Institute for Visualization and Interactive Systems

Bachelor's Thesis Nr. 51

Development of an Audio Input Toolkit for Multiple Sources

Thomas Kosch

Course of Study:	Software Engineering
Examiner:	Prof. Dr. Albrecht Schmidt
Supervisor:	Dipl.-Inf. Markus Funk
Commenced:	2013-04-23
Completed:	2013-10-22
CR-Classification:	H.1.2

Kurzfassung

Audio Dienste, wie voice over IP oder Spracherekennungsdienste, haben sich in den letzten Jahrzehnten stark weiterentwickelt. Diese rasante Weiterentwicklung ist einer immer einfacheren Benutzung dieser Dienste zuzuschreiben. In dieser Thesis über das verarbeiten mehrerer eingehender Audiosignale, soll ein Audio Toolkit entwickelt werden, dass diese eingehenden Audiosignale verarbeitet. Als Eingabegeräte werden Bluetooth Headsets verwendet, welche die Audiosignale über das UDP Protokoll an ein Audio Toolkit übertragen. Das Audio Toolkit ermittelt aus allen eingehenden Signalen ein dominantes Signal, welches als einziges hörbar sein soll.

Ziel dieser Bachelorarbeit ist die Entwicklung des beschriebenen Audio Toolkits. Das dominante Signal kann an einen voice over IP Dienstleister wie beispielsweise Skype oder an eine Sprachererkennungssoftware, wie beispielsweise die Microsoft Speech API, übertragen werden.

Darüber hinaus gibt diese Thesis einen Überblick über die Funktionsweise der Soundverarbeitung, Entwicklung geeigneter Algorithmen und dem Entwicklungsprozess.

Abstract

Audio services, like voice over IP or several voice recognition systems, are developing very fast and since they are easy to use nearly everybody is linked to such systems. In this thesis about the processing of multiple audio inputs, an audio toolkit for processing multiple audio inputs has to be developed. Used audio input devices are bluetooth headsets, which can send audio via UDP to the audio toolkit. This audio toolkit is able to process these multiple audio inputs and determines a dominant signal. The dominant signal is a signal from a specific client with an audio input device.

The focus of the audio toolkit is to suppress every other signal than the dominant signal. The dominant signal can then be transferred to a voice over IP service, like Skype, or to a voice recognition system, like the Microsoft Speech API.

This thesis gives a general overview how audio processing works, the development of algorithms which determine the dominant signal and the development process.

Contents

List of Figures

List of Tables

1. Introduction

During the last years, voice over IP, short VOIP, and voice recognition systems largely increased and still will grow further. This comes with the ease of use of such systems and technologies, which improved over the last years. Nowadays, it's no problem to access the internet and make a VOIP call with tools like Skype.

In the last years, technologies and procedures were developed to improve audio recording with several audio input devices, for example built-in-microphones, wired headsets or wireless bluetooth headsets. In this thesis, the focus of sending audio signals was set on bluetooth headsets.

1.1. Motivation

When many people discuss simultaneously in the same physical place, its difficult to understand what the conversational partner is saying. The main cause are the disturbations by other people in the same place, like in trains, a car full with people or a variegated place.

The same problem appears in big VOIP conferences. Many people from all over the world are speaking together at the same time, making it difficult to understand each other.

Some applicable solutions already exists, like muting manually or scaling down the voice of other people which disturb the conference to improve the audible quality of the conversational partner. This are manual actions which have to be performed by the user. From this it follows, that the user has to change the settings by hand and every time the audible environment changes, eventually the user has to reconfigure his settings.

Noise and other sound disturbances, like using a microphone with a bad audio quality, can have an effect on the audio quality of the audio conference or speech recognition system. The error rate of speech recognition increases very fast with increasing amount of additive noise.

However, a full automatic solution is more comfortable, because it requires less action from the user. An audio toolkit, which automatically determines a dominant signal and makes this dominant signal audible for every person involved into a VOIP conference or to a speech recognition.

An use case is a VOIP conference with many people, which are speaking simultaneously. This makes it difficult to understand a person which is really speaking at the moment. If only one signal would be hearable, the conversation may become better understandable.

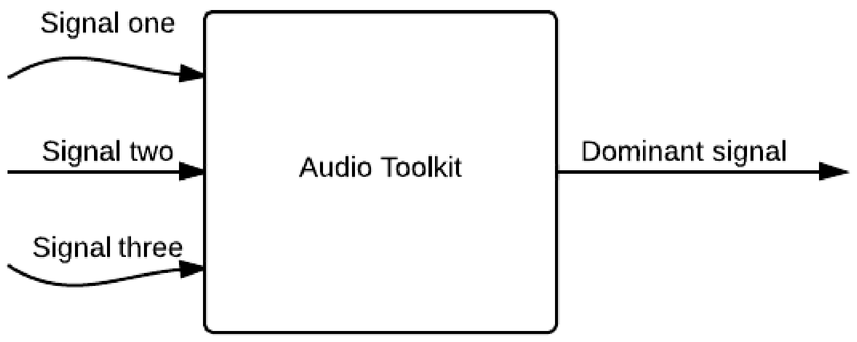

Figure 1.1.: Black box model of the audio toolkit. Multiple input signals are processed by the audio toolkit. The audio toolkit determines dynamically one signal as the dominant signal and makes it audible. All other signals are suppressed until one of these signals becomes dominant.

Another use case is a smart home with integrated speech recognition, which can be configured by the owners speech, for example by letting the shutter down or increase the temparature of the heater by voice. If there are many people in the smart home talking simulteanously, some speech commands may not be recognized or may be interpreted false.

In this bachelor thesis several methods for determining a dominant signal are developed. This dominant signal is the only audible signal, which has to be filtered out of the other signals. Therefore, algorithms have to be developed, which efficiently determine the dominant signal.

Furthermore, an audio toolkit will be developed, which is able to filter an audible dominant signal out of multiple input signals. This thesis shows the development process of the algorithms and the development process of the audio toolkit. For a visualisation of the idea behind the functionality of the audio toolkit, see Figure 1.1.

1.2. Related Work

Some approaches to suppress signals in advantage for one single signal out of every incoming signal were already discovered.

With the invention of the telephone by Philip Reis, people are able to make calls all around the world. These calls often contain private informations, which should be secret to other people. Sensitive organisations, like secret agencies or police departments discovered that fact very fast.

Thus they started to use walkie talkies, which send audio data on a specific encrypted broadband, which can't be listened in to easily. These devices already supported a push to talk function. The user of a walkie talkie must press a specific button to make himself audible for other walkie talkie user, which are in range and in the same bandwidth as the user. With these devices, some fundamental approaches were made for suppressing other audio signals. The approach by pressing a button for making the user audible still requires interaktion by the user. Walkie talkies doesn't support a full automatic mute solution.

The principle of walkie talkies is known as push to talk. Push to talk means, that an interaction like pressing a button is necessary for making himself audible. Other software like Teamspeak support a push to talk functionality in their settings. This still mutes the user himself and not the other user, which disturb for example a conversation.

Another solution used for example by teamspeak is to set up a threshold. If the voice of an user exceeds this threshold the sound data will be transmit and audible to other user. This solution isn't optimal, since little disturbations like surrounding sounds over the threshold will be transmitted. Another problem can occur, if more than user starts to talk. This can make it difficult to understand each other.

The fast development and expansion of smartphones among people keeps the desire to use VOIP and speech recognition services on these devices. Some apps like Skype or Siri are already available on these devices. Some approaches, like measuring decibel or using a smartphone as a sound processing device for the environment have already been done [SZZ+11]. Smartphones offer a great potential for sound applications in near future.

Speech recognition acquired more and more attention over the last years. A speech recognition system detects voice pattern and tries to determine a valid command. Furthermore, a good speech recognition must be able to detect the language and the dialect, which is not always easy. Patents for recognizing language and dialects already exist, but are hard to integrate into a speech recognition sytstem [BME+99].

Mathematically seen, a language can contain infinite many dialects. The art of dialect, a speech recognition system must detect, depends mostly on the origin and the nationality of the person which uses the speech recognition system [KT97].

Suppressing additive noise plays a big role for a good speech recognition quality. Even a small amount of noise can destroy the recognition of speech during speaking [HP00]. Reducing of noise is possible with pseudoinverse Wiener-Filtering [SFB00] or transforming the sound into the wavelet domain and processing it further from there [HM03]. Both steps require many computation steps and are quite complex. Because noise reduction is very complex it isn't part of this thesis.

1.3. Purpose of this Thesis

This thesis tries to develop applicable solutions for muting non-speaking or quiet speaking persons in a conversation. This includes the usage of different devices, mainly bluetooth headsets. These art of headsets provide the best sound and recording quality without much costs [CKL$^+$04].

The development process of the solutions are described with their advantages and disadvantages.

The solutions will then be implemented and evaluated in an user study. Important informations will be gathered about the participants after the evaluation to show if such an audio system is desired or denied.

Structure

The thesis is structured in the following way:

Section 2 – Basics of Signal and Sound Processing: This chapter is describing the problems and possible solutions for determining the dominant signal.

Section 4 – Development Process of the Audio Toolkit: This chapter describes the development, problems that occurred while the development and approaches of the audio toolkit, which determines the dominant signal out of multiple input signals. Furthermore, this chapter describes the used technologies for validating and developing the audio toolkit.

Section 5 – Evaluation: This chapter contains the approach and an accomplishment of a survey. Furthermore, the results and conclusions are shown.

Section 6 – Limitations: This chapter discusses restrictions of the audio toolkit.

Section 7 – Conclusion and Outlook: This chapter shows an outlook into the progress of the digital sound processing.

2. Basics of Signal and Sound Processing

This chapter shows some approaches to a solution for the problem described in the introduction. It explains the basic knowledge of sound processing, encoding and decoding of the data. This includes the definition of sound, how sound works and how sound can be processed efficiently.

2.1. Digital Sound Processing

The development of digital sound processing took great progress over the last centuries. Since the interest became great in the research of saving audio onto persistent mediums, people were trying to perform research on those mediums. Methods for processing and modifiying these audio recordings were a big goal in research. In the following, some basics of the digital audio processing will be explained.

2.1.1. Audio Signals

Audio Signals consists of waveforms, which variegate in different amplitude, audio sampling rate, pitch etc. . The two most important properties, of an audio signals waveform, is the amplitude and the audio sampling rate.

Waveforms

Sound that we hear are vibrations in air, which are caused by human voice, music or noise. Sound moves air molecules next to it, which in turn moves air molecules next to them.

A microphone has the ability to detect these air vibrations. E.g. if a tuning fork is stroked, the air pressure begins to rise. While these vibrations exist, the air pressure rises. When the vibration reaches its maximum, it goes back to normal and overshoots the normal line, resulting for reaching its minimal extrema (Figure 2.1). This procedure goes on until the vibration fades.

This "up" and "down" at Figure 2.1 in the sinusoid is called amplitude. The higher the amplitude is, the louder the sound is. The closer the waveforms are, the higher the sound appears to be. With closer waveforms, more vibrations will appear.

These vibrations are measured in Hertz. One cycle corresponds one Hertz. The cycles must be fast enough to be heard by a human being. A regular human ear is able to hear cycles

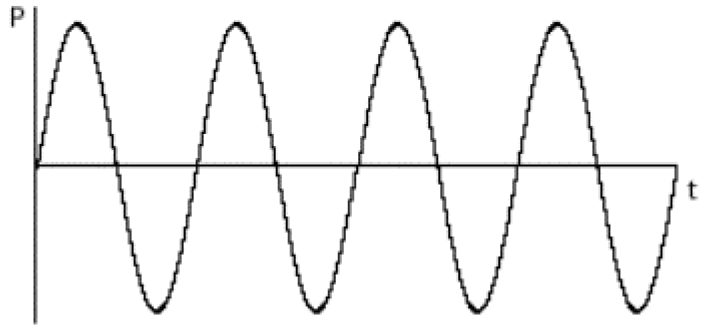

Figure 2.1.: Sinusoid as a waveform, which represent a constant air pressure. P represents the level of the amplitude. The parameter t represents the elapsed time.

between 20 to 20.000 Hertz, but with increasing age a regular human can only hear 17.000 Hertz or lower [MSP96].

Audio Sampling Rate

The audio sample rate explains, how many data points are saved per second. For example, an audio CD is sampled with 44.100 Hertz. This means, that one data point is performed in $\frac{1}{44.100}$ second. A great audio sampling rate usually means a higher audio quality [Esp04].

A normal human can hear up to 17.000 Hertz. Higher frequencies cannot be heard by a human ear. A big audio sampling rate has the following advantages:

- A great audio sampling rate has a better audio quality, because more data points are saved.
- Recordings, whom have a great sampling rate, can be used for processing in studios.

Unfortunately, there are a lot of disadvantages too:

- A great audio sampling rate needs more disk space to be saved. This cannot be avoided, because more data points are saved than a recording with a low sampling rate.
- If a recording with a great sampling rate has to be processed (e.g. increasing amplitudes to make it louder or removing background noise), this can have a negative effect on the processing time of the audio recording, because more data points have to be processed.

Nyquist-Shannon-Theorem

The Nyquist-Shannon-Theorem explains why it's necessary to sample audio signals always with a doubled sampling rate as later wanted. For example, if an audio signal has to be sampled with 5000 Hertz, it has to be sampled with 10.000 Hertz instead to perserve the audio quality.

Because of noise, which is created by the recording of a discrete signal, an exact reconstruction of the audio signal is only possible, if the audio signal was sampled with a doubled sample rate. The exact reconstruction of a discrete audio signal is impossible, since an exact reconstruction of an audio signal would need an infinite amount of sample or data points.

Fortunately, the signal can be restored by the interpolation of the sampling points. A very good interpolation technique is the Lagrange- or B-Spline interpolation, which nearly completely reconstructs the signal like it was recorded.

Most interpolation techniques must fulfill two important properties. These two properties are accuracy and complexity. The cubic B-Spline interpolation is very accurate and doesn't require many computation steps [Sha49] [OP89] [CW08] [KH99].

Quantization

The digitalization of audio has two steps. First, the incoming infinite accurate and exact analogue signal has to be sampled with finite accuracy. This step means loss of informations, which can not be avoided. This error is very minimal. A human hear isn't able to hear the error. In the second step the informations won out of the analogue signal are saved into a digital format [KH99].

2.2. Digitalization of Analogue Signals

The digitalization of the recorded or actually recording audio happens in the context of the so called Pulse Code Modulation.

2.2.1. Pulse Code Modulation

The Pulse Code Modulation (PCM) is a transmitting technique, which transmits each wave to a channel. The transmission are sampled and are nearly taken simultaneously. The conversion of voice into pulse code modulation needs sampling and quantization.

The Nyquist-Shannon-Theorem states that a signal can be reproduced if a signal is sampled with the doubled sampling rate like the highest sampling point of the original signal. E.g. if a discrete signal $f(x)$ contains a highest sampling point of x samples per second, the doubled amount of the sampling rate is necessary to fully reconstruct the original signal (in this case $f(2x)$).

After sampling, the signal will be quantized (see figure 2.2). The points of the recorded signal can now be saved in binary form. For reconstruction of the digital signal, the signal has to be interpolated by an efficient interpolation technique.

If the interpolation was successful, the signal can be played as an analogue signal. There are many ways to transmit the signals to a PCM system. The sound quality is nearly the same like the once recorded signal [BE47].

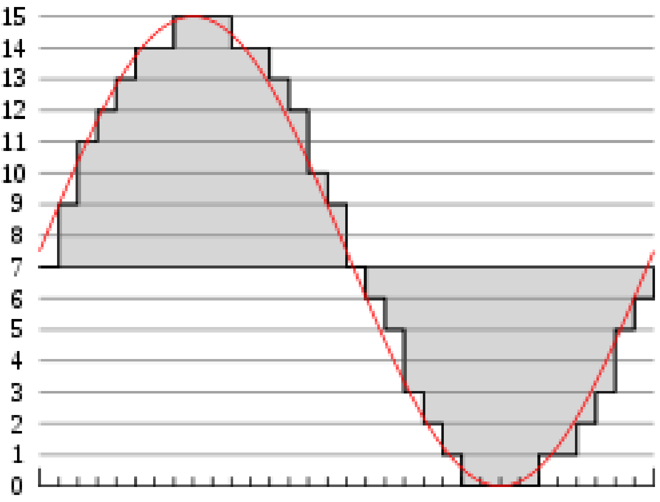

Figure 2.2.: PCM of a recorded signal: The red sinusoid function shows the natural signal, the black "box function" shows the digital recorded signal.

Connection of PCM and the Nyquist-Shannon-Theorem

The basic idea behind PCM is the Nyquist-Shannon-Theorem. The Nyquist-Shannon-Theorem asserts, that a signal $s(t)$ can be recovered from a sequence of of its sample values according that W is the highest amplitude in the whole signal. The signal $s(t_j)$ can now be recovered in the point j with $t_j = \frac{j}{2W}$ [Llo82]. PCM is a modification of this, which makes it easier to transmit data.

3. Prototype

For the development of the audio toolkit GUI mockups are necessary. These mockups have to satisfy the functional requirements of the audio toolkit.

3.1. Specification

The audio toolkit must be able to record signals from multiple input sources and determine a dominant signal. The dominant signal is the sound signal which has to be piped through the toolkit to a destination source.

3.1.1. Requirements

The audio toolkit has to meet the following requirements:

- Support for Linux and Windows.
- Support for maximal five devices.
- An algorithm, which determines out of more than one speaker a dominant speaker.
- A Graphical User Interface.
- Network support.
- Piping the dominant speakers signal to a source (e.g. speaker).
- Piping the dominant speakers signal to a file (e.g. speaker).
- The audio toolkit processes time critical operations. The developed algorithms have to be efficient.
- Support for other audio services, for example Skype or Teamspeak.

3.2. Mockups of the Desktop Application

The mockups are designed for the operating systems Microsoft Windows 7 and Linux. At Figure 3.1 the mockup of the audio toolkit for Linux is visible. At Figure 3.2 the mockup of the audio toolkit for Microsoft Windows 7 is visible.

3.2.1. Section "Sending audio data via UDP"

In the section "Sending audio data via UDP" are three text fields and two buttons visible. The text fields "IP" and "Port" require a valid IP address and a valid port number. The text field "ID" requires an unique ID, which has maximal ten digits. This ID is required for recognizing each sending client. If the button "Send" is hit, the audio toolkit starts recording the audio data and sends it via UDP protocol to the IP address and port specified in this section. The button "Disconnect" will appear enabled and the button "Send" be disabled. Furthermore, the text fields will be disabled. If the button "Disconnect" is hit, the audio toolkit stops sending audio data. In this case, the button "Send" becomes enabled again.

3.2.2. Section "Receiving audio data via UDP

In the section "Receiving audio data via UDP" is one text field and two buttons visible. The field "Port" requires a valid port number. If the button "Receive" is hit, the audio toolkit starts listening on the port specified in this section. The button "Receive" becomes disabled and the button "Disconnect" becomes enabled. The text field "Port" will be disabled. If the button "Disconnect" is hit, the audio toolkit stops listening on the port specified in this section. In this case, the button "Receive" becomes enabled and the button "Disconnect" becomes disabled.

3.2.3. Section "Options"

In the section "Options" are two checkboxes and a text field. At least one and only one checkbox can be checked. If the checkbox "Write to file" is checked, a save location for the file is required. If the checkbox "Streaming" is hit, the audio signal will be piped onto the preferred audio output device, for example the speaker. All checkboxes and text fields in this are disabled while receiving data.

3.3. Mockup of the Android Application

The Android application contains three text fields and two buttons. The application is supposed for sending data only. Like in the desktop version of the audio toolkit, the fields "IP" and "Port" require a valid IP address and a valid port number. The field "ID" has to contain an unique ID of the sender. If the button "Send" is hit, the button "Disconnect" becomes enabled

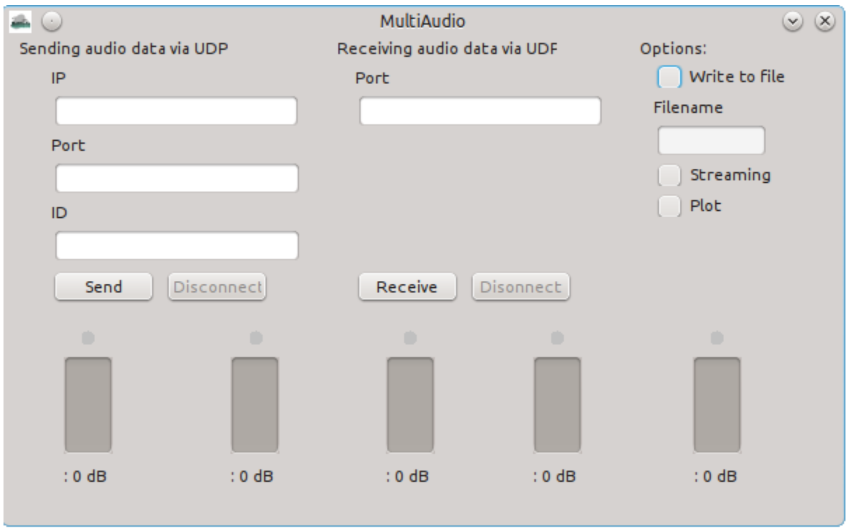

Figure 3.1.: The main UI of the developed audio toolkit on Linux. It supports both sending and receiving of UDP packages.

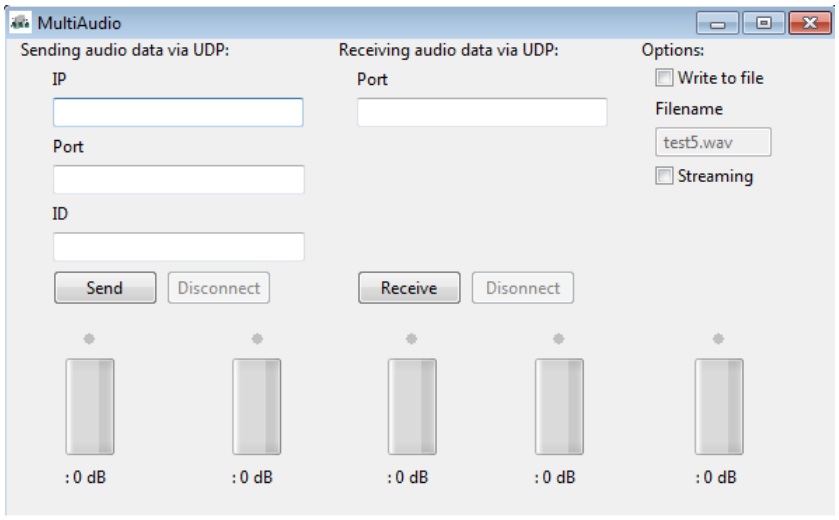

Figure 3.2.: The main UI of the developed audio toolkit on Windows. It supports both sending and receiving of UDP packages.

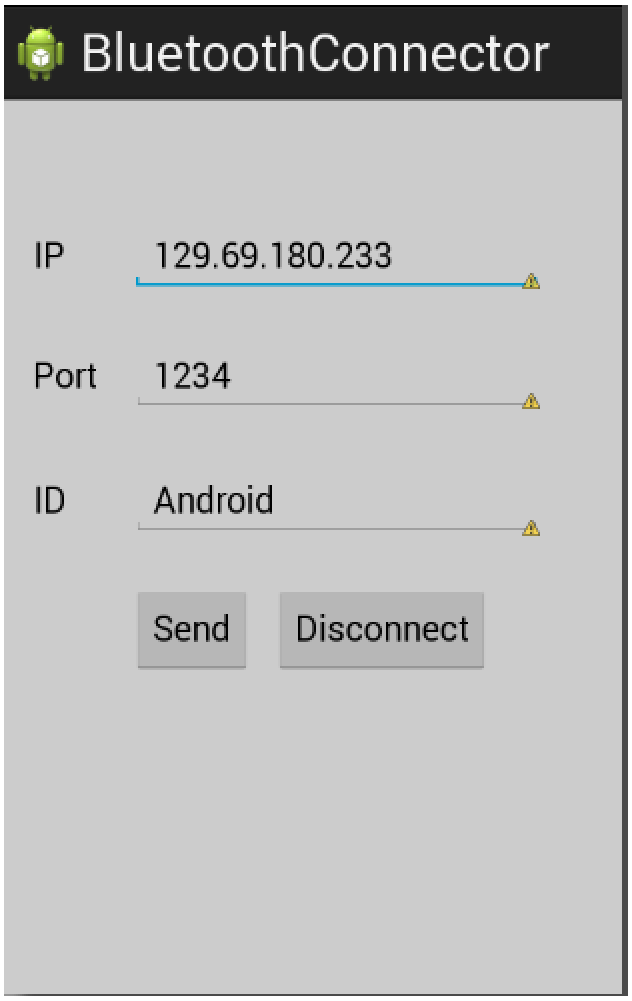

Figure 3.3.: The Android UI for the developed audio toolkit. It only supports sending UDP packages to a host. It can't figure as a host. It contains a sample IP, port and ID.

and the button "Send" becomes disabled. If the button "Disconnect" is hit, the button "Send" becomes enabled and the button "Disconnect" becomes disabled.

4. Development Process of the Audio Toolkit

This chapter explains how the audio toolkit was developed and tested. Different approaches and aspects will be discussed.

4.1. Prerequisites

While the development of the audio toolkit, some restrictions and hints were discovered. These restrictions and hints are listed below.

4.1.1. Connection of Bluetooth Headsets

While trying to connect more than one bluetooth headset to a computer, it was discovered, that most widely used operating systems like Microsoft Windows or Linux aren't able to handle more than on bluetooth dongle or more than one bluetooth headset to one bluetooth dongle. But there are some applicable solutions to this problem:

- A virtual machine can take one bluetooth dongle and one bluetooth headset. By starting more virtual machines more devices can be connected. This solution is easy to use but has a lot of memory usage. Furthermore, every virtual machine has to be configured which costs a lot of time.
- The multiple usage of laptops is a very easy solution too, but needs a lot of configuration time.
- Trying to connect to the host via smarter devices, which everyone can handle and send the audio data via the UDP protocol.

The last point includes Smartphones and offers the most interesting solution. Smartphones are comfortable devices which can be handled very easy. Because of that, an Android app was developed, which records data from the Android phone and sends it via UDP to the host which takes the data and processes it. The Android app is compatible with every Android device with version 4.3 or fewer.

Data	**Value**
Frequency	8000 Hertz
Bits	16
Channel	1

Table 4.1.: Used soundformat

4.1.2. Art of the Connection

Like mentioned in the subchapter before, the UDP protocol for streaming audio is used. The UDP protocol transmits bytearrays. The audio toolkit was specified with 2200 bytes per array, which consist of pure audio bytes. The host receives the bytarrays and calculates the amplitudes and the decibel to process the audio. For determining the sender of the incoming bytearrays, ten bytes are added to the bytearrays as an id.

The total amount of bytes send by the sender is 2210 bytes. The last ten bytes are representing the id. The id cannot be longer than ten characters.

4.1.3. Soundformat

The soundformat must be equally defined between host and sender. If the soundformat is not equally defined, errors will occur while processing the audio. An overview of the used soundformat can be taken from table 4.1.

4.1.4. Programming Language

The used programming language is Java 1.7. The audio toolkit requires Java 1.7. Java offers a Java Sound API, which allows to record and play audio signals. The Java Sound API allows to decompose the recorded audio signals into their bytes, audio streams and can write them into files which correspond to the WAV standard [DJ04].

4.1.5. WAV Format

The WAV format is a sound format, which contains Pulse Code Modulation data and was developed by Microsoft. Its header consists of 44 bytes. The first 12 bytes contain an id, the chunksize and the format. The id always contains the letters "RIFF" and the format always contains the letters "WAVE".

The chunksize contains the size of the whole file with the header. The next 24 bytes contain a subchunk id, which always contains the letters "fmt", the subchunk size, the audio format which usually contains the number 1 (which means that no compression was applied to the

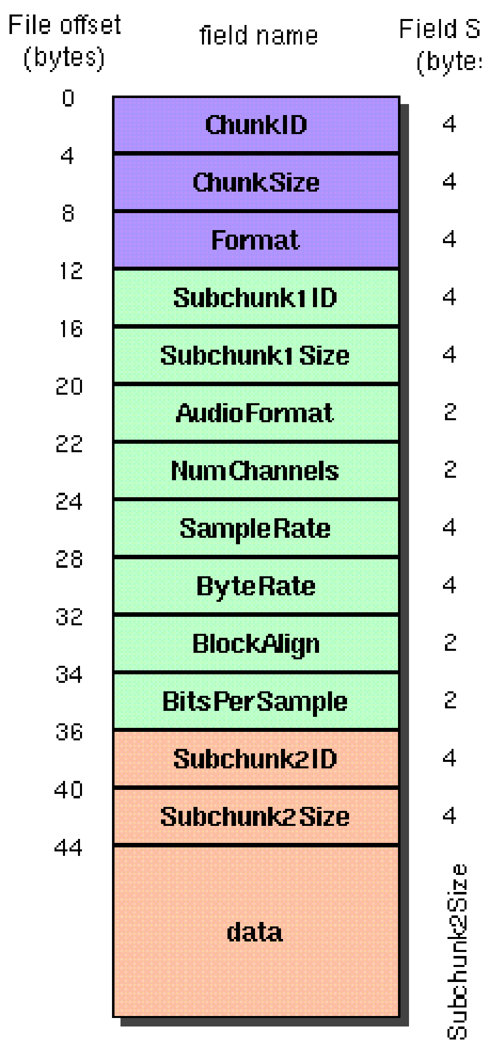

Figure 4.1.: Graphical structure of the WAV format [wava].

file), the number of channels (1 = mono, 2 = stereo etc.), the byte rate, the block align and the bits per sample.

The last 8 bytes of the header consist of a Subchunk id, which contains the letter "data" and the subchunksize, which contains a calculated number out of the formula $\frac{NumberSamples \cdot NumberChannels \cdot BitsPerSample}{8}$. Figure 4.1 shows a graphical illustration of the WAV format.

The rest of the data consist of the recorded audio signal, which is raw data PCM. Without compression (audio format in the header is set to 1) the WAV format can produce big files, which are cumbersome to handle. This seems to be a big disadvantage, but because no quality or information are lost, the WAV format gains support in the most sound processing applications [Sta03].

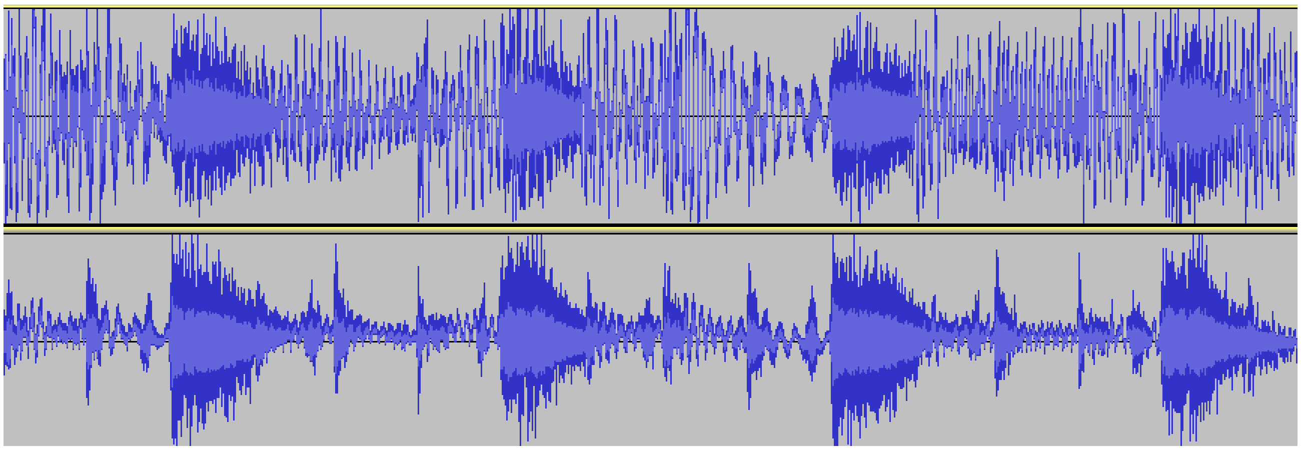

Figure 4.2.: Difference between WAV and MP3. **Top:** A simple beat sampled as WAV format. High frequencies are available, which are making the sound clear and smoother. **Bottom:** The same beat as MP3. High frequencies are suppressed. The beat is still audible, but sounds not clear and not so smooth as the WAV format [wavb].

A big advantage for the WAV format is the fact that it's able to save PCM raw data. This makes WAV suitable for processing audio in professional music studios, because high frequencies (up to 192 kHz) are allowed to record, or make high quality recordings.

A big disadvantage is the fast increasing disk space taken by the WAV format. Each byte is saved into the file without compression, which takes up a lot of data. Another format for saving sound data is MP3 (MPEG-2 Audio Layer III). It uses a discrete cosine transform to cut off too high or too low frequencies, which can not be heard by a human ear. This saves a lot of disk data and sound data can be saved efficiently on a disk. The discrete cosine transform is not part of this thesis. Interested readers should follow the references [MBTY03]. The difference of an audio signal recorded with WAV- or MP3 format can be found in Figure 4.2.

4.2. Determining the Dominant Signal

This section shows the development of the algorithms.

4.2.1. Low- and Highpassfilter

The first approach was to use low- and highpassfilter. This approach uses the following principle:
First off, all input devices has to be calibrate to the voice of the user. Then the filter should be used to determine the dominant voice in the toolkit. This voice has to be piped through the sound system. More mathematically:

$smoothedValue_i = \sum_{i=0}^{N} \frac{byte_i + smoothedValue_(i-1)}{smoothConstant}$, where the smoothedValue at the position i is calculated. The previous smoothed value is taken for calculation to smooth the values better.

Without smoothing the values there have hard quality issues, which can be heard in recordings. The smoothConstant is a constant which decides how "hard" can be filtered. The higher the constant is choosen, the higher the amplitudes will be cut off. A low constant will not cut high signals properly. The only way to make this procedure usable is to calibrate the users voice in a defined range of time. Because every human voice is different, it is difficult to develop an algorithm, which determines the optimal smooth constants. In this thesis the development of such an algorithm was aborted.

4.2.2. Loudest Voice

One another aspect is to determine the person with the loudest voice. The signal with the loudest voice has to be piped to a sound system. More mathematically:
The size of the bytepackage, which contains the sound data, is set to 2200 bytes. In every UDP package are 2200 bytes send, which contains 2200 amplitudes. From these amplitudes one can calculate the decibel, which means in regard that the array contains 2200 decibel values.

From these values the median decibel is taken. The highest decibel median is globally saved and incoming decibel median will be compared with the globally saved decibel median. If one incoming decibel median is higher than the globally saved decibel median, the incoming decibel median will be saved globally and set as the dominant signal. The calculation of the decibel median is as follows:
From the incoming bytearrays, $byte_i$ for the i incoming bytearrays, the decibel must be calculated. Each decibels can be calculated with the formula $decibel = |SmoothConstant * (\log(byte_i[j]))|$, where $SmoothConstant$ is a constant to increase or decrease the calculation of the decibel and $byte_i[j]$ is the j position of the the bytearray $byte_i$.

Furthermore, the constant $SmoothConstant$ allows to modify the result of the decibel in a relative way. With these decibel stored, it's possible to determine one dominant signal, which can be transmit into the audio system. A visual example of this approach can be taken from Figure 4.3

4.2.3. Fast Fourier Transformation

An another idea is to perform a Fourier transform. A Fourier transform is a procedure, which decomposes a signal into its frequencies. This allows a better analysis of the signal. Besides of the fast Fourier transform exist two other transformations: The continuous Fourier transformation and the discrete Fourier transformation.

The details of these two transformation aren't scope of this thesis. Because the continuous Fourier transformation handles infinity long signals and the discrete Fourier transformation has a complexity of $\mathcal{O}(N^2)$, while N is the total amount of incoming sampled points of the signal.

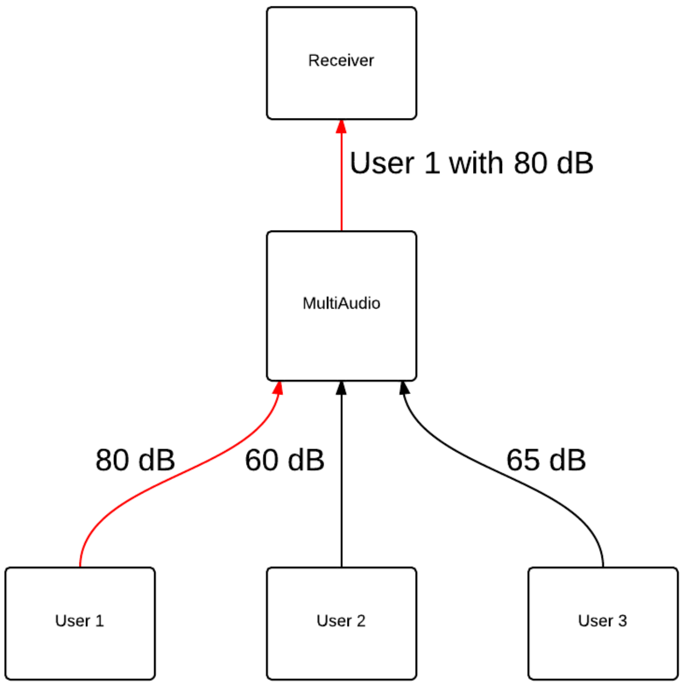

Figure 4.3.: An example of to approach with the highest signal: The signal of user 1 is dominant because he is louder than the other two user.

The fast Fourier transform has the same functionality as the discrete Fourier transformation but the complexity is different.

The fast Fourier transform computes the frequencies of an incoming signal in $\mathcal{O}(n \log n)$. For large data (or long signals, which have many different frequencies) this is a great speed effort. The formula for the discrete Fourier Transform is as follows:

- $f_p = \frac{1}{\sqrt{M}} \sum_{m=0}^{M-1} f_m e^{-\frac{i2\pi m}{M}} for(p = 0, 1, \ldots, M-1)$[Smi07]

As mentioned above, this procedure of the Fourier transform takes the complexity $\mathcal{O}(N^2)$. In contrast to the discrete Fourier transformation, the fast Fourier transform has the complexity $\mathcal{O}(n \log n)$.

The fast Fourier transform is a divide-and-conquer algorithm, which splits the problem size M in $\frac{M}{2}$. The algorithm continues to split the problem size M recursively until the problem size 1 is reached. Then the discrete Fourier transform method is used. Separating the data structure M has the complexity $\mathcal{O}(\log n)$. Calculation the transformation on the separated data structures has the complexity $\mathcal{O}(n)$.

A big disadvantage of the fast Fourier transform is the requirement of a data size, which is a power of two. If this algorithm is used to analyse a frequency, the system using the algorithm can't be scaled efficiently.

The result is the complexity $\mathcal{O}(n \log n)$ [Ber69]. The fast and discrete Fourier transformation can be used for analysing signals such as sound signals. In Figure 4.4 a signal of hearbeat is shown. In Figure 4.5 the fourier spectra of Figure 4.4 is shown.

The fourier spectra allows to determine the loudness trend of an incoming signal. With this spectra compared to the other incoming spectra a determination of the loudest signal is allowed. Furthermore, it is possible to determine which client is continuous loud. This is important to determine "short loudness" e.g. persons who cough can be loud too and can be determined as a dominant speaker by a naive algorithm [GME11]. The Fourier transformation can determine the trend of the loudness. Small loud gaps of quiet clients can so be ignored.

Figure 4.4 shows a heartbeat, which decibel are going continuously down. The analysis of the Fourier spectrum (Figure 4.5) shows a trend, which is going continuously down. The spectra showed that the graph of Figure 4.4 has a constant falling decibel. The Fourier transformation is also a great method for analysing signals.

4.3. Choice of Algorithm

The three mentioned algorithm allow the filtering of noise and determining the dominant speaker. Besides that, the algorithm have their disadvantages:

- **Low- and Highpassfilter:** Every time a filter is performed on a continuous signal, it looses some informations. In sound processing a false configured constant can have fatal results: Besides of the fact, that the filter already destroys informations, a false constant can destroy the whole recording at instance [TC02].
- **Loudest Voice:** The loudest voice algorithm is the most simple algorithm to implement. The disadvantages are the problems with real time sound processing. If a person coughs, the person can shortly be the loudest person in the conversation. This will result in a short term dominant signal for the coughing person. Another problem is the normalisation of the input devices, since the input devices can have different sensitive recording levels.
- **Fast Fourier Transformation:** The fast Fourier transformation offers a fast transformation in the Fourier spectrum. In this Fourier spectrum, high frequencies and high amplitudes can be cut off to quite sender. The fast Fourier transformation has the complexity $\mathcal{O}(n \log n)$, which can be transformed nearly in real time. The disadvantage of the fast Fourier transformation is the need of data length in the power of two [Wel67]. This makes this method useless in real time sound processing.

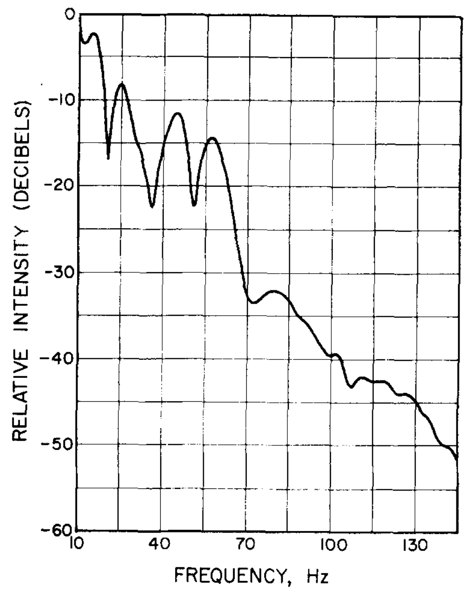

Figure 4.4.: Frequency and decibel of a heartbeat [YGU+76].

The low- and highpassfilter method requires many computation steps, like smoothing hard edges before filtering, the filtering itself etc. . Furthermore, many informations at high frequencies are destroyed, which still are audible for the human ear.

The fast Fourier transform requires a computation time of $\mathcal{O}(n \log n)$ and the amount of data must be a power of two. Furthermore, an algorithm has to be developed which makes a correct Fourier analysis of the continuous incoming Fourier spectra. An algorithm with linear computation time and with independence of the incoming amount of data is still better.

Therefore, the algorithm for determining and piping the loudest voice was choosen as the implementation for the audio toolkit. The algorithm offers the possibility of an implementation with linear computation time.

4.4. Development

The main part of this thesis is the development of the audio toolkit with the functionalities described above. Only the algorithm for determining the client with the loudest sound is developed. An overview of the implementation approach can be found in Figure 4.6

4.4.1. Java Sound APIs

The audio toolkit processes time critical operations, so the audio toolkit has to process the sound fast. The choice of a good Java Sound API is a crucial part of the development of the audio toolkit. The Java Sound API has to meet the following requirements:

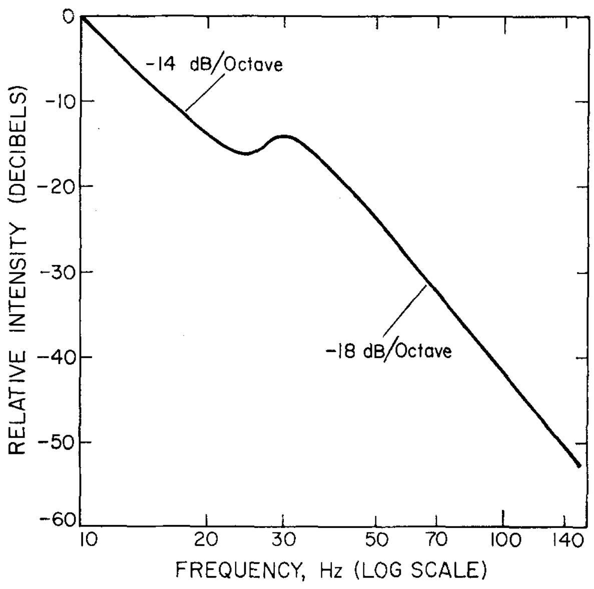

Figure 4.5.: Fourier spectra of figure 4.4 [YGU+76].

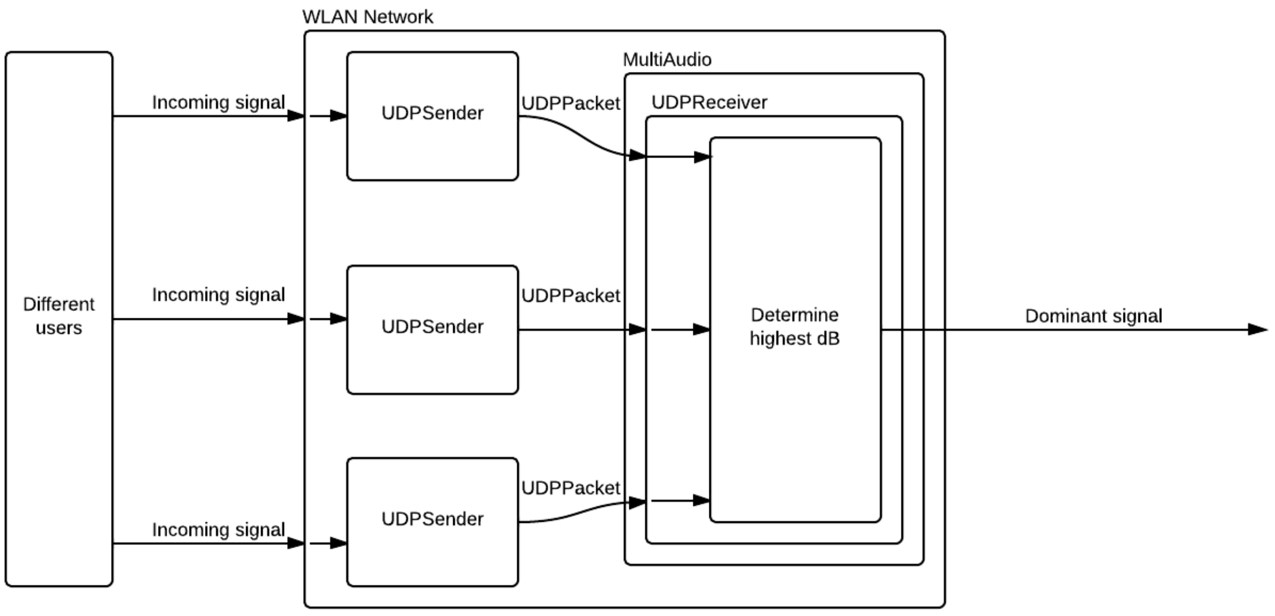

Figure 4.6.: Implementation design of the audio toolkit.

- Efficient sound processing.
- Pure Java implementation.
- Good support for the internal Java Sound API (important for writing streams in files etc.).
- Efficient garbage collection.

The following APIs were tested:

- Tritonus: Open Source Java Sound [tri].
- Advanced Linux Source Architecture [als].
- Java Sound API (integrated in the JDK 1.7) [jav].
- The Beads Project - Realtime Audio for Java and Processing [bea].
- FMOD - A C library with a Java wrapper (called NativeFmodEx) [fmo] [nat].

The Tritonus library offers good support for the combination with the Java Sound API, because Tritonus itself is pure Java and represents a wrapper for the functions Java already contains. Working together with these two libraries allows to produce powerful sound processing applications.

The ALSA library is not compatible with Microsoft Windows and is only Linux compatible. On only Linux systems ALSA is the best choice for sound processing tasks.

The Beads Project offers a lot of great functionalities. Unfortunately, these functionalities aren't compatible with the Java Sound API.

FMOD is a C library, which has a Java wrapper library called NativeFmodEx. Due to the fact that the project has no maintenance (last update in the Year 2005) and it needs the FMOD C library always linked to the project, FMOD wasn't the first choice for this project [Sti96].

Because of the advantages mentioned above, the choice falls on the integrated Java Sound API and Tritonus.

4.4.2. Sending and recording signals

One device can only select one microphone to transmit the audio to the host, on which the audio toolkit is deployed. It's a hardware limit to let two microphone simultaneously record an audio signal.

Furthermore, most operating systems doesn't allow to connect more than one bluetooth headset to one bluetooth dongle or to install more than one bluetooth dongle on one operating system, such as Microsoft Windows 7.

In case of these problems an Android application was developed which has the functionality to transmit the recorded audio via UDP to the host. The audio toolkit contains this functionality too, so a regular PC with a recording device is able to send the recording data via UDP.

The recorded sound signal will be decomposed into its bytes and assigned to a 2210 bytes big array. From these 2210 bytes 2200 are reserved for the sound bytes. The last 10 bytes are reserved for an id which the user can choose. By this id every sender of the UDP data can be recognized. This 2210 bytes big array will be send to a host, on which the audio toolkit is listening to a specific port.

4.4.3. Receiving data

On the host the data sent by the sender will be received. The receiver gets the 2210 bytes big packages and separates them into the 10 bytes big id and into the 2200 bytes big audio data.

From these incoming audio data, the decibel will be calculated. Since decibel, short dB, is a relative and logarithmic metric, the audio toolkit calculates it with the logarithmic formula $decibel = |SmoothConstant * (\log(byte_i[j]))|$, while the $SmoothConstant$ is set to 23.5 [FM33]. The sender with the highest decibel will be saved and the audio data with the id will be transmit to the destination. The receiver saves the id and compares each incoming package with the decibel of the dominant id.

If a package with higher decibel and another id than the actual dominant id appears, the dominant id and the dominant decibel will be reset with the new values and piped again to the destination.

4.4.4. Implementation Details

The implementation of the audio toolkit exists as a desktop version and an Android version. The implementation for a regular computer has the ability to take the role as a host and as a sender. The desktop application can also receive and send audio data. The Android version for smartphones can send audio data to a host, but can not be a host itself.

If the desktop application is receiving data, it's listening to a port specified in the graphical user interface. It calculates the out of the incoming data the loudest sound and pipes it to a file or to a sound system.

If the desktop or Android application is sending data, it sends data to IP address and port specified in the graphical user interface. The deskop application is able to figure as a host and sender simulteanously. An architecture is available at figure 4.7.

4.5. Integration

An installation of the actual Java 1.7 is necessary to run the audio toolkit. The audio toolkit is only availabe for the operating systems Microsoft Windows and Linux. Support for Mac OSX doesn't exist, since the internal Java SWT libraries for Mac are not included in the jar and are not tested.

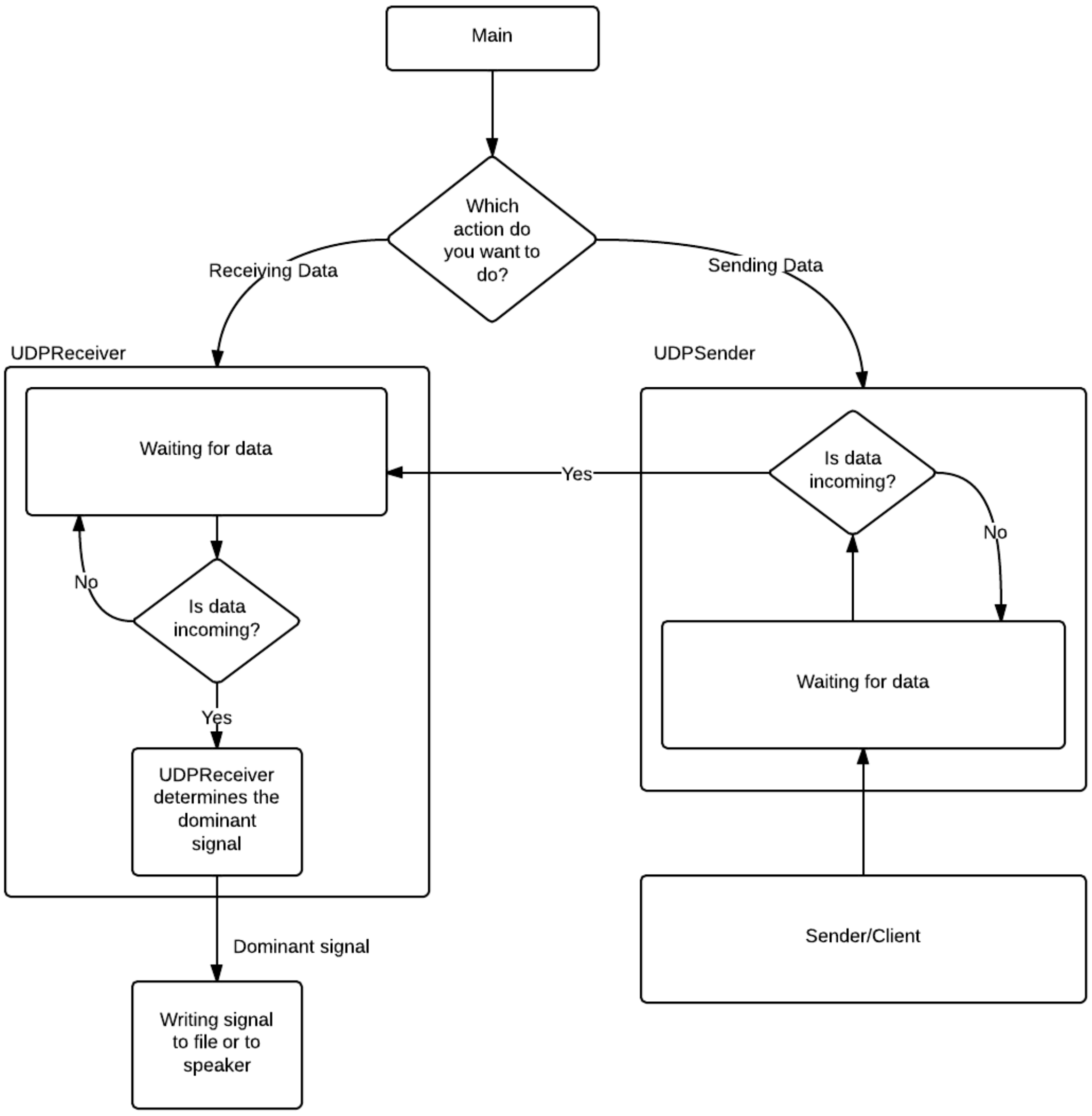

Figure 4.7.: Architecture of the implemented audio toolkit.

5. Evaluation

For the evaluation the Apparatus-Procedure-Results technique is used.

5.1. Userstudy

Five participants should read a text loud. At random intervals, the actual reader was interrupted by another participant. The whole recording is saved as a WAV file. The result can be analysed by the WAV file. For the user study, a self written role text was read aloud. The role text was written for five persons (see appendix A.1).

The five paricipants has to read the text twice. One time without any artificial disturbances and one time with artificial disturbances. Both recordings are saved as a WAV file, which is required for further analysis after the study.

5.2. Participants

For an good analysis of the study it was necessary to gather important informations about every participant. Every participant is male and the age was between 21 and 25. Four participants are studying Software Engineering, while one participant ist studying law. Three participants used the desktop application, while the other two participants used the Android application.

5.3. Apparatus

All participants are in one room and each participant gets a bluetooth headset. These bluetooth headsets are connected to an android phone or to a computer. Furthermore, every participant gets a role to speak. An extra participant has the exercise to make acoustic disturbances, for example coughing or sneezing. These disturbances should be not louder than the person that speaks. The recorded data will be send to a host, which listens to all incoming devices. The data will be analysed after the study.

5.4. Procedure

The procedure has two parts:

- The first part is reading their role without any artificial disturbances. This recording is necessary for later analysing.
- The second is reading their role with artificial disturbances. This recording is necessary for comparing it with the recording of the first part.

In both parts the participants read their text. Their readings will be recorded as a WAV file on a persistent medium, one file without artificial disturbances and one file with artificial disturbances. Both recordings are necessary for the analysis of the audio toolkit and how good it has filtered the disturbances.

5.5. Conclusion of the Evaluation

This section describes the result of the audio toolkit, compared with a recording with artificial disturbances and a recording without artificial disturbances.

5.5.1. Functionality of the Audio Toolkit

The evaluation produced two recordings. One with artificial disturbances and one without artificial disturbances. These two compared, obtained the following interesting results:

- Disturbances were filtered nearly completely. Disturbances, which were louder than the speaker, weren't filtered properly.
- Very often participants became fast dominant in very short time intervals. This is due to the fact, that every person was in the same room while the user study.

  The processing of the dominant signal is calculated on every device in the room and gives an output. The result is a fast switch of the speaker, which makes it hard to understand what is spoken.

5.5.2. Result of the Participants

Every participant was able to use the desktop or Android application without any problems. The participants wanted a help function, which explains the basic functionality of the audio toolkit. Furthermore, the partipants missed a automated microphone normalisation function. The particpants found it very difficult to configure each device equally sensitive.

The integration into other sound services like Skype, Teamspeak or the Microsoft Speech API is by the participants desired. This function has to be selectable, for example with a checkbox. A permament activation of the audio toolkit into sound services wasn't appreciated as well.

5.5.3. Results

The evaluation shows, that many details have to be implemented before the audio toolkit becomes really useful in daily use.

Every participant wished a functionality like the audio toolkit in VOIP or voice recognition services and every participant has already knowledge with such services.

Some extensions, like the normalisation have to be implemented to enable a clearly recording of the dominant signal. If user of the toolkit are in the same room, they overlap themselves because every device records every persons voice. This causes difficult audible recordings or streams.

6. Limitations

This chapter lists the limitations discovered while writing the thesis. These restrictions were validated while writing the thesis and testing the audio toolkit.

6.1. Maximal Amount of Bluetooth Headsets

The amount of the simultaneously connected bluetooth headsets is by design limited to five devices. It's easy to expand the amount of the bluetooth headsets simultaneously connected to the audio toolkit.

Furthermore it's only possible to connect one bluetooth headset to one device. If more clients want to connect, more devices and bluetooth headsets are necessary. Supported devices are regular PCs or Android phones matching the operating system requirement mentioned above.

6.2. Android Application

An optional Android application was developed for the connection to the host. The application was developed with the Android SDK in pure Java.

The application supports sending of live recorded sound via UDP to a host, on which the audio toolkit is deployed. For every further connection to a host, an android device and the developed Android application is necessary.

6.3. Network Connection

Every device which wants to connect to the audio toolkit has to be in the same wireless network like the host. Important is the correct IP address and port of the host, on which the audio toolkit has to be started and listening too.

6.4. Normalisation of Input Devices

Every device can be configured with a different recording intensity. Some devices can also act more sensitive to waveforms than other devices. This means, in focus of the audio toolkit, that the audio toolkit determines some devices louder than another even when all people are talking equally loud.

Normally, the intensity of the recording can be configured manually. This would require a perfect configuration of every device. This costs much time and requires a lot of configuration steps.

One way to avoid this is the normalisation of the input devices. The normalisation sets every device equally sensitive. Every waveforms are interpreted equally sensitive.

Due to the complexity of this task, the normalisation of the input devices isn't implemented. An idea is to play a reference audio to all input devices. The audio toolkit has to save the sensitivity of the input devices and scale them continuously.

6.5. Noise

Noise in sound recordings presents a big problem for speech recognition sound systems [HE95]. Several experiments have shown, that even very small amounts of noise can disturb a whole speech recognition system.

The audio toolkit isn't able to filter noise out of continuously incoming audio signals. Noise filtering can't be performed without loosing audio quality. The dominant audio signal can be interpreted false.

Besides that, due to the complexity of implementing a noise filter without loosing any audio quality, noise reduction wasn't implemented in this thesis.

7. Conclusion and Outlook

This thesis explained how sound processing can be performed with program languages like Java. Furthermore an audio toolkit was developed which processed incoming audio signals, mostly recorded via bluetooth headsets, and determined the loudest signal to suppress other speakers, which were not that loud as the loudest speaker.

As the audio toolkit was developed, an user study was performed to measure the functionality of the audio toolkit and to discover restrictions or limitations. A full list of restrictions can be found in the chapter Limitations.

7.1. Working with the Java Sound API

The Java Sound API offers a great potential for audio processing. Additional libraries and ready implemented packages are available on the Internet. A great source is JSResources, which cover many examples for the Java Sound API [jsr] [Esp04].

The best practice is the usage of libraries, which are implemented in pure Java. Otherwise it becomes difficult to integrate own implemented program parts with the modules of the Java SDK. Another problem are dependencies, like C libraries which must be linked to the project or jar file. Therefore, it's recommended to use the Java SDK as much as possible to avoid rendundancies with other external components.

It's highly recommended to use the JDK 1.7 for sound processing. One reason are the general security bugs Java has fixed [LL05]. Another reason is the implementation status of the different JDKs. The Java Sound API is fully implemented since the JDK 1.5 [Esp04]. More details of the Java Sound API and sample implementations can be found on www.jsresources.org [jsr].

7.1.1. Deployment on a Server

The audio toolkit can be deployed on a server for mass usage. Actually the audio toolkit only supports maximal five user, but more than five user are still possible. Like mentioned in the chapter "Development Process of the Audio Toolkit", the audio toolkit uses a sampling rate of 8000 Hertz with one channel and 16 bits. This takes 64 kb/s bandwidth, which corresponds ISDN speed. Today, this is a very small bandwidth. Since the audio toolkit is implemented in Java, the audio toolkit can be deployed on a Server running Linux or Windows Server. The only requirement is the installation of the JDK 1.7.

7.2. Using the Audio Toolkit in Sound Studios

The audio toolkit was generally developed for a limited amount of users. For that reason, a small sampling rate and a regular bit size was chosen. This allows to increase the amount of users without a great burden to the bandwidth.

The audio toolkit can be used in audio studios as well, but not in this state like it is now. Some details, like increasing the sampling rate, have to be changed to make it usable in audio studios.

7.3. Integration into Skype

Another possibility or improvement is the integration into Skype [skyc]. The audio toolkit is able to "catch" all corresponding signals and determine the loudest one. The loudest one can be piped to all other users in the actual skype conference.

This would reduce noise of other users in the conference and only one person would be audible to other people. This functionality isn't implemented, but can easily integrated by piping the dominant signal into the microphone Skype uses or using the Skype developer API [skyb].

7.4. Future Work

The audio toolkit still doesn't cover a full functional product, which can be used in professional sound processing. Many funcitons, like noise filtering or allowing more than five clients simultaneously to connect tho the host.

7.4.1. Improving the User Interface

An useful extension is a better integration of the pipe source of the incoming dominant signal. The UI can be modified, such that it recognises active VOIP or speech recogniton services. The user can then choose the output source. This would ease the use of the audio toolkit, if it has more than one output source.

7.4.2. Increasing the maximum amount of simultaneously connected User

Increasing the maximum amount of simultaneously connected clients is a great extension for server deployment. User can connect to a server on which the audio toolkit is deployed and is listening at a specific port. This use case can be used for mass usage with many people int a room. This requires the extensions mentioned above.

7.4.3. Noise reduction

Noise reduction is a very crucial topic for speech recogniton applicatons. Even small noise artifacts in recordings can make a speech recognition useless. An integrated noise filter for the incoming signal can improve the audio quality rapidly. An algorithm for filtering noise out ouf sound still has to be developed. The difficulty lies in the substitution of the corrupt byte by a byte which fits into the sound pattern.

7.4.4. Normalisation

An another interesting improvement is the normalisation of the input devices. Without normalisation every input device has a different sensitivity. The result are different reacting input devices, which easily can transmit the wrong dominant signal. A possible solution for avoiding this, the devices can be normalised by playing a reference sound at every device. Every device can then configure its sensitivity automatically.

Normalisation has already been developed for hearing aid systems [Cor01].

7.5. Outlook

With increasing amount of using VOIP services like Skype or Teamspeak and in voice recognition systems like Apples Siri, the interest in sound processing toolkits gains much more interest. The interest comes with the ease of use with such systems and will gain more potential and usage in the future (see Figure 7.1).

In order of that, more audio toolkits or sound processing tools will appear in near future, which will ease the use of sound systems.

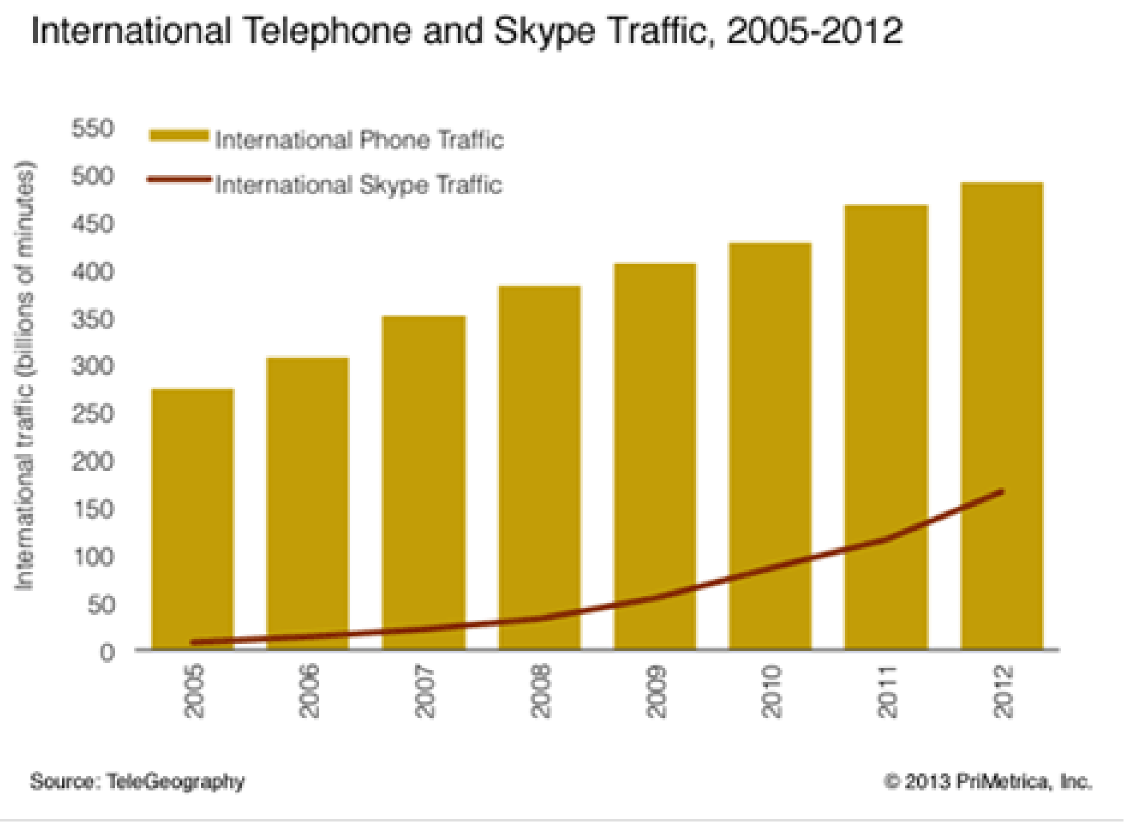

Figure 7.1.: Usage of regualar phones and the VOIP service Skype. Skype holds an increasing usage trend [skya].

Bibliography

[als] Advanced Linux Sound Architecture. URL http://alsa-project.org/. (Cited on page 32)

[BE47] H. S. Black, J. Edson. Pulse code modulation. *American Institute of Electrical Engineers, Transactions of the*, 66(1):895–899, 1947. (Cited on page 17)

[bea] The Beads Project - Realtime Audio for Java and Processing. URL http://www.beadsproject.net/. (Cited on page 32)

[Ber69] G. Bergland. A guided tour of the fast Fourier transform. *Spectrum, IEEE*, 6(7):41–52, 1969. (Cited on page 29)

[BME$^+$99] V. L. Beattie, D. R. Miller, S. E. Edmondson, Y. N. Patel, G. A. Talvola. Multi-dialect speech recognition method and apparatus, 1999. US Patent 5,865,626. (Cited on page 11)

[CKL$^+$04] L.-J. Chen, R. Kapoor, K. Lee, M. Sanadidi, M. Gerla. Audio streaming over Bluetooth: an adaptive ARQ timeout approach. In *Distributed Computing Systems Workshops, 2004. Proceedings. 24th International Conference on*, pp. 196–201. IEEE, 2004. (Cited on page 12)

[Cor01] L. E. Cornelisse. Software implemented loudness normalization for a digital hearing aid, 2001. US Patent App. 09/985,976. (Cited on page 43)

[CW08] E. J. Candès, M. B. Wakin. An introduction to compressive sampling. *Signal Processing Magazine, IEEE*, 25(2):21–30, 2008. (Cited on page 17)

[DJ04] C. Draxler, K. Jänsch. SpeechRecorder-a Universal Platform Independent Multi-Channel Audio Recording Software. In *LREC*. 2004. (Cited on page 24)

[Esp04] M. R. R. Esparza. *Java Sound ALS Plattform FÜR DIE Entwicklung STUDIO-TAUGLICHER Audioapplikationen*. Ph.D. thesis, Master's thesis, Fachhochschule Stuttgart, 2004. (Cited on pages 16 and 41)

[FM33] H. Fletcher, W. A. Munson. Loudness, its definition, measurement and calculation. *The journal of the acoustical society of America*, 5(2):82–108, 1933. (Cited on page 33)

[fmo] FMOD - A C library with a Java wrapper. URL http://www.fmod.org/. (Cited on page 32)

[GME11] B. Gold, N. Morgan, D. Ellis. *Speech and audio signal processing: processing and perception of speech and music*. Wiley. com, 2011. (Cited on page 29)

[HE95] H. Hirsch, C. Ehrlicher. Noise estimation techniques for robust speech recognition. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, pp. 153–156. IEEE, 1995. (Cited on page 40)

[HM03] I. Hossain, Z. Moussavi. An overview of heart-noise reduction of lung sound using wavelet transform based filter. In *Engineering in Medicine and Biology Society, 2003. Proceedings of the 25th Annual International Conference of the IEEE*, volume 1, pp. 458–461. IEEE, 2003. (Cited on page 11)

[HP00] H.-G. Hirsch, D. Pearce. The Aurora experimental framework for the performance evaluation of speech recognition systems under noisy conditions. In *ASR2000-Automatic Speech Recognition: Challenges for the new Millenium ISCA Tutorial and Research Workshop (ITRW)*. 2000. (Cited on page 11)

[jav] Java Sound API (integrated in the JDK 1.7). URL http://docs.oracle.com/javase/tutorial/sound/. (Cited on page 32)

[jsr] JSResources. URL http://www.jsresources.org/. (Cited on page 41)

[KH99] R. Kirk, A. Hunt. *Digital Sound Processing for Music and Multimedia*. Music Technology, 1999. (Cited on page 17)

[KT97] A. Kroch, A. Taylor. Verb movement in Old and Middle English: Dialect variation and language contact. *Parameters of morphosyntactic change*, pp. 297–325, 1997. (Cited on page 11)

[LL05] V. B. Livshits, M. S. Lam. Finding security vulnerabilities in Java applications with static analysis. In *Proceedings of the 14th conference on USENIX Security Symposium*, volume 14, pp. 18–18. 2005. (Cited on page 41)

[Llo82] S. Lloyd. Least squares quantization in PCM. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982. (Cited on page 18)

[MBTY03] M. Mathew, V. Bhat, S. M. Thomas, C. Yim. Modified MP3 encoder using complex modified cosine transform. In *Multimedia and Expo, 2003. ICME'03. Proceedings. 2003 International Conference on*, volume 2, pp. II–709. IEEE, 2003. (Cited on page 26)

[MSP96] MSP. How Digital Audio Works. *cycling*, 1996. (Cited on page 16)

[nat] NativeFmodEx. URL http://jerome.jouvie.free.fr/nativefmodex. (Cited on page 32)

[OP89] C. Offelli, D. Petri. Interpolation techniques for real-time multifrequency waveform analysis. In *Instrumentation and Measurement Technology Conference, 1989. IMTC-89. Conference Record., 6th IEEE*, pp. 325–331. IEEE, 1989. (Cited on page 17)

[SFB00] V. Stahl, A. Fischer, R. Bippus. Quantile based noise estimation for spectral subtraction and Wiener filtering. In *Acoustics, Speech, and Signal Processing, 2000. ICASSP'00. Proceedings. 2000 IEEE International Conference on*, volume 3, pp. 1875–1878. IEEE, 2000. (Cited on page 11)

[Sha49] C. E. Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, 1949. (Cited on page 17)

[skya] Interntational Skype and telephone grafic. URL http://www.trutower.com/wp-content/uploads/2013/02/Skype-VoIP-Research.png. (Cited on pages 7 and 44)

[skyb] Skype Developer API. URL https://developer.skype.com/. (Cited on page 42)

[skyc] Skype Web Site. URL http://www.skype.com/en/. (Cited on page 42)

[Smi07] J. O. Smith. *Mathematics of the Discrete Fourier Transform (DFT): With music and audio applications.* Julius Smith, 2007. (Cited on page 28)

[Sta03] Stanford. WAVE PCM soundfile format. *Stanford*, 1(1):1, 2003. (Cited on page 25)

[Sti96] L. J. Stifelman. Augmenting real-world objects: A paper-based audio notebook. In *Conference companion on Human factors in computing systems*, pp. 199–200. ACM, 1996. (Cited on page 32)

[SZZ+11] R. Schlegel, K. Zhang, X.-y. Zhou, M. Intwala, A. Kapadia, X. Wang. Soundcomber: A Stealthy and Context-Aware Sound Trojan for Smartphones. In *NDSS*, volume 11, pp. 17–33. 2011. (Cited on page 11)

[TC02] G. Tzanetakis, P. Cook. Musical genre classification of audio signals. *Speech and Audio Processing, IEEE transactions on*, 10(5):293–302, 2002. (Cited on page 29)

[tri] Tritonus: Open Source Java Sound. URL http://www.tritonus.org/. (Cited on page 32)

[wava] Graphical illustration of the WAV format. URL https://ccrma.stanford.edu/courses/422/projects/WaveFormat/wav-sound-format.gif. (Cited on pages 7 and 25)

[wavb] How do WAV and MP3 files differ. URL http://www.dawsons.co.uk/blog/how-do-mp3-and-wav-files-differ. (Cited on pages 7 and 26)

[Wel67] P. Welch. The use of fast Fourier transform for the estimation of power spectra: a method based on time averaging over short, modified periodograms. *Audio and Electroacoustics, IEEE Transactions on*, 15(2):70–73, 1967. (Cited on page 29)

[YGU+76] A. Yoganathan, R. Gupta, F. Udwadia, J. Wayen Miller, W. Corcoran, R. Sarma, J. Johnson, R. Bing. Use of the fast fourier transform for frequency analysis of the first heart sound in normal man. *Medical and biological engineering*, 14(1):69–73, 1976. doi:10.1007/BF02477093. URL http://dx.doi.org/10.1007/BF02477093. (Cited on pages 7, 30 and 31)

All links were last followed on October 22, 2013.

A. Appendix

A.1. Text for Evaluation

Paul:	Hallo Peter. Lange nicht mehr gesehen!
Peter:	Paul, du altes Haus wie geht es dir?
Paul:	Alles was Beine hat. Ich wollte gerade mit Hugo in die Eisdiele.
Peter:	Keine schlechte Idee bei dem heißen Wetter. Kann ich euch begleiten?
Paul:	Finde ich gut!
Hugo:	Hallo Paul. Hast du den Peter auch gleich mitgebracht?
Paul:	Der ist mir nachgelaufen, darf ich ihn behalten?
Peter:	He!
Hugo:	War doch nur ein Scherz. Lasst uns endlich ein Eis bestellen.
Hugo:	Hallo. Ich hätte gerne eine Kugel Erdbeere und eine Kugel Himbeere.
Herbert:	Na wen haben wir denn da!
Gerhard:	Wenn das nicht Hugo ist!
Hugo:	Seit wann arbeitet ihr denn hier?
Peter:	Wollt ihr noch was dazuverdienen?
Gerhard:	Eher weniger. Wir werden dafür nicht bezahlt.
Paul:	Und was macht ihr dann hier?
Herbert:	Wir wurden beim entwenden fremden Eigentums erwischt.
Hugo:	Und dann hat man euch erwischt?
Gerhard:	Natürlich wurden wir erwischt! Sonst wären wir ja nicht hier.
Peter:	Wie lange müsst ihr denn hier eure Strafe verbüßen?
Herbert:	160 Sozialstunden pro Person.
Paul:	Na dann habt ihr ja ordentlich zu arbeiten.
Peter:	Mein Mitleid hält sich allerdings in Grenzen.
Hugo:	Die Folgen von der Autoschieberei hätte man sich davor überlegen sollen!
Gerhard:	Es tut uns auch Leid. Aber jetzt müssen wir weiterarbeiten.
Peter:	So, ich muss dann mal Heim. Muss noch Mathematik fertigmachen.
Hugo:	Geht mir auch so. Muss noch verstehen, wie man Wronskideterminanten berechnet.
Paul:	Und ich muss noch meine Bachelorarbeit fertigschreiben. Bis bald und vielen Dank fürs mitnehmen.
Peter:	Tschüs.
Hugo:	Tschüs.

A.2. Used Tools

The following tools were used for the developing of the audio toolkit:

- Eclipse SDK (for developing in Java).
- Android SDK (for developing the additional Android app).
- Audacity (for comparing and analysing recorded WAV files).
- Version control was performed with Git.

Decleration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature