are also listed in the devices description.

Eventing

UPnP also implements a concept for so-called „state variables“ [18]. State variables are used to keep some form of the state of the UPnP devices. In case the state is changed, a new state is sent to all devices that are subscribed to the event. Devices can subscribe to events using the URL that can be found in the description of the devices.

There are two types of supported eventing [4]. The first is the unicast eventing, where the control point can subscribe to get state updates. The second is the multicast eventing. In this the variables can be defined as multicast events. In this case the event is sent over UDP to all the listening devices on the multicast address. Event notifications are sent by the GENA (General Event Notification Architecture).

Presentation

After the control point has discovered a device and retrieved a description of the device, the last step in UPnP networking is the presentation [4]. The control point can load the content/data into the player/renderer in case the device has the URL of the presentation. The player/renderer can allow the user to control the playback of the content/data, depending on the type of the data content. The level of the control of the playback depends on the content and the player/renderer device.

2.3 DLNA

The Digital Living Network Alliance was formed by *Sony* in 2003, to come up with a standard that all media-related consumer electronic devices can support [27]. The goal was

to make high range of network capable multimedia devices able to communicate with each other.

DLNA uses the UPnP protocol for communication [24]. It defines strict rules to unify the communication between network media devices. DLNA specifies all the necessary functions, an UPnP capable device should implement, to be a reliable and fully compatible with other network of devices.

The Digital Living Network Alliance certifies consumer electronic devices (TVs, smartphones, AV receivers, Blu-ray players, ...) in order to ensure the compatibility between different brands. This makes the manufacturers stick to the regulations made by the DLNA to make the device more salable on the market. Currently there are over 250 companies contributing to DLNA.

Device Classification

The DLNA specification is divided into 3 main groups based on the type of the device [27]. Home Network Devices, Mobile Handheld Devices, and Home Infrastructure Devices (for example routers that connect other devices). Within these groups, the devices can be further divided into several classes based on their role in the communication: [1]

Digital Media Server stores content and makes it available for the network Digital Media Players, Mobile Digital Media Players, Digital Media Renderers and Digital Media Printers. Some examples of digital media servers include PCs and network attached storage (NAS).

Digital Media Player finds content offered by the Media Servers, and provides playback and rendering capabilities. Digital Media Players are not visible to other devices on the network such as Media Controllers. Examples of a Digital Media Player are TVs, home theater systems, game consoles and handheld mobile devices like smartphones.

Digital Media Renderers are similar to Digital Media Players, but they render or play content received from a Media Server. Digital Media Renderers are unable to find content on the network, so they must be set up and controlled by a Media Controller. A device that is a combination of a Media Player and a Media Renderer, can either find a Media Server on its own or be controlled by an external Media Controller. Digital Media Renderers are TVs, AV receivers and remote speakers for music.

Digital Media Controller finds content provided by a Media Server and matches it to the rendering capabilities of a Media Renderer. It sets up the connections between the Server and the Renderer. An intelligent remote control is for example a Controller device, such as a tablet or smartphone.

Digital Media Printer products provide printing services to the DLNA home network.

Mobile Digital Media Server devices differ from the Digital Media Server devices in that they support formats more suitable for mobile devices. Smartphones and portable music player are typical examples of the Mobile Digital Media Servers.

[1] Information about the device classification was collected from the official Digital Living Network Alliance website (http://www.dlna.org/dlna-for-industry/certification/dlna-device-classes/digital-media-server)

Mobile Digital Media Player differs from the Digital Media Player in that they support formats more suitable for mobile devices. For example a multimedia teblet can be a Mobile Digital Media Player.

Mobile Digital Media Controller finds content offered by the Media Servers, and match it to the rendering capabilities of a the Media Renderer,and setting up the connections between the server and renderer. An example is an intelligent remote access device like a smartphone or tablet.

Mobile Digital Media Uploaders send content to the Media Servers with upload functionality. For example digital cameras, tablets and smartphones can be Media Uploaders.

Mobile Digital Media Downloader finds and downloads content exposed by the Media Servers and play the content locally on the device after download. For example a portable music player can behave this way.

Media Formats

A DLNA certified media server/renderer must support all of the file formats named in the following list [2]. The device can support other file formats also but these default ones must be supported. Most of the DLNA certified devices support more media formats than these basics.

Picture: JPEG (Optional: GIF, TIFF, PNG)

Audio: LCPM, MP3, AAC-LC (Optional: WMA9, AC-3, AAC, ATRAC3plus)

Video: MPEG2, AVC/H.264 (Optional: MPEG1, MPEG4, WMV9, HEVC H.265)

Print: HTML

When the player does not support a file format that it was ordered to play, the server can transcode it to a standard format [27]. The transcoding process takes a lot of power and it slows down the streaming.

2.4 Android

Android is a mobile operating system that is based on a modified version of Linux [20]. The Android project was found in 2003 by the *Android, Inc* to create a mobile platform operating system. In 2005 *Google* took over the Android project development.

The main advantage of Android is that is offers unified approach to application development [20]. With this approach, the developer only needs to to develop the application for the Android system, and it is able to run on a high variety of devices that has this OS. The OS allows to create high range of different application that can use different features provided by the Android system. This positive feature is the reason manufacturers use this

[2]The list of media format is form the official DLNA guidelines: http://www.dlna.org/dlna-for-industry/guidelines

operating system in their new devices, and today this OS owns over 44% of the mobile device market [23].

Android is an open source project, which is a great advantage for the manufacturers. This allows the free customization in the software and also has no fixed hardware configuration. Android itself supports the following features [20]:

- **Storage** - SQLite lightweight relational database
- **Connectivity** - GSM/EDGE, Bluetooth, WiFi, LTE, WiMAX ...
- **Messaging** - SMS and MMS
- **Web browser** - WebKit
- **Media support** - H.264, MPEG-4, AMR, AAC, MP3, JPEG, PNG, GIF ...
- **Hardware support** - Accelerometer, Camera, GPS, Proximity sensor ...
- **Multi touch screen**
- **Multitasking**
- **Flash support**
- **Tethering** - Internet sharing as a wireless hotspot

Architecture

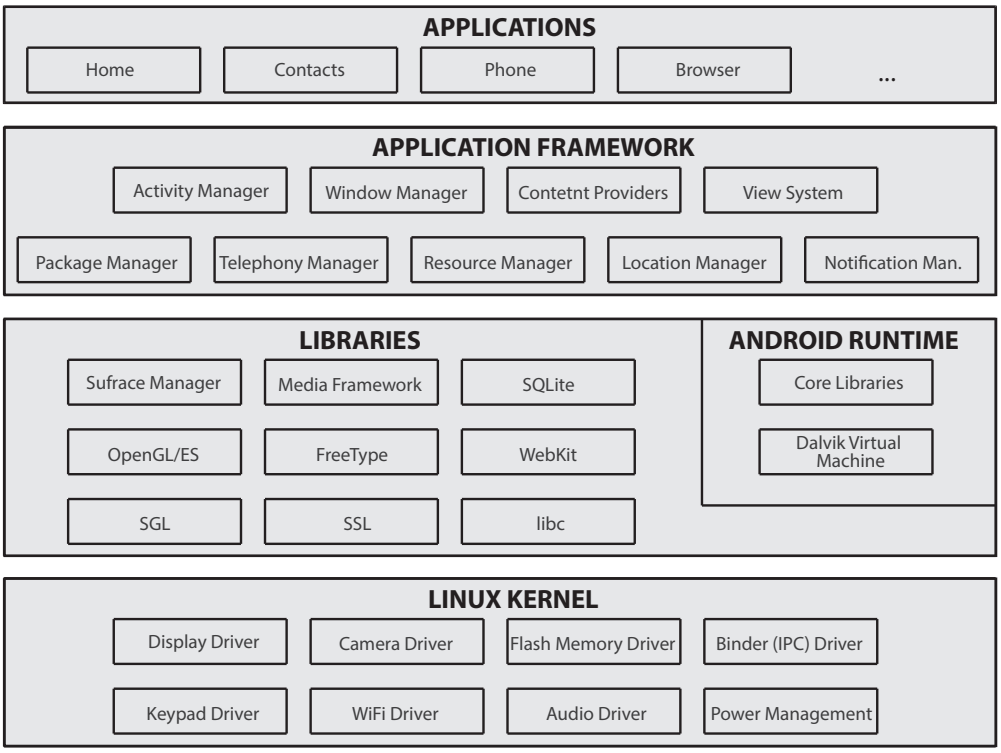


Figure 2.6: Android architecture diagram.

The Architecture of the Android operating system is divided into five sections in four main layers [20]:

- **Linux kernel** - The kernel is the basic level of the Android system. This kernel layer contains all the device drivers for all the hardware components of the given Android device.
- **Libraries** - The libraries provide the basic function of the Android.The SQLite library for example, provides database support so an application can use this kind of data storage. The WebKit library provides functionalities for web browsing.
- **Android runtime** - On the same level with the libraries is the Android Runtime. This provides a set of core libraries. These libraries implement the Java methods, so the higher level applications can be written in the Java language. The Dalvik virtual machine of this runtime enables for every single application to run its own process, with its own Dalvik virtual machine. All Android applications are compiled into a Dalvik executable format. This Dalvik Virtual Machine was designed and optimized specifically for the Android OS to be power efficient for the battery powered mobile devices with limited hardware specification.
- **Application framework** - This framework implements wide range of methods that can be used by the developers in the applications.
- **Applications** - The application layer contains all the applications that are installed on the Android device like Phone, Contacts, Web Browser, File Browser, etc..

Activity Lifecycle

The `Activity` is the main class of every Android application [20]. This class starts every application. The name of an `Activity` can be specified by the developer. Every single `Activity` that is implemented by the application, must be defined in the `AndroidManifest.xml`. This xml file is the main component of every single application. It contains the name and version of the application, list of the components (Activities), permissions, services and other properties that are used by the application.

An `Activity` is created by extending the `Activity` Java class of the ADT. This class implements a series of events that control the behavior of the `Activity` lifecycle (see Figure 2.7) [20].

It is not necessary to define all these methods for every `Activity`, but it is recommended if the application has non-traditional behavior [20].

Android Services

A service is an application in Android which runs in the background without needing to interact with the user [20]. The best example of the services is playing music in the background or logging the location using GPS.

By extending the `Service` base class of the Android library, a new service can be created [20]. Just like the `Activity` class, the `Service` has some methods we have to use to define its behavior.

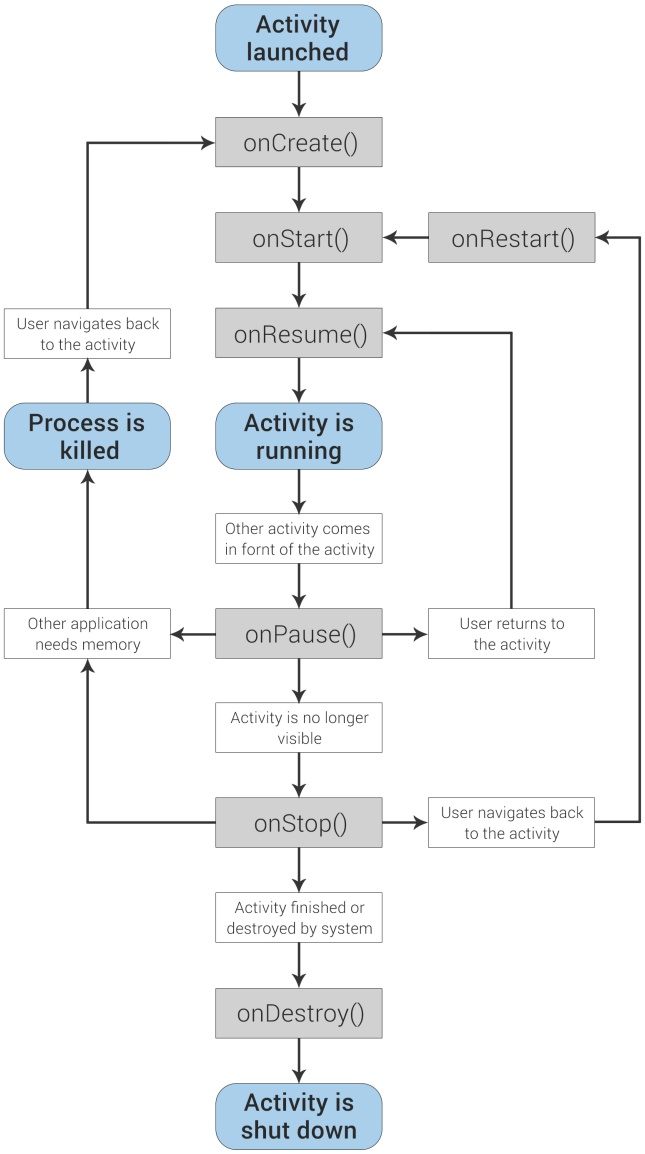


Figure 2.7: Activity lifecycle diagram.

The `onBind()` method makes binding possible by an activity to the service [20]. This in turn enables an activity to directly access members and methods inside a service.

The `onStartCommand()` method is called when the service is started [20]. This can be made by using the `startService()` method. This method signifies the start of the service, and this method contains the code of the behaviour of the service.

The `onDestroy()` method is called when the service is stopped using the `stopService()` method [20]. This is where the clean up of the resources are implemented, which are used by the service.

To make the services possible to run, all services that are started by the application must be declared in the `AndroidManifest.xml` file [20].

Androids Audio Related APIs

The Android library has many implemented classes which help us to create different multimedia applications. Some of the most significant ones are the following [8]:

- `AudioManager` provides access to volume and ringer mode control.
- `AudioRecord` class manages the audio resources to record audio from the audio input hardware of the platform.
- `AudioTrack` manages and plays a single audio resource.
- `MediaRecorder` is used to record audio and video from the input hardware of the platform
- `MediaPlayer` class can be used to control playback of audio/video files and streams.
- `RemoteController` is used to control media playback, display and update media metadata and playback status.
- `Ringtone` provides a quick method for playing a ringtone, notification, or other similar types of sounds.

Media Recorder

`MediaRecorder` class allow us to build out own audio recording functionality. It enables wide flexibility, such as controlling the length or the quality of the recording [9].

The `MediaRecorder` class is used for both audio and video capture [15]. After constructing a `MediaRecorder` object, to capture audio, the `setAudioEncoder` and `setAudioSource` methods must be called. If these methods are not called, audio will not be recorded. (The same goes for video. If `setVideoEncoder` and `setVideoSource` methods are not called, video will not be recorded.) Other methods such as `setOutputFormat` allows us to choose what file format should be used for the recording and `setOutputFile` allows to set up the output file.

The `MediaRecorder` operates as a state machine. Figure 2.8 shows a diagram from the Android API reference page for `MediaRecorder`, which describes the various states and the methods that may be called from each state [15].

The following lists show us the possible options we can use for media recording connected with the codec and the source of the recording: [3]

`AudioSource`: The following options are available to set the audio recorder source [11]:

- `CAMCORDER`: Microphone audio source with same orientation as camera if available, the main device microphone otherwise
- `DEFAULT`: Default audio source
- `MIC`: Microphone audio source

[3] These lists are from the Android Developers website

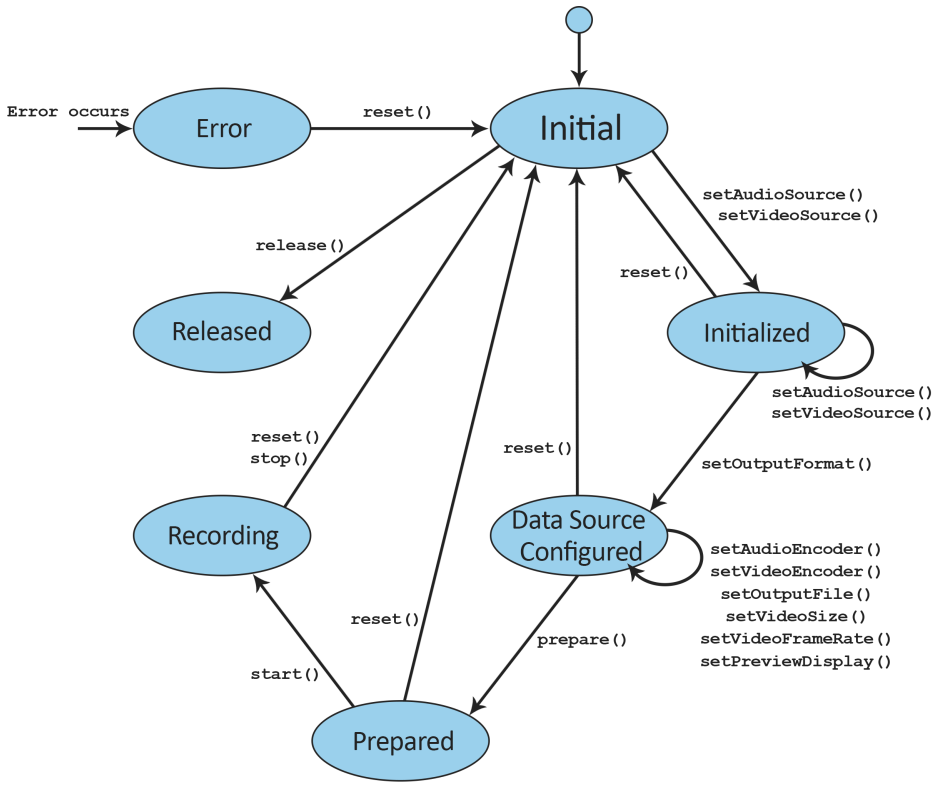


Figure 2.8: `MediaRecorder` state diagram from Android API reference.

- `REMOTE_SUBMIX`: Audio source for a submix of audio streams to be presented remotely. (Only from API level 19, Android 4.4)
- `VOICE_CALL`: Voice call uplink + downlink audio source
- `VOICE_COMMUNICATION`: Microphone audio source tuned for voice communications such as VoIP.
- `VOICE_DOWNLINK`: Voice call downlink (Rx) audio source
- `VOICE_RECOGNITION`:Microphone audio source tuned for voice recognition if available.
- `VOICE_UPLINK`: Voice call uplink (Tx) audio source

`AudioEncoder`: By the Android system, the following media codecs are supporte [10]:

- `AAC`: AAC Low Complexity (AAC-LC) audio codec
- `AAC_ELD`: Enhanced Low Delay AAC (AAC-ELD) audio codec
- `AMR_NB`: AMR (Narrowband) audio codec
- `AMR_WB`: AMR (Wideband) audio codec
- `DEFAULT`: Default audio codec
- `HE_AAC`: High Efficiency AAC (HE-AAC) audio codec (Only from API level 16, Android 4.1)

Native Development

Android NDK (Native Development Kit) is a set of tool for the developers, that allows to implement a part or a whole Android application in a native language like C, C++ or assembly [22]. Native application development may improve the performance, usually of a processor-bound application. Many multimedia applications and games use native code to improve the applications performance.

There are three reasons, that a native code improves the performance of an application [22]. While the Java code is compiled into a Java byte-code, and it is run by the Dalvik Virtual Machine, the native code is compiled into a binary code and it is run directly by the operating system. In the system, a Just-in-Time (JIT) compiler is added to the Dalvik Virtual Machine. The JIT compiler analyzes and optimizes the Java code before it's execution, and it makes the execution slower. This is the first reason the native code is faster in most of the cases than the Java code.

The second factor that can improve the performance of an application is, that the native code can access some lower level processor features [22]. These features are not accessible by the applications using only the Android SDK.

The third reason is the optimization possibility of the application on the assembly level [22]. This kind of optimization is also used in desktop software development.

Using NDK also have some cons [22]. Calling JNI (Java Native Interface) methods also take extra work for the Dalvik Virtual Machine. Also, the native code cannot be optimized by the JIT compiler just like the Java code. Developing with NDK does not guarantee the performance improvement of an application furthermore it can even harm the performance some times.

Rooting Android

Rooting an android device means that the user gets root permission for all the functions of the system [17]. Rooting an android device can give the opportunity to do much more than the phone can do out of the box, e.g., wireless tethering, speeding up the device with overclocking, or customizing the themes. With a rooted phone, certain applications can access system settings, as well as flash custom ROMs to the phone, which add all sorts of extra features. There are many different Android phones on the market so the method to root a device differs for each model.

Rooting a device brings many privileges but there are also disadvantages of having a rooted device.

Advantages [29, 28]:

- More applications: applications can control the system and use more features of it.
- Latest OS updates: the user can manually install any system updates, and always has the access to the newest versions provided by the Android developers.
- More customizations: the user interface is highly customizable.

- Speed or battery life boosts: Rooting allows the user to install custom kernels that are optimized for different power modes.

Disadvantages [28]:

- Rooting immediately voids the warranty of the device.
- Poor performance: Boosting up a performance of the phone should be an advantage. However, in many cases the phone loses it's performance and reliability.
- Viruses: The applications with root privilege can easily infect the system with any viruses.

The biggest risk of rooting a phone is that, there is always a small possibility with flashing. In rare cases the phone becomes unable to function, becomes „bricked“ [17].

2.5 Real-time Transport Protocol

Real-time Transport Protocol (RTP) is used to transmit digitalized audio or video signals over IP Internet [7]. RTP provides two key facilities, i.e., sequence number, timestamp, which allow the receiver to detect order of data loss and to control the playback. RTP is designed to carry wide variety of real-time data so it does not enforce a uniform interpretation of semantics. The fixed RTP header contains information about the payload, thus the receiver knows how to interpret the received data.

RTP only focuses on transporting content, however another aspect of real-time transmission is equally important: monitoring of the underlying network during the session and providing out of band communication between the endpoints [7]. Out of band mechanisms can be used for detecting and sending informations about playback delay, jitter changes or even for parallel information sending, like captions for a video stream. Real-time Control Protocol (RTCP) provides exactly these kind of data for the RTP communication. RTCP allows senders and receivers to transmit a series of reports to one another, that contain additional information about the transferred data and the performance of the network.

Real-time Streaming Protocol

Real-time Streaming Protocol (RTSP) defined in RFC 2326, is an application-level protocol, that enables control over the delivery of data with real-time properties over IP [33]. This protocol is designed to work with lower level protocols RTP and RTCP to provide complete streaming service over Internet. It works for a large audience multicast and for single-viewer unicast too.

RTSP provides remote control functionality for audio and video streams, such as pause, fast forward, reverse, and absolute positioning. Sources of data include both live data feeds and stored media files [25].

RTSP establishes and controls streams of continues audio and video media between the media server and the clients. A media server provides playback and recording function for the media streams, while the client requests these data. RTSP supports the following operations [33]:

Retrieval of media from the media server: The client can request a presentation description via HTTP or some other method.

Invitation of a media server to a conference: A media server can be invited to join an existing conference. After joining the conference, the media server can play or record the the media in a presentation.

Adding media to an existing presentation: The server and the client can notify each other about any additional media becoming available.

Methods of the RTSP

The method tokens of this protocol indicate the action to be performed on the other side of the communication. The following list summarizes all the methods of the RTSP protocol. It also shows which request is sent by which participant of the communication, and the requirement of the implementation of the method [1, 33]:

`OPTIONS`: client to server / server to client, required
Information to the other party about the options that can be accepted.

`DESCRIBE`: client to server / server to client, recommended
Description of a presentation or media object identified by the request URL from the server.

`ANNOUNCE`: client to server / server to client, optional
From client to server it posts the description of the presentation or media. From server to client, it announces updates in the description of the session.

`SETUP`: client to server, required
Client asks the server to allocate resources for the stream and to start the RTSP session.

`PLAY`: client to server, required
Client asks to start sending data via the allocated `SETUP`

`PAUSE`: client to server, recommended
Client temporarily halts the stream without freeing the server resource associated with it.

`TEARDOWN`: client to server, required
The client ask the server to stop the stream with also freeing the resource associated with it.

`GET_PARAMETER`: client to server / server to client, optional
Retrieves value of the parameter of a presentation or a stream.

`SET_PARAMETER`: client to server / server to client, optional
Sets the value of the presentation or a stream.

`REDIRECT`: server to client, optional
The server informs the client that if must connect to an other server URL provided in the mandatory location header.

RECORD: client to server, optional
The client initiates recording a range of media data according to the presentation description.

If the server does not support a particular optional or recommended method, it must return `501 Not Implemented` answer, and the client should not try that request again [1].

2.6 Existing Android Applications Using UPnP

on the actual *Google Play* store, there are a couple of applications that use UPnP communication architecture [21, 13]. These applications can be divided into three groups, based on their functions and role in UPnP communication.

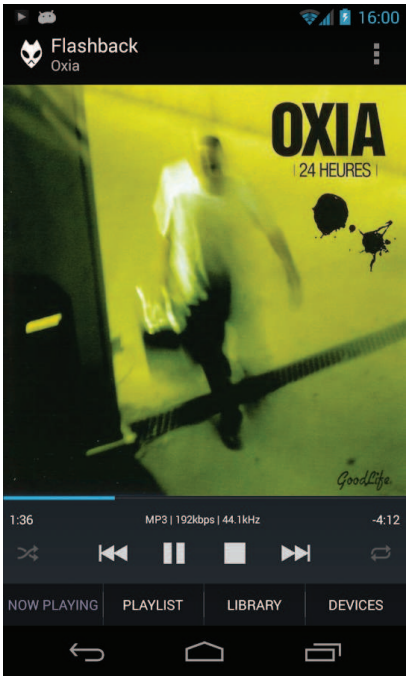


Figure 2.9:
`BubbleUPnP` music controller interface.
Source: play.google.com

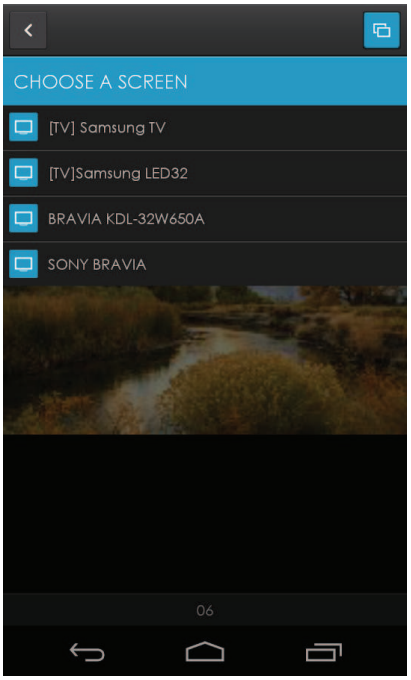


Figure 2.10:
`iMediaShare` media renderer selection screen.
Source: play.google.com

The first type of existing UPnP application is the media renderer (player) [21]. It is basically a media renderer application that allows only to play the media on the android device. These applications browse media servers and the playback is realized on the mobile device not on an other media renderer.

The second type is the UPnP remote control or control point application [21]. Remote control applications searches the local network for UPnP clients and servers. Their basic function is to browse for a media file on the media server and initialize the streaming to a media player. The application in this case is a remote controller which allows the user to control the streaming with functions like pause the playback of the video or changing to

next picture or music track. High number of UPnP remote control applications work also like a media renderer.

The third type of Android UPnP using applications are the media servers [13]. There are plenty UPnP server applications on the current Android market, that are capable to share the media, stored on in the memory of the hand-held device to a media renderer. These applications also have the functions to control the playback on the media renderer.

Features of some of the most used media server applications are the following [21, 13]:

BubbleUPnP by Bubblesoft (see Figure 2.9)

- Music and video streaming from a UPnP/DLNA Media Server to the Android device.
- Control audio playback of the android device from another UPnP Control point.
- Downloading media from the media server to the android device.

Flipps (fromer iMedia Share) by Flipps Media Inc. (see Figure 2.10)

- Accessing to over 100 online media channels (YouTube, Vimeo, BBC, CNN,...).
- Media playback controlling for media renderer.

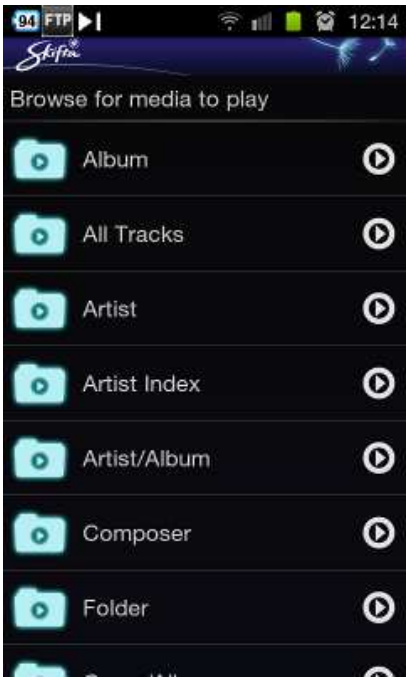


Figure 2.11:
`Skifta` media brower screen.
Source: play.google.com

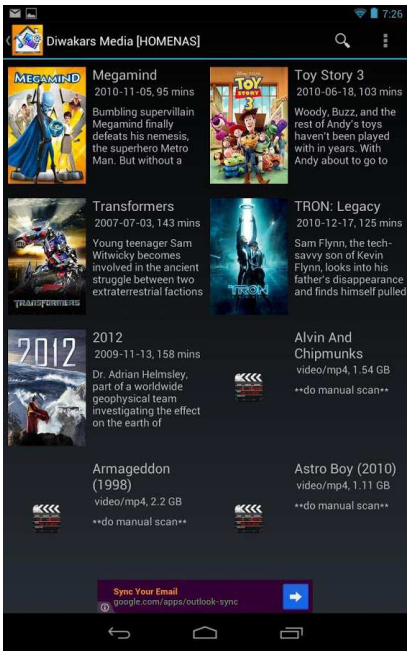


Figure 2.12:
`MediaHouse UPnP/DLNA Browser` browsing surface.
Source: [13]

Skifta by Qualcomm Atheros, Inc. (see Figure 2.11)

- DLNA Certified application
- Accessing media, stored on a media server and in the local storage.
- Media server - sharing the local media for media renderers.

- Controlling media playback of the media renderers.

UPnPlay by Bebopfreak

- Accessing and streaming music and video via media servers.
- Music and video streaming from a UPnP/DLNA Media Server to the Android device.
- Downloading media from the media server to the android device.

MediaHouse UPnP/DLNA Browser by Diwakar Bhatia (see Figure 2.12)

- Accessing and streaming music and video via media servers.
- Music and video streaming from a UPnP/DLNA Media Server to the Android device.
- Media playback controlling for media renderer.

Chapter 3

Application Concept

To create the basic concept of the application, first it is important to analyse what other existing applications offer (described in Chapter 2.6).

All existing media server applications work the same way in basics. Each of them allows the user to stream a local media file available on the device's storage or some of them to play an online media stream [13].

None of these actually available applications allow the user to stream a phone call, notifications and other voices (game music and fx, phone ringing, audio played by other application, etc.).

3.1 Basic Concept

To cope with the problems mentioned in the previous section, I designed an application to fill the gap of the missing features on the current market. The idea is, that the application streams all audio which would be heard through the device's speakers, just the way that a Bluetooth (Section 2.1) headphones works, but in this case using UPnP and RTSP streaming. Basically UPnP was not designed to work like Miracast (Section 2.1) or Apple's AirPlay (Section 2.1). It is not designed to share the device's screen or audio output to an other device, but also doesn't make it impossible.

So why UPnP? The first reason is that AirPlay is only available only for Apple devices [19]. The second is that Miracast is not universal and popular enough. It has the limitation that only devices by same manufacturers can work together [19]. The third reason is that, high number of TVs and AVRs with network connection feature are DLNA certified [24]. That makes them a fully functioning audio receiver. The last reason is that there are no applications yet which try to solve audio mirroring for Android devices using UPnP.

In order to create a widely compatible application, it is recommended to design it using the DLNA standard. This means, that the application should provide services that are fully understandable by most of the media related devices on the network [24]. For us, this mainly contains the media encoding we would like to stream, and the events (actions) that control the playback.

3.2 User Interface Concept

In Figure 3.1 there is an application screen that is designed to control the application named `AudioShare`. The four blue numbered frames are indicating the main features and functions that the application should provide. The following points define what these different elements are, and what functionality they provide.

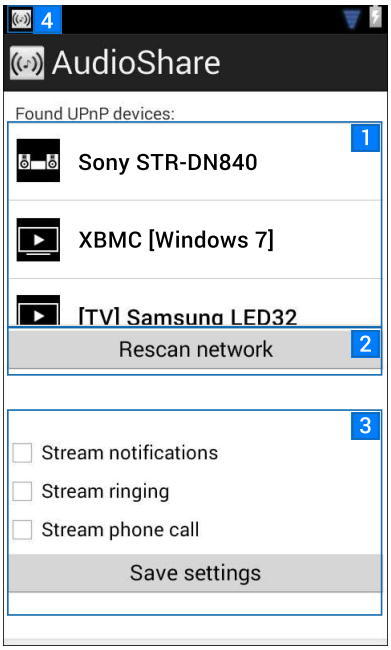


Figure 3.1: Application GUI concept.

1. This list shows the found UPnP players on the network. By clicking on a found device, the audio streaming automatically starts 4. The icon before the name of the device indicates the type of the device found (TV, AVR, HiFi, etc.). While the streaming is active this list becomes disabled.
2. The „Rescan network“ button has two functions. While looking for specific device, the user can make a new network scan for UPnP devices. When the audio streaming service is running, this button becomes a „Stop streaming“ button, that stops audio streaming.
3. This part of the GUI makes the user change basic settings about streaming. The user can change, what sources will be streamed.
4. When the streaming is running, an icon is showed on the Android status bar, thus the user knows that the application is running an the streaming is in process.

Chapter 4

Implementation

This chapter describes the techniques and tools used in the implemented application. The implementation started with two different applications to solve two basic problems.

The first application was designed to work with the audio sources of the android and to make them able to create stream the audio on the network. In Section 4.2 is described the solution of getting the the audio from different sources, and Section 4.3 explains the usage of a RTSP streaming library used in this application.

The second application solves the problem of the communication with the UPnP protocol. This implementation is described in Section 4.4. In the final application 4.5 this was extended with the functions of the first applications making the RTSP streaming.

4.1 Tools and Libraries

In this project, Android Developer Tools (ADT) [1] were used for implementation. ADT is an Eclipse [2] integrated development environment containing Android software development kit (SDK).

In this application, a library named `Cling` is used for UPnP communication on the network [3]. `Cling` is a UPnP library implemented in Java language. This application uses the `2.0-alpha3` version of `Cling` that was released on 22nd January, 2014. Compared to `Cling` version one, this second version in addition contains the implementation for Android platform. This library implements the UPnP discovery and description only. For further communication and control message sending to the device, the `cling-support-2.0-alpha3` library is also needed.

A library named `libstreaming` [4] is used for RTSP audio streaming. `Libstreaming` is a library, which allows to stream the camera and/or microphone of an android powered device using RTSP. It also supports AAC audio format for streaming, so all DLNA certified devices are able to play the audio provided in its stream.

[1] ADT is available on website: http://developer.android.com/sdk/index.html
[2] Eclipse official website: http://www.eclipse.org/
[3] The `Cling` library is available on: http://4thline.org/projects/cling/
[4] The `libstreaming` library is available on: https://github.com/fyhertz/libstreaming

4.2 Android Audio Manager and Media Recorder

To test the functions and possibilities provided by the Audio Manager implemented in the Andriod SDK, a simple application was created to test and learn the usage of the audio related APIs of the platform.

The audio source listening application uses the `AudioManage` class from Android libraries. This class allows to manage the audio related functions of the Android system.

This application also contains a service that listens if there is an audio that is produced by any application running on the device.

In Figure 4.1, there is the interface of the application named `VolumeSet`. It shows that the listening service is running, and a blue `S` letter is showed on the system's notification bar. This `S` means that there is currently no audio played on the device. In case any audio is played in the speakers, an `M` letter shows up on the notification bar.

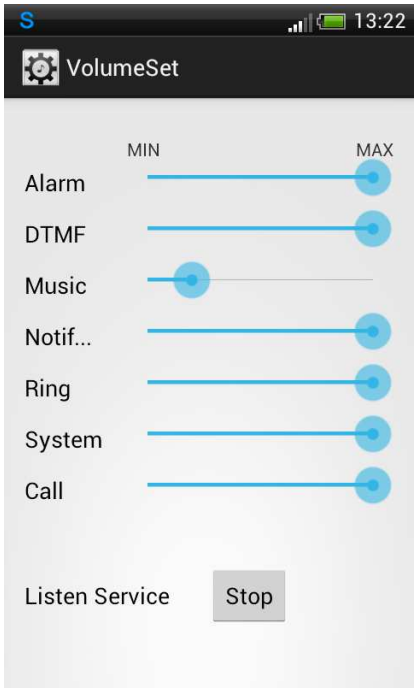


Figure 4.1: Audio manager application's interface.

After a long research (on different websites and in other sources), I realized that, to make the application,that makes it possible to get audio streams from different audio sources of the device, a native development is required and that this solution requires a rooted android device (more about rooted Android in Section 2.4).

During further studies and research, other possible solution was found. This solution uses the `MediaRecorder` class of the Android API. It is used by the `libstreaming` library (mentioned in Section 4.1), that became the part of this application.

The `MediaRecorder` API, records the audio to an output file set with the method `setOutputFile()`. This solution is a simple hack of this recorder, to write the recorded

data to a local socket instead of a file. The pseudo-code in listing 4.1 shows us how is the `MediaRecorder` output redirection implemented.

Listing 4.1: `MediaRecorder` setup pseudo-code

```
myLocalServerSocket = new LocalServerSocket(LOCAL_ADDR);

myReceiver = new LocalSocket();
myReceiver.connect( new LocalSocketAddress(LOCAL_ADDR));
// also setting up buffer size and timeout for the receiver

mySender = myLocalServerSocket.accept();
//also setting up buffer size for the sender

myMediaRecorder = new MediaRecorder();
// also setting up recorder source, codec and quality

myMediaRecorder.setOutputFile(mSender.getFileDescriptor());
```

To get the audio stream, we just have to read the data from the `mySender` socket. This is performed by the `RtspServer` service implemented by the `libstreaming` library, that is described in Section 4.3.

4.3 Streaming with RTSP

The RTSP communicarion is implemented in the `libstreaming` library (mentioned in Section 4.1). To set up the server, it is necessary to create a new `Session` class. This class is responsible for the correct setup of the streaming preferences (implementation showed in Listing 4.2). To set up the source of the audio (microphone, phone call or all audio), we have to perform a change in audio source of the audio track that is streamed. In this source code variable `audioQuality` is an `AudioQuality` class that holds the settings of the quality of the audio. In our case this is set to 128Kb/s with 44100 Hz sample rate. The variable `audioSource` is an integer, that is set to one of the constants of the `MediaRecorder.AudioSource` object.

Listing 4.2: `Session` setup

```
Session mySession = SessionBuilder.getInstance()
  .setContext(getApplication())
  .setAudioEncoder(SessionBuilder.AUDIO_AAC)
  .setAudioQuality(audioQuality)
  .setVideoEncoder(SessionBuilder.VIDEO_NONE)
  .build();

mySession.getAudioTrack().setAudioSource(audioSource);
```

It is also possible to set a custom port of the streaming using shared preferences provided by the Android system. The implementation of this setup is showed in Listing 4.3. In our case the port `MY_PORT_NUMBER` is set to value 44444. Without setting this value the communication uses the default RTSP port number 8086.

Listing 4.3: Custom port number setup

```
Editor editor = PreferenceManager.getDefaultSharedPreferences(this).edit();
editor.putString(RtspServer.KEY_PORT, String.valueOf(MY_PORT_NUMBER));
editor.commit();
```

After all the setups, to start or stop the RTSP server from the activity, the code provided in Listing 4.4 is necessary.

Listing 4.4: Start and stop `RtspServer`

```
// Start the RTSP server
this.startService(new Intent(this,RtspServer.class));

// Stop the RTSP server
this.stopService(new Intent(this,RtspServer.class));
```

4.4 UPnP Communication

Libraries named `Cling` and the `Cling Support` (mentioned in Section 4.1) were used in this application, for the communication with the UPnP protocol. These libraries implement all the necessary functions to find and communicate with other devices on the network.

The `MainAcitivity` class of the application starts a service, that is implemented in an `AndroidUpnpService` class. This service binds devices on the network. The found devices are stored as a `Device` object and saved into an `ArrayAdapter`, then listed on the screen. More about this service in Section 4.5, part Application behaviour.

The communication with the devices is managed by using `ActionCallback` classes, that are implemented in the `Cling Support` library. The `setAVTransportURI` action is the one, which extends the `ActionCallback` class. This action sends the URI string of the media location, that should be played by device. Another used action is the `Play`. The `Play` action tells the device to start the playback of the media that's URI was sent earlier.

To stop the media stream a `Stop` action is sent to the device. There is also a possibility to call a `Pause` action, but in our case the application doesn't use this function.

The listing number 4.5 shows the pseudo-code of the actions. As an example it uses `Play` action. In this code, the `device` variable is `Device` class that was selected from the `ArrayAdapter` listed on the screen. The `upnpService` is the service that binds the devices on the network.

Listing 4.5: `ActionCallback` pseudo-code

```
ServiceId serviceId = new UDAServiceId("AVTransport");
Service service = device.findService(serviceId);
ControlPoint controlPoint = upnpService.getControlPoint();

ActionCallback playAction = new Play(service);
playAction.setControlPoint(controlPoint);
playAction.run();
```

4.5 The Final Application

The final application named `AudioShare` is a simple application, that allows the user to stream the audio from the microphone, phone-call or other audio sources to an UPnP media renderer. The implementation of the different components were explained in previous sections. This section focuses on how the application behaves, and how it interacts with the user.

Application behaviour

Figure 4.2 shows, what kind of processes and actions are performed by the application.

To start the application, the device has to be connected to a WiFi network. If it is not connected, the application notifies the user about it and asks to connect to a WiFi hot-spot.

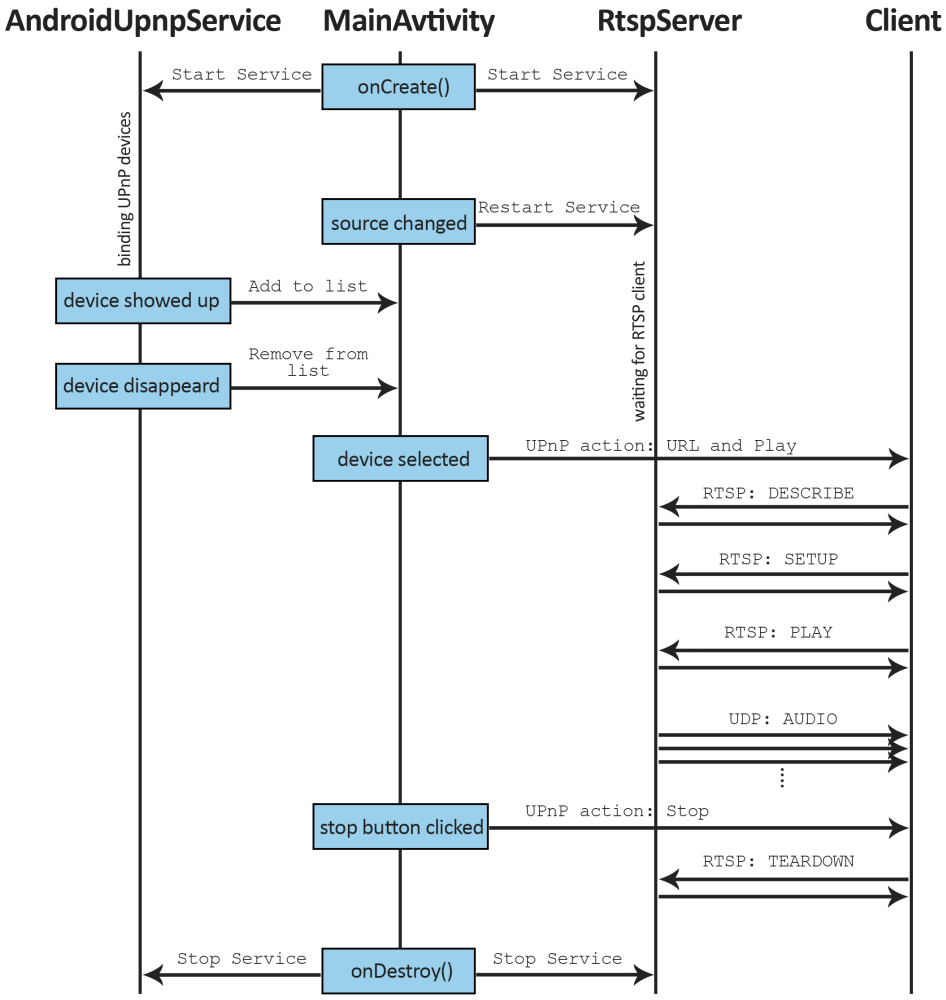


Figure 4.2: Application Flow Diagram.

When starting the applications `MainActiviry` activity, the `onCreate()` method is called, as for every Android activity dercribed in Chapter 2.4. At this point, both the

RtspServer and the AndroidUpnpService service is started.

The AndroidUpnpService is running in the background, while the application is alive. The service binds the UPnP devices on the network and adds or removes them from the list listed on the screen by the MainActivity.

The RtspServer service also runs in the background during the whole lifetime of the application. This service waits for clients to connect and request the audio stream. If the audio source was changed by the user, this service is restarted. This restart is necessary for the MediaRecorder run by the service (see MediaRecorder state diagram in Figure 2.8). Performing a change in source is not available while audio is streamed to a client.

With selecting a device from the list showed on screen (list showed in Figure 4.3), the application sends the URL to the selected UPnP device and an action to start playing it. The device this way connects to the running RtspServer and the streaming with the RTSP protocol is started. If a not valid (no media renderer) device was selected from the list, the application notifies the user that a wrong device was selected.

When the Stop button is clicked (visible in Figure 4.5), an action to stop is sent to the client. The client this way disconnects from the server.

In case the application is exited (onDestroy()), both the AndroidUpnpService and the RtspServer service is stopped.

User interface

The following figures show the graphical user interface of the application AudioShare. The application has a portrait mode only interface.

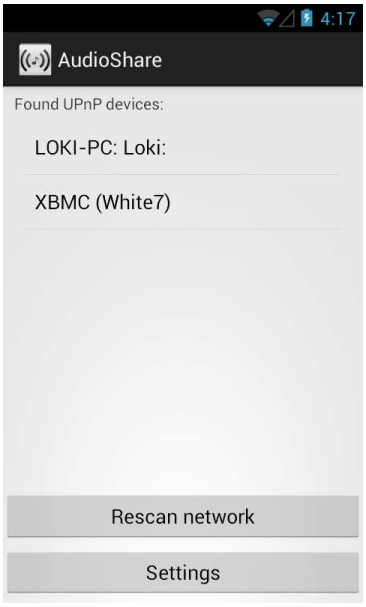


Figure 4.3: AudioShare home screen.

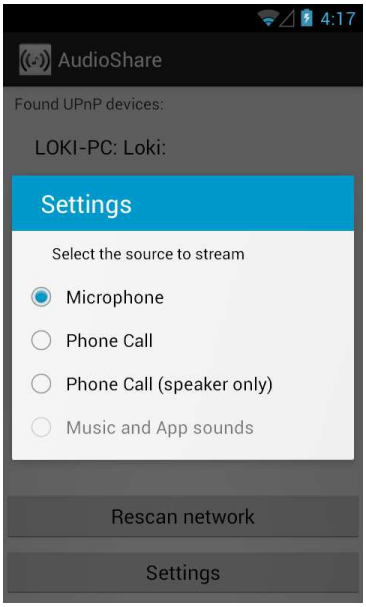


Figure 4.4: Audio source settings.

In Figure 4.3, there is the default application layout with the listed found devices and the button, to refresh the list of devices and to open the settings (see Figure 4.4). In the settings

window the last option to set the audio source to *Music and App sounds* is only available for devices with Android API version 19 or later [5].

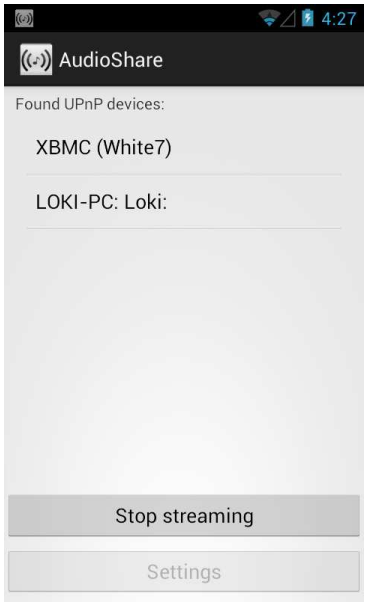


Figure 4.5: `AudioShare` is running.

When clicking on the device, the streaming starts and a notification appears on the Android notification bar (see Figure 4.6). While the streaming is running (see Figure 4.5), there is no possibility to change the settings, or to select a different device from the list of devices.

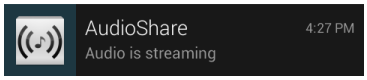


Figure 4.6: Notification of the running the streaming.

When the audio streaming is stopped, the notification disappears from the notification bar.

[5]The reason for this limitation is the described in Section 2.4 part Media Recorder, Audio Source.

Chapter 5

Testing and evaluation

The application was tested on 3 different smartphones with different API levels of Android, and with a UPnP media renderer connected to the network.

The three android devices were a *HTC Desire S* with a non-rooted Android 4.0.4, a *Samsung Nexus 10* with non-rooted Android 4.4.2, and a *Samsung Galaxy S (GT-I9000)* with a rooted *CyanogenMod*[1] Android version 4.1.2. The media renderer was an *XBMC*[2] *version 12.3* multimedia application installed on 64 bit Windows 7 operating system. The used network was a WiFi b/g connection with a 24Mb/s speed.

With all three devices the application behaved in most cases the same way. To detect the *XBMC* media renderer, it was not necessary to use the `Refresh` button. The devices connected or disconnected from the network were automatically added or removed from the list. To create a connection with the the media renderer usually takes 2 to 3 seconds until the playback starts. The delay in the streaming was usually 0.5 to 1 seconds depending on the workload of the network. This delay was also caused by buffer setups of the media renderer. In case of an highly overloaded network an over 6 second delay was measurable, or even data loss was emerged. Some data loss was also caused by the overload of the phones memory or CPU. This problem has occurred only for the *HTC Desire S* and the *Samsung Galaxy S* devices, that have only a 1GHz CPU and under 512 MB of RAM.

On all three devices the application took 4.5MB of storage size. While running without streaming, the application consumed 13-14 MB of RAM. The RAM usage was raised to 16-17MB when the streaming was running.

Deficiencies and limitations

During testing, few limitations and problems were detected in the applications reliability and functionality.

In most of cases a problem has occurred when the application was started without WiFi connection. After connecting to the network while the application was in the background,

[1] *CyanogenMod* official website: http://www.cyanogenmod.org/
[2] *XBMC* official website: http://xbmc.org/

the client was unable to connect to the RTSP server. The solution for this problem was to kill the application in task manager, then start it again with the connected WiFi.

Due to the limitations of the Android system, the final application does not provide all the functions that were set as a goal in Section 3.1.

The application is able to stream the microphone and the phone call audio, but it is not able to stream the audio from music or video players, nor from games. Even if the Android API level 19 (Android 4.4) contains the `MediaRecorder.AudioSource` setting to `REMOTE_SUMBIX`, this setting is not working for third-party applications. It is only reserved for use by system components [11].

The application was not tested on a device with rooted Android 4.4 operating system. On that system, all kind of audio streaming might work, due to the permission to access system features. Only this rooted android system might be able to provide a functional audio stream of every audio source of the device .

Chapter 6

Future Improvements

The created application is capable to provide audio stream to UPnP devices. However, in this form it has no practical or meaningful usability in real life. If we would decide to continue our development, it would be possible to create an application that is practical and useful for the users.

One option is to create a walkie-talkie application. For that, an additional UPnP renderer service would be necessary for the application. That way the phones would be able to find each other, and stream audio to each other.

Another option is, to use it as a phone communication extension for PC. That way the user would be able to make receive phone calls using a PC, even if the phone is not in the a reachable destination. The mobile could notify the PC client using custom action about the incoming phone call and the user could accept is, and establishing a two way audio streaming between the devices. The user could use the microphone and the speakers of the computer instead of the hand-held device.

Bibliography

[1] H. Schulzrinne A. Rao and R. Lanphier. Real Time Streaming Protocol (RTSP). RFC 2326, April 1998.

[2] Rogue Amoeba. Preventing audio delays while watching videos with airfoil. [online]. http://rogueamoeba.com/support/knowledgebase/?showArticle=AirfoilVideoPlayer. [cited 2014.07.09].

[3] AndroidPIT. Top 5 music players: The best way to play music on android. [online]. http://www.androidpit.com/top-5-music-players. [cited 2014.05.23].

[4] Various Authors. Upnp device architecture 1.1. [online]. http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf. [cited 2014.01.15].

[5] Bocha. Mindent a bluetooth multipointról. [online]. http://mobilarena.hu/teszt/mindent_a_bluetooth_multipointrol/a_blutooth_multipoint_lenyege.html. [cited 2014.06.09].

[6] Dennis Burger. Bluetooth, airplay, dlna: What's the best streaming format today? [online]. http://hometheaterreview.com/bluetooth-airplay-dlna-whats-the-best-streaming-format-today. [cited 2014.01.15].

[7] Douglas E. Comer. *Internetworking with TCP/IP Vol 1*. Alan Apt, 2000. ISBN: 0-13-018380-6.

[8] Google Android Developers. Android.media, 2014. http://developer.android.com/reference/android/media/package-summary.htmll.

[9] Google Android Developers. Mediarecorder, 2014. http://developer.android.com/reference/android/media/MediaRecorder.html.

[10] Google Android Developers. Mediarecorder.audioencoder, 2014. http://developer.android.com/reference/android/media/MediaRecorder.AudioEncoder.html.

[11] Google Android Developers. Mediarecorder.audiosource, 2014. http://developer.android.com/reference/android/media/MediaRecorder.AudioSource.html.

[12] Jason A. Donenfeld. Airtunes 2 protocol. [online]. http://git.zx2c4.com/Airtunes2/about/. [cited 2014.07.09].

[13] Dreamcss. Top 10 dlna streaming apps for android. [online]. http://blog.dreamcss.com/android/dlna-streaming-apps-for-android/. [cited 2014.05.23].

[14] eMarketer.com. Smartphone users worldwide will total 1.75 billion in 2014. [online]. http://www.emarketer.com/Article/Smartphone-Users-Worldwide-Will-Total-175-Billion-2014/1010536. [cited 2014.05.23].

[15] Shawn Van Every. *Pro Android Media*. Apress, 2009. ISBN: 978-1-4302-3268-1.

[16] Sumanth Gopinath and Jason Stanyek. *The Oxford Handbook of Mobile Music Studies, Volume 1*. Oxford University Press, 2014. [cited 2014.05.23].

[17] Whitson Gordon. Everything you need to know about rooting your android phone. [online]. http://lifehacker.com/5789397/the-always-up-to-date-guide-to-rooting-any-android-phone. [cited 2014.06.12].

[18] Armijn Hemel. Upnp stack layout. [online]. http://www.upnp-hacks.org/upnp.html. [cited 2014.01.15].

[19] Chris Hoffman. Wireless display standards explained: Airplay, miracast, widi, chromecast, and dlna. [online]. http://www.howtogeek.com/177145/wireless-display-standards-explained-airplay-miracast-widi-chromecast-and-dlna. [cited 2014.01.15].

[20] Wei-Meng Lee. *Beginning Android Application Development*. Wrox, 2011. ISBN: 9781118087299.

[21] LiNUXLiNKS.com. 5 best free android upnp clients. [online]. http://www.linuxlinks.com/article/20120112154102183/UPnPClients.html. [cited 2014.05.23].

[22] Feipeng Liu. *Android Netive Development Kit Cookbook*. Packet Publishing, 2013. ISBN 978-1-84969-150-5.

[23] Natasha Lomas. Android still growing market share by winning first time smartphone users. [online]. http://techcrunch.com/2014/05/06/android-still-growing-market-share-by-winning-first-time-smartphone-user [cited 2014.07.09].

[24] Adrienne Maxwell. How do i connect my tablet to my tv? let me count the ways... [online]. http://hometheaterreview.com/how-do-i-connect-my-tablet-to-my-tv-let-me-count-the-ways. [cited 2014.01.15].

[25] Rafael Osso. *Handbook of Emerging Communications Technologies*. CRC Press, 1999. ISBN: 978-0849395949.

[26] Steve Paine. Widi ? wireless display overview, specifications, testing and demos. [online]. http://www.umpcportal.com/2014/02/widi-wireless-display-overview-specifications-testing-and-demos/. [cited 2014.07.09].

[27] A.J. Peck. What is dlna? [online]. http://usacomp2k3.blogspot.sk/2009/12/what-is-dlna.html. [cited 2014.07.09].

[28] Thomas Phelps. To root or not to root. [online]. http://google.about.com/od/socialtoolsfromgoogle/a/root-android-decision.htm. [cited 2014.06.12].

[29] Brent Rose. 9 reasons to root your android device. [online]. http://gizmodo.com/5982287/reasons-to-root-your-android-device. [cited 2014.06.12].

[30] Serge Smirnoff. Bluetooth audio quality - a2dp. [online]. http://soundexpert.org/news/-/blogs/bluetooth-audio-quality-a2dp. [cited 2014.07.09].

[31] Chris Smith. Making calls fifth most popular use for smartphones, says report. [online]. http://www.techradar.com/news/phone-and-communications/mobile-phones/making-calls-fifth-most-popular-use-for-smartphones-says-report-1087623. [cited 2014.05.23].

[32] Quora/Bjorn van Raaij. How are volume changes applied to an airplay audio stream? [online]. http://www.quora.com/AirPlay-media-streaming-protocol/How-are-volume-changes-applied-to-an-Airplay-audio-stream/. [cited 2014.07.09].

[33] Richard Zurawski. *The Industrial Information Technology Handbook*, chapter RTP, RTCP, and RTSP - Internet Protocols for Real-Time Multimedia Communication. CRC Press, 2004.

Appendix A

Content of the CD

The attached CD contains the following directories and files:

- Directory: `AudioShare` - This directory contains the source files of the application. This folder can be imported into Eclipse IDE as an existing project. In the `AudioShare/bin` folder, there is a `AudioShare.apk` file that can be directly installed on Android devices.
- File: `thesis.pdf` - The technical report in PDF format.