The intelligent stethoscope
– Wireless signal communication –

Marc Boher Massaguer

2010-09-15
LiTH-IMT/ERASMUS-R-- 10/37-- SE
Acknowledgements

First, I want to thank my supervisor, Peter Hult, for giving me the opportunity to do this thesis. I really liked the idea of this project and this is what made me continue with hope, even if at the start it looked like it was not going to be possible to get successful results. In the same way, I also want to mention Bengt Ragnemalm for his help every time I had trouble with the Bluetooth device.

During this experience here in Linköping I’ve had good and not so good moments but, overall, it can only be qualified as amazing. That was possible thanks to all the friends I met here. Names like Asn, Hélène, Hang, Ema, Yasin, Pierluca, Giulia, David, Anna, Sujit, Berta, Joana (who also was the volunteer for the heart auscultation), Natalia, Fabrizia, Arturo, Thibaut and Ricardo come to my mind right now, with no special order. Thank you guys, I’ll miss you all!!

I also want to thank my friends from Barcelona. I’m sorry we barely kept in touch with some of you while I was in Erasmus, but you know that I missed you (but just a bit, haha). Robert, as you guessed, that doesn’t apply to you because we spoke almost every day! But I wouldn’t say we spent too much time because I really enjoyed the conversations, even if they had to be totally meaningless, haha. Special thanks are also for you, my sister and Ruth for the “moral support” when things were tough ;).

All thanks would be worthless without mentioning my family. Thank you all for letting me have this experience and supporting me even from the distance. Gràcies pare, mare, Mire... Alpha! I com no, àvia, tiet Lidia, Laura, Eva, tiet Maties i tiet Mercè. Un petonàs a tots!! Tack så mycket!!
Abstract

In this thesis we want to record a heart or lungs auscultation and send it to a hospital located anywhere in the world. To do so, we will use an electronic stethoscope, a custom made Bluetooth device and a mobile phone. We create a mobile phone program using J2ME language, which will handle the connection with the Bluetooth device, will set it up, record the signal, process it and send it through e-mail. Finally, we validate the received signal to make sure the whole process is successful.
Contents

1 Introduction ............................................. 12
  1.1 Motivation ........................................ 12
  1.2 Objectives and aims ............................. 13
  1.3 Thesis Outline .................................. 14

2 Background ............................................ 16
  2.1 The stethoscope ................................ 16
  2.2 Related works .................................. 18

3 The project ........................................... 20
  3.1 Usage and description of the elements used .......... 20
    3.1.1 Patient’s house ............................. 21
    3.1.2 Hospital ..................................... 21
    3.1.3 Digital stethoscope ......................... 22
    3.1.4 Bluetooth device ............................ 23
    3.1.5 Mobile phone ............................... 24
  3.2 Feasibility study of the project .................. 24
    3.2.1 Technical feasibility ....................... 24
      3.2.1.1 Hardware ............................... 25
      3.2.1.2 Software ............................... 28
    3.2.2 Economical feasibility ..................... 29
3.2.2.1 Hardware ... ... ... ... 29
3.2.2.2 Software ... ... ... ... 29

4 Implementation and results ... ... ... ... 30
4.1 Implementation ... ... ... ... ... ... ... ... ... ... ... ... ... 30
4.1.1 Interface ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... 31
4.1.2 Connection and configuration of the device ... ... ... ... ... ... ... 40
4.1.3 Recording of the source ... ... ... ... ... ... ... ... ... ... ... ... 43
4.1.4 File processing ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... 44
4.1.5 Setup and internal storage ... ... ... ... ... ... ... ... ... ... ... ... 46
4.1.6 About e-mailing the recorded file ... ... ... ... ... ... ... ... ... ... ... 47
4.2 Results ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... 49
4.2.1 Heart auscultation ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... 49
4.2.2 Validation of the signal ... ... ... ... ... ... ... ... ... ... ... ... ... ... 50
4.3 Found bugs ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... 54

5 Conclusions and future work ... ... ... ... ... ... ... ... ... ... ... ... ... 56
5.1 Conclusions ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... 56
5.2 Future work ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... 57

A J2ME Code ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... 60
A.1 Important functions that appear in this thesis ... ... ... ... ... ... ... ... 60
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Acoustic stethoscope</td>
<td>17</td>
</tr>
<tr>
<td>2.2</td>
<td>Electronic stethoscope</td>
<td>17</td>
</tr>
<tr>
<td>2.3</td>
<td>Fetal stethoscope</td>
<td>18</td>
</tr>
<tr>
<td>3.1</td>
<td>Overview</td>
<td>21</td>
</tr>
<tr>
<td>3.2</td>
<td>Welch Allyn Elite™</td>
<td>22</td>
</tr>
<tr>
<td>3.3</td>
<td>Bluetooth device</td>
<td>23</td>
</tr>
<tr>
<td>3.4</td>
<td>Sony Ericsson K610i</td>
<td>25</td>
</tr>
<tr>
<td>3.5</td>
<td>Type of connections</td>
<td>25</td>
</tr>
<tr>
<td>4.1</td>
<td>Application’s flowchart</td>
<td>32</td>
</tr>
<tr>
<td>4.2</td>
<td>Main menu</td>
<td>33</td>
</tr>
<tr>
<td>4.3</td>
<td>Connected screen</td>
<td>33</td>
</tr>
<tr>
<td>4.4</td>
<td>Can’t connect screen</td>
<td>34</td>
</tr>
<tr>
<td>4.5</td>
<td>Recording screen</td>
<td>34</td>
</tr>
<tr>
<td>4.6</td>
<td>Processing data screen</td>
<td>35</td>
</tr>
<tr>
<td>4.7</td>
<td>Setup menu</td>
<td>35</td>
</tr>
<tr>
<td>4.8</td>
<td>Confirmation screen</td>
<td>36</td>
</tr>
<tr>
<td>4.9</td>
<td>Search menu</td>
<td>36</td>
</tr>
<tr>
<td>4.10</td>
<td>Searching devices screen</td>
<td>37</td>
</tr>
<tr>
<td>4.11</td>
<td>Found devices screen</td>
<td>37</td>
</tr>
<tr>
<td>4.12</td>
<td>Searching services screen</td>
<td>38</td>
</tr>
<tr>
<td>4.13</td>
<td>Found services screen</td>
<td>38</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

4.14 File processing procedure . . . . . 45
4.15 Composing and sending the e-mail . 48
4.16 Received e-mail . . . . . . . . . 49
4.17 Example of a raw file . . . . . . . 50
4.18 Example of a processed file . . . . 50
4.19 Rested heart . . . . . . . . . . . 51
4.20 20 Hz Sinusoid, sampled at 960 Hz 52
4.21 100 Hz Sinusoid, sampled at 960 Hz 52
4.22 420 Hz Sinusoid, sampled at 960 Hz 52
4.23 20 Hz Sinusoid, sampled at 4800 Hz 53
4.24 100 Hz Sinusoid, sampled at 4800 Hz 53
4.25 420 Hz Sinusoid, sampled at 4800 Hz 53
4.26 480 Hz Sinusoid, maximum for a
960 Hz sampling speed . . . . . . . . 54
Nomenclature

API Application Programming Interface
BPM Beats Per Minute
BT Bluetooth
CPU Central Processing Unit
HTTP Hyper Text Transfer Protocol
HTTPS Hyper Text Transfer Protocol Secure
J2ME Java 2 Micro Edition
LSB Least Significant Bit
LSByte Least Significant Byte
MIDP Mobile Information Device Profile
MSB Most Significant Bit
MSByte Most Significant Byte
OS Operating System
SPP Serial Port Profile
URL Uniform Resource Locator
Chapter 1

Introduction

1.1 Motivation

The technique of listening to internal sounds, such as the circulatory and respiratory systems, has been used for many centuries. Through all the years it has been very important as a preliminary diagnosis, but maybe it has not evolved as much as other medical techniques. It still requires substantial clinical experience, a fine stethoscope and good listening skills. However, thanks to the evolution of electrical systems in the last decade, it has been possible to introduce electrical stethoscopes that make the listening a bit easier. The use of amplifiers and built-in filters limit the frequencies the doctor hears to just the ones that are important for the auscultated organ. Although this evolution is quite important, it is still common to see doctors using analog stethoscopes, proving that the technological leap occurred with the arrival of the digital era was not enough to make dramatic changes in this field.

As it’s been said before, the introduction of electronic stethoscopes improve the listening experience for the doctor. However, it also means that the auscultated sound is digitalized and, therefore, it is possible to process, edit or send easily. This is where the new technologies on communication play a decisive role: thanks to Internet and the mobile phone technologies, now it’s possible to establish a communication between two points located almost anywhere in the world.

Considering the situation that we put forward above, it is now technologically possible to record the sound of an organ of the body and send it somewhere else in the world, where doctors with experience in that field can analyze the data and give feedback if necessary. In other words, it will be possible to do an auscultation at the patient’s house without the need to have someone with listening skills and clinical experience, given that the recorded sound can be further analyzed by a doctor in the hospital. Moreover, the recorded sounds can be kept in the hospital databases and be used in the future to compare the old organ sounds with the new ones.
1.2 Objectives and aims

In LiU’s IMT department we want to focus on the idea explained in last section, trying to achieve a system that lets us do all the things mentioned there. According that these days everybody has a mobile phone, and those mobile phones can not only call, but also process data, we’ve decided that we’ll do a mobile phone program that will do all the required actions to send the stethoscope’s listened sounds to a hospital (i.e. Linköping’s hospital). The connection between the stethoscope and the mobile phone will be wireless, and it’ll be possible thanks to a Bluetooth device that will be directly plugged to a digital stethoscope.

This whole idea requires a lot of work, so it must be done in different stages. At first we will focus on the most important ones, the ones that without them it’s impossible to achieve its main function: to send an auscultated sound to a hospital. These stages are:

- Design and implementation of the Bluetooth (BT) module
- Design of the mobile phone’s program

Even if the above parts are the main ones, we must keep in mind that the function they have to do is, at least, to send and receive signals. Once the above is done, it will be possible to add other features like giving an interface to the program.

This thesis focuses in the second main stage: Design of the mobile phone’s program. However, to have a successful design of this program it’s indispensable to know how the Bluetooth device works, given that it has to communicate with it. As a result, this thesis also covers the functionality of the used devices, its discovered bugs and discusses if it was possible to fix them.

Here’s a list of functions that can be found in this program:

- Record auscultated sound sent by the BT device.
- Integrity check to see if the BT device is setup successfully.
- Use of different file names to avoid overwriting old recordings.
- Process received data in order to erase synchronization and control bits.
- Shut down BT device remotely.
- Use of internal storage to save settings between sessions.
- Settings menu with the following functions:
  - Search and link to Bluetooth devices so that it can connect with them.
  - Choose Bluetooth settings (i.e. encrypt, authenticate...)
  - Change program settings (i.e. folder and file name)
  - Delete all recorded files
- Save settings or just change without saving
- Load settings

Once the received sound is properly processed and recorded, this software was supposed to send this file to the hospital via e-mail. However, after some research, this idea was discarded for the reasons stated in section 4.1.6, page 47.

1.3 Thesis Outline

This thesis has been divided in 5 chapters. Chapter 2 explains what a stethoscope is and talks about a related work, while chapter 3 describes the elements used in this thesis and tries to see if what we want to do is economically and technologically feasible. In chapter 4 we describe the mobile phone program and we validate the results obtained with it. Finally, in chapter 5 we expose the conclusions of the thesis and suggest future work.
Chapter 2

Background

2.1 The stethoscope

A stethoscope is a diagnostic instrument used by medical professionals to listen to internal sounds of an animal body, usually a patient’s chest cavity, heart, different pulse points, and even intestines and blood flow in arteries and veins. It was invented by the French physician René Théophile Hyacinthe Laënnec, who is generally considered to be the father of chest medicine. Even if it can have multiple uses, doctors tend to use it only to listen for sounds of congestion in the lungs and irregular heartbeats. Nurses may also use stethoscopes to listen for restored blood flow during blood pressure checks[1]. It is commonly used as a preliminary diagnosis tool because of its ease of use, the fact that it’s a non-invasive examination procedure[5] and that you don’t have to wait for the results. However, it requires substantial clinical experience and good listening skills in order to get a diagnostic.

There are 3 types of stethoscopes: Acoustic, electronic and fetal.

Types of stethoscopes

**Acoustic** This type is the most famous one and also the oldest one (Figure 2.1). Operates on the transmission of sound from the chest piece, via air-filled hollow tubes, to the listener’s ears. The chest-piece usually consists of two sides that can be placed against the patient for sensing sound — a diaphragm (plastic disc) or bell (hollow cup). If the diaphragm is placed on the patient, body sounds vibrate the diaphragm, creating acoustic pressure waves which travel up the tubing to the listener’s ears. If the bell is placed on the patient, the vibrations of the skin directly produce acoustic pressure waves traveling up to the listener’s ears. The bell transmits low frequency sounds, while the diaphragm transmits higher frequency sounds[4, 6].

**Electronic** Electronic stethoscopes are slowly appearing in the market with some improvements every time, but they still have not been able to dis-
cuss the hegemony of the acoustic stethoscope. They use the advantages of the digital information to amplify, process and filter the sounds that are important for that matter. Because of that, they usually have some buttons to choose between a heart auscultation and a lungs one. This type of stethoscopes can also use the advantage of the digital data for recording the auscultation and reproducing it as if it was a music device. Unlike acoustic stethoscopes, which are all based on the same physics, transducers in electronic stethoscopes vary widely. The simplest and least effective method of sound detection is achieved by placing a microphone in the chest-piece. However, this method suffers from ambient noise interference and has fallen out of favor. They are also known as *stethophone* (Figure 2.2).

**Fetal** A fetal stethoscope or fetoscope is an acoustic stethoscope shaped like a listening trumpet. It is placed against the abdomen of a pregnant woman to listen to the heart sounds of the fetus. The fetal stethoscope is also known as a Pinard’s stethoscope or a pinard, after French obstetrician Adolphe Pinard (1844–1934) [4]. Figure 2.3.
2.2 Related works

With the introduction of electronic stethoscopes and the evolution of the technology of communications, the most obvious step is to take advantage of that and try to add wireless functionality to these stethoscopes. It is also straightforward to think that it’ll be really interesting to record the auscultated signal and send it to a hospital. Because of that, many universities, as well as companies, are trying to accomplish this objective. Right now it is possible to find stethoscopes that record the sounds into an internal memory, but right now the wireless communication part is still not common. However, there are already some works from universities that cover this idea. The following one is especially interesting:

**Integration of Bluetooth-enabled Sensors into E-Health Application for Home Healthcare and Monitoring**

This is a Master thesis project done by Agustín García Pérez, in the university of Borås (Sweden). It is focused in measuring blood pressure and weight from a patient using two Bluetooth enabled measurement devices. There is also an implementation of a web server used to store all patient reports with updated information in real-time, with the option to access this information through a web page. The implementation of a server like the one that is shown in Agustín García’s project would be very interesting, as it’s stated in Section 5.2 of this project.
Chapter 3

The project

3.1 Usage and description of the elements used

In order to accomplish the mission of the project, we will use a digital stethoscope with a Bluetooth device connected in it, and a mobile phone with special software created for that matter. The auscultating process is the following:

1. Connect with a cable the stethoscope to the Bluetooth device and turn both on.

2. Run the mobile phone’s software and press connect to initiate the connection.

3. Start the auscultation and press the record button when needed.

4. Finish the auscultation and press the stop button.

5. Wait the program to process the file.

6. Send the recorded file through e-mail, if needed.

The figure 3.1 shows the elements involved in it.
Figure 3.1: Overview

Legend

A Patient’s house
B Hospital
1 Digital stethoscope
2 Bluetooth device
3 Mobile phone

3.1.1 Patient’s house

The key part of this project is to be able to do an auscultation to a patient without having to be in the hospital. An ideal place would be, for example, the patient’s house. In that place we will have the digital stethoscope, a Bluetooth device and a mobile phone with special software installed in it.

3.1.2 Hospital

Here’s where the recorded measurement will be received. Thanks to Internet, there’s no limitation in distance between the hospital and the patient’s house as long as both have Internet connection. Inside the hospital there will be a server with a built-in database that will receive and process the recordings of all patients. Then, doctors in any computer terminal inside the hospital will
be able to access that database, visualize the files and proceed to give feedback if needed. However, this part is not necessary in this stage of the project and, therefore, it will not be covered in this thesis.

### 3.1.3 Digital stethoscope

All digital stethoscopes are valid as long as they have an analog audio output. In this thesis we will be using a Welch Allyn Elite™ stethoscope (figure 3.2), with the following specifications:

- Frequency range: 20 – 20,000 Hz
- Amplification: Adjustable. Up to 93 dB
- Weight: 170 g.
- Output: 2,5mm stereo jack
- Batteries: 1 x Lithium CR123A (expected battery lifetime is approx 240 hours)
- Modes of auscultation: Diaphragm and bell modes.

![Welch Allyn Elite™](image)

**Figure 3.2: Welch Allyn Elite™**

**Description of the auscultation modes**

**Diaphragm mode:** For lungs auscultation. When this mode is selected, the lungs symbol is illuminated. Excellent for higher pitch murmurs, clicks and ejection sounds as well as pulmonary sounds. It filters between the following frequencies: 350 – 1900 Hz.

**Bell mode:** For heart auscultation. When this mode is selected, the heart symbol is illuminated. It provides excellent detection for low and medium heart sounds and murmurs such as diastolic rumbles, gallop rhythms, etc. It filters between the following frequencies: 20 – 420 Hz.

---

1Full datasheet of this stethoscope can be found in Appendix ??
3.1.4 Bluetooth device

We will be using a prototype made in our department, the IMT. It is a custom made device (figure 23) that consists in a board with the following main components:

- Free2move F2M03AC2 Bluetooth module
  - Bluetooth version: 1.1
  - Class 2 (about 10 meters of range)
  - Compatible with Serial Port Profile (SPP)
  - Included antenna

- Analog to Digital Converter
  - 16 bit
  - 100 KS ()
  - 4 channel

- DAC (Digital to Analog Converter, right?)
  - 16 bit

- 2 Analog amplifiers, both are digitally gain programmable
  - General purpose Instrument Amplifier
  - General purpose amplifier stage

- 550 mAh lithium-polymer battery. Can be charged from USB-port.

![Figure 3.3: Bluetooth device](image)

To sum up, this is a multipurpose board that must be set up before its use, as it has to know which components we want to enable and how we want them to work. To do so, the mobile phone’s program will connect to the device and send some configuration bytes that are chosen according to the device protocol\(^2\).

\(^2\)There’s a brief description of the protocol in page. Complete protocol can be found in section ??.
Considering the objectives of this project, the main functions this device will do are the following:

- Digitalize the analog audio signal it receives.
- Protect data integrity against channel quality fluctuations, using an integrated buffer.
- Send that signal using its Bluetooth embedded device.

3.1.5 Mobile phone

This is the device that will host the program created in this thesis. In order to be compatible with it, it must have:

- Bluetooth wireless network connection
- Serial Port Profile (SPP) for Bluetooth
- Compatibility with Java 2 Micro Edition (J2ME)
- Compatibility with J2ME Profile Mobile Information Device Profile 2.0 (MIDP-2.0)
- Compatibility with Java Application Programming Interface (API) for Bluetooth JSR-82
- Internet connection.
- Minimum GPRS, but it’s recommended to have 3G or better.

All the procedures will be controlled from the mobile phone. It’ll decide when to record and when to finish the auscultation. It’ll save in its memory the recorded audio and it’ll also be the one that sends it to the hospital. With further work on the project, it will also be possible to receive feedback from the hospital.

The mobile phone we’ve chosen for this thesis is the Sony Ericsson K610i. It meets all the requirements mentioned before, supporting Bluetooth 2.0 and 3G technology, for faster Internet data connection.

3.2 Feasibility study of the project

Before start working on the project it’s important to know if our project is feasible technically and economically.

3.2.1 Technical feasibility

It depends on the hardware and the software used.
3.2.1.1 Hardware

We need an electronic stethoscope, a Bluetooth device and a mobile phone. Ideally the stethoscope will send the measured signal to the Bluetooth device, which at the same time will forward it to the mobile phone. Finally, the mobile phone will create a file and, once the recording it’s finished, it will send it to the hospital. Some of the connections we’ve analyzed need to send the information in real-time, while some of them don’t, as it’s shown in Figure 3.5.

Considering this, we have to be sure that everything that needs to be done in real-time is done correctly. Those things are:

- Communication between stethoscope and Bluetooth device
- Communication between Bluetooth device and mobile phone
- A/D Conversion in Bluetooth device
- Computing processes in mobile phone
- Writing speed in mobile phone storage unit

As we’ve seen, it’s not necessary to send the files to the hospital at the moment they are recorded. That’s why it doesn’t matter if the Internet connection the mobile phone has is fast or not. However, it’s recommended to have 3G or better (HSDPA, HSUPA, LTE...) in order to make the sending process as painless as possible, especially when sending long records.
CHAPTER 3. THE PROJECT

Communication between stethoscope and Bluetooth device
The stethoscope and the Bluetooth device are connected with a cable, using a 2.5mm stereo jack. The signal used in this communication is analog, so there will just be a little loss of sound quality between the ends of the cable. Because of that, there won’t be any problem to establish a real-time communication.

Communication between Bluetooth device and mobile phone
Bluetooth connection has a practical speed of up to 721 Kbps for 1.2 version and 2.1 Mbps for the 2.0 + EDR. However, this is the speed you could reach if the channel used (the air) to do the communication was perfect. That’s not the case, and in fact the air is a very noisy and unpredictable channel. Moreover, the 2.4 Ghz frequency Bluetooth uses is also used for many other applications because it doesn’t require any license to use it. Technologies like Wi-Fi use that frequency and tend to interfere with Bluetooth. That means that the practical speed is usually much lower than the theoretical one, so these speeds are not very useful to us. Because of that, we measured the average speed we reach sending files to our test mobile phone. In order to have the best results we did the test in an area with low 2.4Ghz interferences. We needed 6 minutes and 2 seconds to transfer 20,352 KB, which means an average speed of 450 kbps, less than one quarter of the maximum theoretical speed.

This measure has been done using k610i’s internal memory card. Further test with the phone’s internal memory gave almost the same results, so in this aspect there is no difference in choosing one or other storage device.

A/D Conversion in Bluetooth device
Ideally, in order to have a successful real-time communication the digitalized data speed should be slower than the minimum Bluetooth connection speed. This is impossible to accomplish, since instant Bluetooth speed fluctuates a lot because of the noisy channel we’re using (the air). Instead of that, we can still have a good communication by adding a buffer in the Bluetooth device and sticking with the following condition: Digitalized data speed should be slower than the average Bluetooth connection speed.

The choice of the buffer size has to be balanced. If it’s too big the device price will probably be too high. If it’s too small the transfer may experience loss of data. In fact, even if we use a really big buffer we’ll experience loss of data due to the random channel behavior. Fortunately, the kind of information we’re sending (audio) doesn’t have to be exact like if we were sending a document file. We can have some loss of data as long as it’s not too much. Thanks to that, we don’t need to implement any kind of error data check or retransmit protocol.

One the other side, we also have to choose the specifications of the A/D converter. A high number of bits will give good sound quality because the conversion will be more accurate. A high sample frequency will let us have higher bandwidth, according to the Nyquist theorem. However, both things
also mean higher transfer speed in order to send all that information in real-time. Because of that, we have to choose the simplest specifications that give us the minimum required sound quality. That would be:

**Sampling frequency**

The higher frequency we need to keep is 1900 Hz for lungs auscultation, according to our stethoscope’s specifications. According to the Nyquist theorem:

\[ f_s > 2B \]

\[ f_s > 3800 \text{ Hz} \]

Where:

- \( f_s \) Sampling frequency
- \( B \) Bandwidth

Considering filters are never perfect, we should consider a sampling frequency of at least 4000 Hz.

**Bit depth**

8 bit resolution is used to record medium and low quality sources. That includes AM radio, telephone audio, cassette and even FM radio. Stethoscope’s audio quality fits in this section, so even recording at bigger bit depth it’ll barely give us better sound quality than using 8 bits of resolution. We could choose less bit depth, but that would not be recommended unless we really need to lower our bandwidth and we can’t reduce frequency sampling. 16 bit resolution is considered good quality audio, and it’s the one used in CDs and most DVDs. 24 and 32 bit are used for extremely high quality sources, and it’s almost impossible to hear the difference with the 16 bit version unless you’re using expensive studio equipment. However, 24 and especially 32 bit are oriented towards the edition of sound, so it’s possible to make modifications without losing almost any information. After the modification, the audio file is usually converted to 16 or 24 bit for easier handling.

According to the conditions we said before, we must check if Bluetooth’s average speed is faster than the digitalized one. Our Bluetooth device has an 8 bit A/D converter that works at 44 KHz sampling frequency, so it’ll use:

\[ 44 \cdot 10^3 \text{ samples/second} \cdot 8 \text{ bit/sample} = 352 \cdot 10^3 \text{ bit/second} = 352 \text{ kbps} \]

Comparing that to the average Bluetooth speed...

\[ 352 \text{ kbps} < 450 \text{ kbps} \]

Right now we’ve seen that the communication speed between the Bluetooth device and the mobile phone is fast enough to have a real-time connection. However, we still have to consider other variables that may slow down the program and make it unfeasible.
CHAPTER 3. THE PROJECT

Computing processes in mobile phone

The processing power of the mobile phone processors is limited by the batteries because if they are too powerful, the device will need to be charged very often. Because of that, mobile phone processors are designed to be very efficient but, at the same time, they can’t have too much processing power. As a result, the applications shouldn’t use too many resources.

Our software is designed under J2ME platform. Java code is known to be simple but not very efficient, although the mobile phone edition, J2ME, is supposed to be better optimized. It’s important to have a program that doesn’t use a lot of processing power because it can slow down the rest of the processes and lead to loss of data when recording the stethoscope signal.

Writing speed in mobile phone storage unit

In a mobile phone, the writing speed in its storage unit is not really important. Considering that faster speed also means faster battery drain, mobile phones are not known for being fast in this matter. Anyways, it seems obvious that writing speed will be faster than any other wireless device those phones have.

We measured that speed in our K610i. We have different results because this phone has an internal memory unit and an external memory slot for proprietary M2 cards. The results are the following:

- Writing speed in internal memory: 1 min 0 sec for 13.312 KB => 222 KB/s
- Writing speed in M2 card: 1 min 13 sec for 33.700 KB => 462 KB/s (Sandisk 64 MB)

NOTE: M2 cards have different writing and reading speeds depending on their brand and model. Brands like Sandisk Ultra and Extreme edition cards are known for its high speeds in professional cameras and laptops, but in a mobile phone we wouldn’t notice any difference because the built-in card reader is way slower than the maximum speed of the card.

3.2.1.2 Software

The most common programming language for mobile phones is J2ME. It’s basically a simplified and optimized for low powered devices version of the most common Java (also known as Java Standard Edition). At first it was very simple and it was not possible to implement many things. But with the years it improved, adding extra profiles and APIs that gave complete control to the Bluetooth and GPRS functions. Thanks to that, now it’s completely feasible to create a mobile phone program that receives Bluetooth data and processes it, but it’s still not enough to send it through Internet the way we initially wanted, as explained in section 4.1.6.
3.2.2 Economical feasibility

The cost of the hardware and software used must not exceed our budget for this project. Fortunately, the advantages this project can offer are way beyond its price.

3.2.2.1 Hardware

The most expensive part of the project is the stethoscope. That stethoscope must have an audio output jack, which makes it a bit more expensive than classic stethoscopes. However, we also need one stethoscope to do regular auscultations in a hospital, so that doesn’t involve an extra spend of money. The Bluetooth device we use is a custom made version created in our department. Its main parts, the A/D and Bluetooth chips are very common and pretty cheap. The only extra cost will be the design and construction, which is all done in our department. Finally, we need a mobile phone with the features described previously. Most of the phones that are sold these days meet all those requirements, so we don’t really need an expensive one.

3.2.2.2 Software

The software we use to create the mobile phone software is Netbeans IDE. This program is open-source and free, so there’s no economical cost in the development of the software. The only thing we need is programming time.

To sum up, considering what we stated in the technical and economical sections, we can conclude that this project is completely feasible.
Chapter 4

Implementation and results

4.1 Implementation

In chapter 3 we saw all the parts we’ll be using in this project. However, we will be programming in only one part, the mobile phone, which will host a custom made application that can be divided into 5 parts:

- Interface
- Connection and configuration of the device
- Recording of the source
- File processing
- Setup and internal storage

The application developed in this thesis has been coded using Java language for mobile phones, known as J2ME. I have used Netbeans 6.8 as the platform to write and try the code because I already had some experience with it from a J2SE programming course I did in Universitat Politècnica de Catalunya, although I needed the help from some programming books\[8, 9, 10\] and the official documentation[14] in order to have the level to work with this language. However, I had some problems setting up this program for J2SE due to some incompatibilities with Windows 7 64 bit. On the other hand, even if Java is a multiplatform language, the truth is that there are still many variables that may change the behavior of the application. For instance, not all mobile phones that accept J2ME have the same libraries included and, as a result, there are some instructions that will only work in some devices or, in some cases, they won’t behave in the same way. That is why the term “multiplatform” that is associated with Java language should be clarified.

In order to test the code while programming, Netbeans includes some generic mobile phone emulators that show how the program will work. Even
though those emulators are fast and pretty good, in some cases they don’t behave in the same way as the mobile phone used for this thesis. Moreover, they also emulate the mobile phone’s Bluetooth, with means that they are not really using the computer’s one. Due to the fact that we must connect to a real Bluetooth device, this kind of emulation is not useful for us, so we had to look for alternatives. There are projects like BlueCove\(^1\) that solve this problem, but we still won’t be sure that the program works properly in our mobile phone until we upload and test it there. This process would be really painful but, fortunately, Sony Ericsson (the brand of the mobile phone used in this thesis) offers an impressive tool that solves all the inconveniences said before. This is a suite for developers called Sony Ericsson SDK, which includes tools and emulators for specific Sony Ericsson phones. There is also a special emulator called Debug On-Device, which allows the developer to run and debug the program directly in the phone. The advantages of this emulation method are obvious: you see the output directly in the mobile phone, knowing that it will behave in exactly the same way once you upload the definitive file to the phone. The problems with Bluetooth emulation explained before are solved as well, so there is no need to use programs like BlueCove anymore. In contrast, this emulator uses the phone’s Central Processing Unit (CPU) for all the work, so the debugging process can be very slow and annoying, but after all it is best choice we can do.

The following subsections are the 5 parts in which this program is divided.

### 4.1.1 Interface

This is the part the user will see when using the program. It’ll be responsible of showing dialogs and forms in the mobile phone’s screen. It’ll also assign functions to the buttons and will include error checking procedures to ensure that everything the user introduces makes sense. In other words, the interface is the link between the user and the internal phone’s processes and, therefore, it should be intuitive and flawless.

Even if the interface does not do any interaction with the Bluetooth module or the stethoscope, it is the one that gathers the other parts of the application, which are independent to each other. This is why the interface flowchart found in figure 4.1 (page 32) is useful to understand how the program works. In this flowchart, blue rectangles are screens, red circles are start or end points and yellow rhombus are decisions the phone takes according to the shown conditions. The text that comes with the arrows are the buttons that need to be pressed to go to the next screen, and if there is no text it means it changes the screen after the required operations are done, without pressing any button. Since this is a flow chart of the interface, it shows what the user can see and do but not what it does in the background. In the following list there’s a description of every phone screen and actions, as well as the name of the main methods that are involved in them.

---

\(^1\)More information can be found in their official website: http://bluecove.org
Figure 4.1: Application’s flowchart
Main Screens

Main menu This is the first screen that is shown when you start the program and it is considered the starting point. From here you can start connecting to the Bluetooth device, go to the Setup menu or exit the program. Figure 4.2.

Connected This screen is shown when the connection with the Bluetooth device is successful. It also shows some detailed information for debugging purposes, like the number of tries needed to configure the devices. This number should always be 1. Figure 4.3.

Can’t connect This is the screen that is shown if the connection with the BT device cannot be completed successfully. I have only experienced three reasons that will lead to this screen: phone’s BT not enabled, BT device disconnected and BT device out of range. This screen should also appear if it was not possible to configure the device once it’s connected, even though this has never happened yet. Figure 4.4.
CHAPTER 4. IMPLEMENTATION AND RESULTS

**Recording** This screen tells the user that the mobile phone is recording the signal that comes from the stethoscope. While this screen is visible, the phone is running a *read* and a *write* process in the background, as described in Section 4.1.3. This process won’t stop until the user presses the *stop* button. Figure 4.5.

**Process data** This screen is shown after pressing the *stop* button. While this screen is visible the phone will run two processes: Finishing the writing to the internal memory and, after that, processing the data. A description of this processing task can be found in Section 4.1.4. Once the tasks are finished this screen will be replaced for the *Record finished* screen. Figure 4.6.

**Set up screens**

**Setup menu** This is the main menu for changing the configuration of the BT device and the program itself. For the first one, it shows its Bluetooth
friendly name, URL, and lets the user choose if he wants to require authentication between the devices, encrypt the transmission or set the device as master. It also gives the option to perform a device search in case the user wants to use a different BT device. For the second one, it is possible to set a new base name for new recorded files, it gives the following file number to be used and it also lets the user reset the file number to 0, which also means deleting all files that already had the same base name. Finally, it is possible to save the new changes, load old recorded changes, discard them or just leave using the new settings but without saving them. Figure 4.7.

**Confirmation** This screen is not a form as the other ones, but an alert message that just warns the user that by pushing the yes button all the files that include the base name set in that menu will be deleted. Figure ??.

**Search menu** This menu tells the user that by proceeding it will start performing a new device search. Figure 4.9.
Figure 4.8: Confirmation screen

Figure 4.9: Search menu


CHAPTER 4. IMPLEMENTATION AND RESULTS

Figure 4.10: Searching devices screen

Figure 4.11: Found devices screen

**Searching devices** This screen will be shown after choosing the *Search* action. It means that the phone is performing a device search and is also trying to get the friendly name of all found devices. Once the procedure is done it will automatically be replaced by the *List of found devices* screen. Figure 4.10.

**List of found devices** This is just a list of all BT devices that were found in last device search, shown with their friendly name. The possible actions for this screen will be different whether there is at least one device or not. If there are no devices found it will only be possible to choose the discard action, otherwise it will be possible to select one of the shown devices. Figure 4.11.

**Searching services** This screen will be shown after choosing the *Select device* action. It means that the phone is performing a service search on the chosen device. Once the procedure is done it will automatically be replaced by the *List of found services* screen. Figure 4.12.
List of found services. This is just a list of all BT services that were found in last service search. The possible actions for this screen will be different whether there is at least one compatible service\(^2\) or not. If there are no compatible services found it will only be possible to choose the discard action, otherwise it will be possible to select one of the shown services. Figure 4.13.

Main actions

Connect. The phone initiates several actions in order to establish a connection with the Bluetooth device. These actions are described in Section 4.1.2. If it is possible to create the connection and configure the device, screen Connected will be shown, otherwise it will show the Can’t connect screen.

\(^2\)Compatible services must use SPP. Devices that accept this profile might not be compatible with this application. However, devices that don’t accept SPP will never be compatible.
CHAPTER 4. IMPLEMENTATION AND RESULTS

**Record** The phone starts recording the auscultated sound, using the instructions described in Section 4.1.3, and it shows the **Recording** screen as well.

**Stop** First, it shows the **Process data** screen. Then, it stops the BT device following the steps described at the end of Section 4.1.3 and starts the file processing stage found in Section 4.1.4. Once the processing it’s finished it shows the **Record finished** screen.

**Menu** Loads the **Main menu** screen. If this action is done during the **Record finished** screen, it also closes the BT connection.

**Exit** Terminates the program. If this action is done during the **Record finished** screen, it also shuts down the BT device.

**Back** Goes back to a previous menu.

**Set up actions**

* **Setup** Loads the **Setup menu** screen.

* **Search BT devices** Shows the **Search menu**.

* **Search** Shows the **Searching devices** screen and then performs a device discovery process following the instructions described in Section 4.1.5. Once it’s done it will load the **List of found devices** screen.

* **Erase files** Loads the **Confirmation** screen.

* **Yes** Erases all files and brings back to the **Setup menu**. It also updates the file count in that menu.

* **No** Brings back to the **Setup menu** without doing any changes.

* **Save** Saves all changes done in the **Setup menu** to the phone’s internal storage and brings back to **Main menu** screen. This action is not shown in figure 4.1, but can be found in can be found in the **Setup menu**.

* **Load** Loads the configuration that was saved previously using the **Save** action. It also updates all the objects in the **Setup menu** according to the new configuration. This action is not shown in figure 4.1, but can be found in can be found in the **Setup menu**.

* **Discard** Brings back to the **Setup menu** without doing any changes. There is also a **Discard** action in that menu that brings back to the **Main menu** screen without applying any changes, although it’s not shown in figure 4.1.
4.1.2 Connection and configuration of the device

It is not possible to send information between the devices unless we previously create a communication between them. The program will create it using the function `bt_Unit.Connect()` and the address shown in the Setup menu. Once it’s done, it will remotely configure the Bluetooth device in order to set the proper configuration, using `bt_Unit.SetupAndStartBT()` function. It is very important that this configuration is done successfully because if it wasn’t like this all the recorded files would be corrupted. To avoid this situation, this last function has also an error checker that will reconfigure the device if it detects it was not done successfully.

In order to configure the BT device, the `bt_Unit.SetupAndStartBT()` will send some commands to it using the customized protocol shown in Table 4.1. The whole command must be 14 byte long, so bytes must be filled with zeros when there’s nothing to write in them. After that, the device will send an answer indicating which command was received and if there was any problem with it, using the protocol that can be found in Table 4.2. To continue sending commands the phone will first have to wait for a while so the device is ready to receive again. In this program we’ve set a waiting time of 250 ms, which is way more than it’s needed, but it assures that if there’s a configuration problem the reason won’t be the speed in sending commands.

The `bt_Unit.SetupAndStartBT()` function will check the BT answers once it has sent all the required commands using `bt_Unit.CheckConfig()`. If there’s a configuration problem, `bt_Unit.CheckConfig()` will return a false and `bt_Unit.SetupAndStartBT()` will try to configure the device again. This process will be done up to 5 times and if this problem was still not solved `bt_Unit.SetupAndStartBT()` will return false so that the phone knows it was not possible to do a correct configuration.

It is important to note that the way `bt_Unit.CheckConfig()` does all the checks is not really logical due to an important K610i bug. During the tests I noticed that the program was sometimes hanging without any apparent reason. I discovered that when doing a read call in an empty Bluetooth buffer stream the phone stays waiting for received data forever instead of returning a -1\(^3\). This bug involves a lot of trouble in all readings for the fact that you must do the exact number of reads if you don’t want to hang the program. However, this option is not useful because if you loose some data you can also reach the end of the buffer before doing the reads that were supposed to do, meaning that the phone would also be hung forever. Nevertheless, there’s a really easy solution to that problem: If the Bluetooth device starts sending data while the phone reads, the read function will never be stuck waiting for new data. As a result, `bt_Unit.SetupAndStartBT()` will send a Start4.1 command to the BT device before starting to read the answers to the previous configuration attempts. Finally, once `bt_Unit.CheckConfig()` returns something, the phone

---

\(^3\)Official documentation for the read function can be found in [http://download.oracle.com/javame/config/clk/docs-refimpl/midp2.0/jsr118/java/io/InputStream.html#read(byte[])](http://download.oracle.com/javame/config/clk/docs-refimpl/midp2.0/jsr118/java/io/InputStream.html#read(byte[]))
<table>
<thead>
<tr>
<th>Byte number</th>
<th>Start</th>
<th>Stop</th>
<th>Power Off</th>
<th>Common Settings</th>
<th>Individual Channel Settings 1. Channel type</th>
<th>Individual Channel settings 3. Channel Gain and Power settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>30</td>
<td>31</td>
<td>33</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>Nr of Channels</td>
<td>Channel</td>
<td>Channel</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>Grundton fCommon prescaler</td>
<td>Type*</td>
<td>DC gain</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>Grundton fCommon b8-15</td>
<td>ADC Input</td>
<td>AC gain</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td>Grundton fCommon b0-7</td>
<td>DC pot Port</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>Least Common Denominator</td>
<td>DC pot CS Pin</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AC pot Port</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AC pot CS Pin</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LED power index</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Possible Types:
1 = PPG_MUX
2 = PPG
3 = Local ADC, local amp
4 = Local ADC, external amp
5 = External ADC, external amp
Table 4.2: Commands and data from device to phone

<table>
<thead>
<tr>
<th>Byte number</th>
<th>Send error</th>
<th>Command acknowledge</th>
<th>Link connected</th>
<th>Resend</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MSBit 6-0</td>
<td>MSBit 6-0</td>
<td>MSBit 6-0</td>
<td>MSBit 6-0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

*Errorflag*
0 = OK
1 = Failed

Table 4.2: Commands and data from device to phone

<table>
<thead>
<tr>
<th>Byte number</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1st packet</td>
</tr>
<tr>
<td></td>
<td>MSBit bit 6-0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
CHAPTER 4. IMPLEMENTATION AND RESULTS

will send a *Stop* command and it will close all the streams so that the data received during this *Start* command gets lost.

Now that we know how this device must be configured, here are the steps that the program will follow to achieve this function:

1. The phone tries to connect to the BT device using `bt_Unit.Connect()`. If it can't connect, it goes to point 13.
2. Once it’s connected, an inputStream and an outputStream are created.
3. It sends the *Common Settings* and waits 250 ms.
4. Then it sends the *Individual Channel Settings 1* and waits 250 ms.
5. Then it sends the *Individual Channel Settings 3* and waits 250 ms.
6. Finally, it sends a *Start* command.
7. The phone reads the inputStream.
8. It discards all the packets that are not commands.
9. It checks if the commands were accepted by the BT device.
10. If they are not OK, it goes back to point 3 up to 5 times. If it’s the 6th time, it goes to point 13.
11. It sends a *Stop* command.
12. It closes the streams created in point 2.
13. The interface shows the *Connected* screen if everything was successful. Otherwise it disconnects and shows the *Not Connected* screen.

4.1.3 Recording of the source

Once the user pushes the Record button, the phone will read the information sent by the stethoscope and it will record it to a specified file. To do so, it will first call `phoneWriter.InitialiseFile()`, which will open and create, if needed, the future two files where the recording will be stored. The name of these files will be chosen according to what is explained in Section 4.1.5. We will use `fileName0_raw.dat` and `fileName0_processed.dat` as file name examples.

Next step will be opening an inputStream and an outputStream and sending to the BT device a *Start* command, in the same way that it was done in the configuration of the device. After that, two background processes will be run during all the recording: `bt_Unit.start()` and `phoneWriter.start()`. The first one will read the data from the BT device and will put it in a common buffer, while the second will read from the buffer and write it in `fileName0_raw.dat`. As it’s
been told, this program uses a common buffer in which two processes work with it, so in order to avoid data corruption it is necessary to have some concurrent programming skills. In this case, I decided to use an object called *Synchronizer* that will own the buffer and will make sure that all the read and write operations are done in a synchronized way. As a result, *bt_Unit* process will call synchronizer.Read(), while *phoneWriter* process will call synchronizer.Write() as fast as they can, and *Synchronizer* will do the timing. At the end, the whole process will actually consist in many read and write operations, given that each operation will handle few data packets and a typical recording will consist in many packets.

Once the user pushes the Stop button, a Stop command is sent to the BT device so it stops sending information. It closes the input and output streams that were opened before and waits for the rest of the background processes to terminate. The *BT_Unit* background process stops reading once the input stream gets closed, while the *phoneWriter* background process stops writing once it doesn’t have anything else in the buffer. Then it proceeds running a thread that processes the written information using *fileProcessor.start(*), as it is explained in the next subsection.

### 4.1.4 File processing

Most communications between devices send two types of data: user data and command data. The first one is the data we want to keep, while the second one is data used to ensure the first one is received properly, which includes extra bits for error checking, synchronizing or just bits used to distinguish these two types of data. Due to the fact that we receive all data together, the next step after recording the source consists in discarding all command data, as well as user data that has not been received successfully. This is what will be done in this part of the project.

The file stored in the phone’s memory with _raw_ in its name (i.e. *fileName0_raw.dat*) is an unprocessed file, meaning that it has exactly the information received by the BT device. This information is received in packets of two bytes (data packets) or four bytes (command packets). In all of them the Most Significant Bit (MSB) is used for synchronization and to indicate what kind of information has the received packet. The rest of the bits will be data bits or command bits, according to the values of the MSB. In order to achieve the goal of this part, the phone will start an independent thread using *fileProcessor.start(*). This thread will first have to discard all the command packets and keep the ones with data. After that, it will analyze the data packets, which will consist in 2 bytes (16 bit) with 14 bits of useful data and 2 useless bits for synchronization, and it will copy these useful bits into a new file that will have _processed_ in its name (i.e. *fileName0_processed.dat*). Considering there will be only 14 useful bits per sample, the information will be written in a 2 byte packet, which will also have 2 zero padding bits located in the Most Significant Byte (MSByte). The Figure 4.14 shows how this new data packets are built.

It is important to note that the whole processing task requires some compu-
tational power, especially for long files, so it can take some time to be finished. This working time will be proportional to the length of the recorded sound and, once it’s done, the main thread will also close the file connection opened in the.

Now that we know how the file processing thread works, here are the steps the program will follow to achieve its function:

1. The phone reads a byte and stores it in the 2 byte array `buffer[0]`. If there was no more information to read, it concludes this process.

2. If the MSB of `buffer[0]` is 1 it means we are at the start of a packet. If it is 0, it discards it going back to point 1.

3. The phone reads another byte and stores it in `buffer[1]`. If there was no more information to read, it concludes this process.

4. If the MSB of `buffer[1]` is a 1 it means we are reading a command packet (4 bytes), so we have to discard it. This is done by reading again the remaining 2 bytes and going back to point 1. If `buffer[1]` is a 0 then we have read an entire data packet (2 bytes) and we can continue.

5. Now we have in `buffer[0]` and `buffer[1]` an entire data packet. The phone proceeds to modify these packets to make them look like in Figure 4.14.

   (a) It takes out the 1 of `buffer[0]`’s MSB by doing `buffer[0] & 0x7F` and copying the result to the `short` type variable `buffer1`.

   (b) It moves the values in `buffer1` 7 slots to the left by doing `buffer1 << 7`.

   (c) It does an OR between `buffer1` and `buffer[1]` to join them by doing `buffer1 | buffer[1]` and copying the result to the `short` type variable `output`.

   Output now looks OK but in order to write it in a file we need a 2 byte array instead of a 2 byte long variable.

   (d) It creates the MSByte by moving 8 slots to the right the `output` variable and copying the result to `buffer[0]`. This is done using `(output & 0xFF00) >> 8`. 

Figure 4.14: File processing procedure
(e) It creates the Least Significant Byte (LSByte) by copying the output variable to buffer[1] and discarding the MSByte. This is done using \( \text{output} \& 0x00FF \).

6. Finally the phone writes the 2 byte buffer array into the internal storage and goes back to point 1.

### 4.1.5 Setup and internal storage

This part is not directly linked with the main function of the program but has an important role for the application in terms of functionality. It lets the user choose which Bluetooth device to use (it will perform a device and service search in order to find devices that support the required Serial Port Profile), as well as other Bluetooth settings like if it is necessary for the devices to authenticate, if encryption will be required or if the phone will behave as the master in the communication. There are other things that can be changed, such as the folder and the name of future recorded files, features very useful in case of auscultating different patients. Finally, it also lets the user delete all the files recorded with a chosen basename, as well as save current settings in the internal storage, load old ones, discard changes or simply apply new settings without saving them.

#### Device search

It is possible to perform a BT device search from the Setup menu. To do so, the user has to put the cursor in the name of the device or in its URL and press the Search button. A new menu with some instructions will be shown, and it will be necessary to press the Search button again to start this discovery process, running `BT_Unit.searchDevices()` function (based on the following references: [12, 13]). This function might look simple because it only asks the phone to start searching devices, but it also requires the use of many other methods to make it work that are not described in this thesis because they're out of the scope of this work. As the devices are being found, the program stores their address in a list, tries to get their friendly names (a customizable name used instead of the BT address), and finally shows this list once the process is finished. Then the user will be able to choose one device or go back to the Setup menu in case there were no devices found. If the user selects a device, then a service search will be performed in order to find if that device has SPP service enabled. A new list of found SPP services will be shown in the screen, asking the user to choose one of them. After that, the selected device will be set as the new BT device to connect to, and its URL and friendly name will be shown in the Setup menu. It is important to note that the use of SPP for the communication between the devices is a must for this program, so if the selected device is not compatible with SPP the phone will ask the user to go back to the Setup menu.
Other BT options

This program lets the user configure 3 different BT parameters: Authentication, encryption and master/slave. Currently the BT device in which the project is based on requires to authenticate when establishing a connection and to set the phone as slave, but that could be different in the future. However, the encryption parameter is totally optional, although considering that this program will manage medical data, it is strictly recommended to activate this encryption unless the user experiences performance problems with it.

File name of future recordings

Each auscultation is stored in the internal memory of the phone using a file name that can be chosen in the Setup menu. In fact, the user chooses a Base name and the phone appends in it the number of the recording, so it is possible to perform multiple auscultations using the same base name and different record index. This is very useful in case that this program is used to perform auscultations on multiple patients: The Base name can be the patient’s name, while the index will be used to differentiate recordings on the same patient. The Setup menu will show the current base name, as well as the following recording number that is going to be used for the next recording. It is not really necessary to check which is going to be the next recording number because after each recording the program will show the full name where it has stored it. This full name will follow this structure: Base name, followed by the current File number, the character “_”, the word processed and, finally, the extension .dat. There will also be another file with the same name but with raw instead of processed in it. This raw file can only be useful for debugging in case of errors of file corruption, since it includes the information received from the BT device before processing it. In this thesis we will use fileName0_raw.dat and fileName0_processed.dat as example file names. More information about the writing process can be found in Section 4.1.3.

In the Setup menu it is also possible to delete all the files that have a specific Base name. To do so, the user has to put the cursor in the number of the next recording and press the Reset button that will appear. An alert menu will appear, informing the user that all files will be deleted and giving the option to choose yes (delete) or no (discard). If the user chooses to continue, the phone deletes all files and sets the File number to 0.

4.1.6 About e-mailing the recorded file

Initially, the main goal of this project was create a program for a mobile phone that could receive an auscultated sound from a stethoscope, record it and resent it through e-mail to a computer located, for example, in a hospital. However, after some research I discovered that this resending part is not possible to do using J2ME in the device we’re using for this project. There are some open source API that allows a J2ME program to send e-mails[3], but it is not possible
to include attachments in them. The use of a FTP server has also been discarded as well, due to the same J2ME limitations, so the only option to send a file using J2ME is to send it to an Internet server using Hyper Text Transfer Protocol (http) or Hyper Text Transfer Protocol Secure (https). This alternative is way more useful than sending recorded files through e-mail, but it also requires a dedicated server in the hospital and way more programming work in it and in the mobile phone’s application. There might also be some privacy issues, so the communication must be done using a secure protocol.

Because of the reasons mentioned before, this thesis does not consider the option to send the recorded file to a hospital. However, there is a workaround for this problem: The K610i offers its own mailing system in its Operating System (OS). This mailing system accepts attached files, so it is possible to include the recorded file in it. That means that even though it’s not possible to send a recorded sound through Internet using the program developed in this thesis, it is still really easy to send that file using the included mobile phone’s mailing system. To access this mailing system, the user will have to exit or minimize the program, go to the phone’s menu, choose Messaging and then Email. From here it will be necessary to configure the mailing system (just for the first time), then choose Write new, fill all required fields, push Continue and, finally, push Send. This process has been successfully tested, as it’s shown in Figure 4.15 for the composing of the e-mail and Figure 4.16 when it’s received. In the future this process could be improved using the Internet server described before, but right now this workaround does the job the way is should in a very easy way.
4.2 Results

In this section we will see a real heart recording and we will try to validate it in order to know if the resulting signal is accurate enough for a future use. This is the most important part of this thesis, since all the work done until now is useless if we’re not able to perform heart or lungs auscultations. Considering that heart auscultations are way easier to hear and see in a graphical way than the lung ones, in this section we will only see the first one.

4.2.1 Heart auscultation

As it’s been explained in Section 4.1, the mobile phone will generate a raw file and a processed file for each recording. The first bytes of a raw file will look like shown in Figure 4.17. This values belong to the example found in Figure 4.19, which is the auscultation of a heart that has been resting for a long time. The file starts with the hex value 0x82, which according to Table 4.2 it means that a command has been received. After that, we’ve got a 0x81, which means that there was an error with the received command. This is a bug that only affects to some commands, as explained in Section 4.3, but the command is actually received successfully. Finally, the following byte, 0x01, is the Start command that was received. This Start command forces the device to start sending information, so the rest of the bytes belong to the signal itself, grouped as 2 bytes per sample.

In Figure 4.18 we have the same recording as before, but in that case it is after the processing procedure described in Section 4.1.4. All the bytes found in this file belong to the recorded sound, grouped as 2 bytes per sample as in the raw file, but without synchronization bits.

Right now we just saw the received files in a byte level. However, it’s impossible guess what we really have unless we see this bytes in a graph.

After plotting the processed file, we’ve got the signal found in Figure 4.19.
As we can see, it is perfectly possible to distinguish systolic and diastolic beats, although it also looks like the samples between them have a bit more noise than expected. This auscultation has been made using a sampling speed of 960 Hz, perfect for heart auscultations (up to 420 Hz, so at least 840 Hz of sampling speed), but not enough for lung ones (up to 1900 Hz, so at least 3800 Hz). Considering that, we can calculate the heart’s Beats Per Minute (bpm) doing the following:

\[
\text{Samples between beats: } 1119 - 261 = 858 \text{ samples/beat}
\]

\[
\frac{960 \text{ samples/second}}{858 \text{ samples/beat}} \cdot 60 \text{ seconds/minute} = 67 \text{ bpm}
\]

The resulting value, 67 bpm, is a pretty reasonable value for this case.

4.2.2 Validation of the signal

In last subsection we saw a real heart signal, but even if it looked like OK to our eyes, it was not possible to know if they were distorted in any way. Because
of that, the only way to validate a signal is to put a previously known one in the BT device’s input and check if we have the same signal in the recorded file. To do so, we created several sinusoid signals with previously known frequencies that are inside the spectrum of a heart and signal. The chosen frequencies are the following: 20 Hz (the lowest auscultable heart frequency), 100 Hz (a medium frequency signal) and 480 Hz (the highest auscultable heart frequency). Then we plugged the sound card output of a laptop to the BT device and reproduced each sound. After recording it, we compared the original sound (sampled at the same frequency as the BT device) with the recorded one, using 2 different sampling speeds: 960 Hz and 4800 Hz. The results can be found between Figure 4.20 and Figure 4.25, where the red line is the original signal and the blue line is the recorded one. As we can see, both signals have almost the same values and they can be considered almost the same. Figure 4.22 might seem wrong because it almost doesn’t have a sinusoid shape, but the result is perfectly normal: Using a sampling frequency of 960 Hz, the maximum frequency we can record is the half, 480 Hz, and this signal is pretty close to that limit. Right now it doesn’t look like a normal sinusoid, but with some post processing it is completely possible to recover the signal and make it look like the others. An even more extreme example is the one shown in Figure 4.26, where we did the same thing with a 480 Hz signal and we can see how we have exactly 2 samples per cycle and the recorded signal also looks like it is doing a good job with it. It is also important to note that the laptop used to reproduce the signals is not perfect, so the real difference between both signals is even less than what we see in the figures.

To sum up, even if in the real heart auscultation it looked like we have more noise than we could expect, the validation of the signal showed us that the process is, in fact, pretty good. However, further tests with way more complex signals should be done in order to be sure about that. On the other hand, the slightly high levels of noise detected in the real heart auscultation will probably come from the stethoscope itself.
Figure 4.20: 20 Hz Sinusoid, sampled at 960 Hz

Figure 4.21: 100 Hz Sinusoid, sampled at 960 Hz

Figure 4.22: 420 Hz Sinusoid, sampled at 960 Hz
CHAPTER 4. IMPLEMENTATION AND RESULTS

Figure 4.23: 20 Hz Sinusoid, sampled at 4800 Hz

Figure 4.24: 100 Hz Sinusoid, sampled at 4800 Hz

Figure 4.25: 420 Hz Sinusoid, sampled at 4800 Hz
4.3 Found bugs

It is impossible to create a program that does not have any errors, flaws, failures or security holes, and the program in this thesis is one example. The fact that we’re using a high level programming language also means that not only the programmer itself can create errors, but also the libraries used in it can have bugs or incompatibilities with the device that we’re using. The only thing we can do is to track the most important bugs and try to fix them. However, there might be some bugs that are impossible to fix, forcing the programmer to find a workaround so the bug does not affect the normal use of the program. In general terms, the bugs will come from the BT device or the mobile phone.

Here’s a list of some of the found bugs:

**BT Device**

**Command acknowledge** The BT device answers with a command error when it receives a *Start* or *Stop* command. However, the device works as if it was received successfully. This bug is still not fixed, although it’s not important at all once you know that it’s a bug and not that the command was not received.

**Battery charge** The BT device had trouble charging battery. It was completely impossible to charge for more than 10-20 seconds since it stopped charging after that. This bug has already been fixed.

**Sampling frequency** It was not possible to fix a value for that. The default value was too slow, about 20 Hz, so it was not possible to record anything. It was pretty hard to find that because we had to check also if it was a problem with the program or the mobile phone that was loosing data. This bug has already been fixed.
**DC gain** It is not possible to fix a value for that. The default value seems OK, so it’s perfectly possible to do some recordings without changing that, but it would be interesting to fix that so it could be possible to find the best value for the recording.

**Mobile phone**

**Read() function** If the buffer is empty and the program calls the `read()` function, it gets stuck there waiting for information to read instead of returning a -1 like it is supposed to be. Moreover, considering that when the BT device is stopped only sends acknowledges, the program will be hanged forever. There is no way to fix this bug because it’s not possible to access this functions, so it was necessary to find a workaround in order to avoid this fatal bug. To do so, the phone will never read the buffer until the device has received a Start command and it’s sending information. After that it will read as much as it’s necessary and will stop de device. This procedure is especially important when configuring the device, as described in Section 4.1.2.

**LocalDevice.getLocalDevice()** It looks like in some conditions if the program does not close everything when it finishes, it is not possible to get the Local Device anymore. When trying this function the phone hangs forever without giving any kind of exception. The biggest problem is that when that happens it is not possible to call this function even if you restart the program. The only solution to this bug is to shut down the phone. It can also be very hard to find because this function works the first time, but not the others, and usually it’s called at the start of the program, so instead of showing the program what you really see is the first “Please wait” screen from the phone, stuck forever. Since this bug it’s impossible to fix, the only way to avoid it is to make sure that every connection is closed properly.
Chapter 5

Conclusions and future work

5.1 Conclusions

This is the moment to think about what has been accomplished in this project. As described in the objectives of this thesis, the main goal is to be able to perform a heart or lung auscultation in a patient, record that sound, and send it to a hospital somewhere else in the world. As we’ve seen in the Results section, it is possible to record and visualize this signal successfully, so this part is done without problems. Moreover, the validation of the signal showed pretty impressive results, although it may be necessary to do further tests with complex signals to see if this performance is always that good. On the other side, it was not possible to send this file using the same program because of the limitations of J2ME, but the workaround explained in Section 4.1.6 is enough to fit our requirements. As a result, we can say that the main objective of this program is fully accomplished.

Even if the recording part is the most important of the project, there are other parts that improve the user’s experience. For instance, the interface used in this program is intuitive, shows the user all the available options, and informs him about what the phone is doing in every moment. The inclusion of a set up menu is also very useful, not only for searching future new devices, but also to change the name of future recorded files in case it’s needed.

On the other hand, this work has also been useful to see that even if J2ME is supposed to be a multiplatform language, like the standard Java, it is in fact not completely true. It is pretty easy to find different behaviors running the same program in multiple devices, so that is why this program might not work in the same way in a phone different than the Sony Ericsson K610i. Furthermore, the existence of bugs can make the programming experience way harder than it looks like, as well as it requires a lot of time in testing the program to make sure it works flawless.
5.2 Future work

The program developed in this thesis can be improved in many ways, although most of them will only make the experience easier for the user, other than really improving the recorded signal. The BT device also needs some work, as described in the following list.

Phone software

- **Sign the software**
  In order to protect the phone’s integrity, J2ME programs have some restrictions in the actions they can do. If the software is not signed with a valid certificate, many of the actions that can be a security risk are not run without a previous confirmation of the phone user. That involves all accesses to the internal memory, as well as all wireless connections.

- **Add the option to send the recorded file to a server through https.**
  This is a really big improvement to this software and should be the first one to be done. Thanks to that, the whole process wouldn’t depend on the phone’s internal mailing system anymore, as well as it would be possible to automatically process the file once it’s received in the server.
  Right now, the processed file generated by the phone cannot be directly listened or watched in a graphical way. The easiest way to have a file that can be listened is to convert it into a *wav* file (file extension .wav). I tried to do that, but I did not get any sound that could be heard, so I guess it needs more work than it looks like. Theoretically, it is necessary to first append in the start of the file a *wave* header, including all the information like sampling frequency or bits per sample. It might also be necessary to adapt the bytes so they are in the right position[11]. Then the sound player will be able to detect it as a sound, but it’s still not enough because the values of the signal have a strong DC value that must be filtered. Finally, the entire signal must be normalized because in fact we just have a 14 bit per sample signal, instead of the 16 bit one that we must have if we want to listen to the sound.
  On the other hand, it will also be interesting to automate the process of creating an image of the signal. All these things will only be possible if we get to send the files to a server, that’s why it’s so important.

- **Improve detection of errors**
  Right now the program does not have many error checks. It’s not really necessary because almost everything works flawless all the time, but it is also true that there is almost no real exception handling in the program.
  Right now all important exceptions are shown in the console, but not in the phone’s screen.

- **Attach information fields like Description in each record**
  Thanks to that it would be possible to guess what’s the difference between
every record. Right now it’s possible to change the base name of the file, which is especially useful if we’re auscultating different patients, but there’s no way to differentiate auscultations with the same base name.

- Ability to choose the sampling frequency
  This new feature is really easy to implement and will be useful if the program is used for other kind of recordings that need a specific sampling frequency. Right now it might also be useful when recording long sounds because the BT device disconnects itself when its buffer gets full, and that happens when using a 4800 Hz sampling frequency, necessary for lungs auscultation. For heart auscultations it’s not necessary to use such a high sampling frequency; 960 Hz is more than enough and there is no problem with recording long sounds at this sampling speed.

- Calibration of the amplification
  This is the only thing we can do by software to really improve the recorded sound. According to the patient and the place where you put the stethoscope, the signal that is received can have extremely different maximum and minimum amplitudes. Ideally we should record a sound that would use all the values we have available for recording (14 bits), but this is not possible due to this changing amplitudes explained before. This calibration would improve that by analyzing one or two seconds of a sound being recorded, calculating the value of the BT device amplifier that would fit best for this signal and setting the device with this new value.

**Bluetooth device**

- Include overflow control
  This device is using a buffer that hosts all data that has not been sent yet. For some kind of reason, due to the phone or the device itself, this device is sending the information in bursts, not in a continuous way. Because of that, the average sending speed gets decreased considerably, at a point where it can be slower than the digitalizing speed, meaning that the buffer can get filled during a long recording. Right now, this device is programmed so that when the buffer gets filled it disconnects the unit, and therefore, the recording becomes useless. In order to prevent that, this device could discard some bits and send to the phone the number of bits discarded, instead of just disconnecting. This would be a good solution, although in fact what we should really do is to find why it’s not possible to have a long recording with the 4800 Hz sampling frequency we were using for lungs auscultation.

- Implementation of error correction
  This feature is always welcome in this kind of communication. However, it involves a big change in the protocol that might not be worth the effort. With all tests done so far it looks like it’s not common to find any communication errors.
• Add battery status
  Right now there’s no way to know the status of the battery. It would be very useful to use one of the device’s lights to indicate if the battery is low in order to prevent the impossibility to record because of the battery.

Hospital
• Create an application for the server to receive recorded files through https
  This has been explained in the *Phone software* part of this section. To do so we have to work in the server located in the hospital and in the phone’s software. It will also be very useful to process received files so we can have a recorded sound that can be listened, as well as an image that shows this signal in a graphical way.
Appendix A

J2ME Code

A.1 Important functions that appear in this thesis

BT_Unit.java

```java
public class BT_Unit {
    public void run() {
        while (!synchronizer.Read(inputStream));
    }

    public void Connect(String direccio) {
        try {
            connexio = (StreamConnection) Connector.open(direccio);
        } catch (Exception ex) {
            System.out.println(ex);
        }
    }

    public boolean SetupAndStartBT() {
        // User configurable
        int TRYLIMIT = 5; // Number of tries to config the device. It shouldn’t be important since I’ve always seen that it checks everything at the 1st try..
    }
}
```
// Not user configurable from here
boolean configOk = false;
openStreams();

do {
    // Stop (It's not necessary, but just to be sure it's stopped)
    writeCommand(new byte[][]{({byte) 02}, 14);

    // Set of Common Settings
    // Bytes 4 and 5 are for sampling frequency ("Grundton fCommon"), we'll use the 5th.
    // Calculation: sampling freq = 14.7456 MHz / (1024 * (fCommonLSB+1)). 1900Hz is the maximum
    // frequency of the stethoscope => 3800 Hz
    // sampling speed.
    // For heart minimum: 14 (it’s 16 but 14 to be sure)
    // For lungs minimum: 2 (3 is almost good, but not enough)
    writeCommand(new byte[][]{({byte) 30, 0x01, 0x00, (byte) 0, (byte) 14, 0x00}, 14);

    // Individual Channel Settings 1: Channel type
    writeCommand(new byte[][]{({byte) 31, 0x01, 0x02}, 14); // The 3rd byte should be 2 if we use the PPG

    // Individual Channel Settings 3: Channel gain and power settings. The 3rd and 4th bytes
    // should change the gain, but right now this feature does not work in the BT device and it
    // doesn't matter its value.
    writeCommand(new byte[][]{({byte) 33, 0x01, (byte) 0xFF, (byte) 0xFF}, 14);

    // Start
    StartBT();

    // Check if commands were ok
    configOk = CheckConfig(true, true, true);
    tries++;
} while ((configOk == false) && (tries <= TRYLIMIT));

StopBT();
closeStreams();

if (tries <= TRYLIMIT) {
    return true;
} else {
    return false;
}

private boolean CheckConfig(boolean commonSettingsTemp,
                            boolean individualSettings1Temp,
                            boolean individualSettings3Temp) {
    // MEU

    int maxBytesRead = 20; // In fact, any number should work. If too big, buffer will get error sooner (because of make filling, just the fact that it can be filled gives an exception), if too small maybe it will mean too much work. I've been working with a 40, which is very big and unusefull, with no problems.

    int bufferSize = 400; // If the commands are sent fast (short sleeps), it's not necessary to have a big one.

    // Not user configurable from here
    int offset = 0;
    byte[] buffer = new byte[bufferSize];
    int nrBytesRead = 0;
    int pointer = 0;

    commonSettings = !commonSettingsTemp;
    individualSettings1 = !individualSettings1Temp;
    individualSettings3 = !individualSettings3Temp;

    while (true) {
        try {
            if ((offset + maxBytesRead) < bufferSize) {
                nrBytesRead = inputStream.read(buffer, offset, maxBytesRead);
            } else {
                nrBytesRead = -1;
            }
        } catch (Exception ex) {
            int k;
            k = 1;
if (nrBytesRead > 0) {
    offset += nrBytesRead;
}

// Algorithm to detect commands
while ((pointer + 4) <= offset) {
    if (buffer[pointer] < 0 && (buffer[pointer + 1] < 0)) {
        if (buffer[pointer] == -126 && (buffer[pointer + 3] == 0)) { // OK
            if (buffer[pointer + 1] == -128) {
                commonSettings = true;
            } else if (buffer[pointer + 2] == 31) {
                individualSettings1 = true;
            } else if (buffer[pointer + 2] == 33) {
                individualSettings3 = true;
            } else {
                pointer = pointer - 2;
            }
        } else if (buffer[pointer + 1] == -127) { // FAIL
            if (buffer[pointer + 2] == 30) {
                commonSettings = true;
            } else if (buffer[pointer + 2] == 31) {
                individualSettings1 = true;
            } else if (buffer[pointer + 2] == 33) {
                individualSettings3 = true;
            } else {
                pointer = pointer - 2;
            }
        } else {
            pointer = pointer + 4;
        }
    } else {
        pointer = pointer + 1;
    }
}

if (commonSettings && individualSettings1 && individualSettings3) == true) {
    return true;
} else {
    pointer = pointer + 1;
}
if ((commonSettings && individualSettings1 && individualSettings3) == true) // Probably not needed, always if it arrives here it should be return false;
{
    return true;
}
else {
    return false;
}
}

private void writeCommand(byte[] command, int cmdLength) {
    try {
        try {
            Thread.sleep(250); // Time needed so the BT device has time to configure. 250ms is A LOT, but it's ok to make sure the configuration is OK.
        } catch (Exception e) {

        }

        byte[] commandToWrite = new byte[cmdLength + 1];
        for (int byteNo = 0; byteNo < command.length; byteNo++) {
            commandToWrite[byteNo] = command[byteNo];
        }
        commandToWrite[cmdLength] = (byte) 0xFF;
        this.outputStream.write(commandToWrite);
    } catch (Exception ex) {
        excHand(e); // Exception handler
    }
}

public void searchDevices() {
    try {
        discoveryAgent.startInquiry(DiscoveryAgent.GIAC, this);
    } catch (Throwable e) {
        excHand(e); // Exception handler
    }
}
```java
public class FileProcessor {

    public void run() {
        int nrBytesRead = 0;
        byte[] buffer;
        buffer = new byte[2];
        short buffer1;
        short buffer2;
        short output;
        boolean readOk = false;
        boolean noMoreData = false;

        do {
            do {
                try {
                    nrBytesRead = inputStream.read(buffer, 0, 1);
                } catch (Exception ex) {
                    System.out.println(ex);
                }

                if (nrBytesRead <= 0) {
                    noMoreData = true;
                } else {
                    if (buffer[0] < 0) {
                        try {
                            nrBytesRead = inputStream.read(buffer, 1, 1);
                        } catch (Exception ex) {
                            System.out.println(ex);
                        }

                        if (nrBytesRead <= 0) {
                            noMoreData = true;
                        } else {
                            if (buffer[1] >= 0) {
                                readOk = true;
                            } else {
                                try {
                                    nrBytesRead = inputStream.read(buffer, 0, 2);
                                } catch (Exception ex) {
                                    // Handle exception
                                }
                            }
                        }
                    }
                }
            }
        }
    }
```
```java
System.out.println(ex);
}
if (nrBytesRead <= 1) {
    noMoreData = true;
}

while (readOk == false && noMoreData == false);
if (noMoreData == false) {
    readOk = false;
    buffer1 = (byte) (buffer[0] & 0x7F);
    buffer1 = (short) (buffer1 << 7);
    buffer2 = buffer[1];
    output = (short) (buffer1 | buffer2);
    buffer[0] = (byte) ((output & 0xFF00) >> 8);
    buffer[1] = (byte) (output & 0x00FF);
    try {
        outputStreamProcessed.write(buffer, 0, 2);
    } catch (Exception ex) {
        System.out.println(ex);
    }
} while (noMoreData == false);
processingFinished = true;
```
public int InitialiseFile(String programName, String path, String fileName) {
    try {
        this.CreateFolder("", programName);
        this.CreateFolder(programName, path);
        this.createFiles(programName, path, fileName);
        this.inputStream = this.dataFileConnection.openInputStream();
        this.outputStream = this.dataFileConnection.openOutputStream();
        this.inputStreamProcessed = this.dataFileConnectionProcessed.openInputStream();
        this.outputStreamProcessed = this.dataFileConnectionProcessed.openOutputStream();
    } catch (Exception ex) {
        System.out.println(ex);
    }
    return recordNumber;
}

SettingsHandler.java

public class SettingsHandler {

public boolean ReadSettings() {
    int size = 0;
    byte[][][] Settings = null;
    int[][] length = new int[numberSettings];

    OpenRecordStore(true);

    try {
        if (numberSettings != recordStore.getNumRecords()) {
            int record = recordStore.getNumRecords();
            return false;
        }
    }
```java
for (int i = 1; i <= numberSettings; i++) {
    if (recordStore.getSize(i) > size) {
        size = recordStore.getSize(i);
    }
}

Settings = new byte[recordStore.getNumRecords()][size];

for (int i = 0; i < numberSettings; i++) {
    length[i] = recordStore.getRecord(i + 1, Settings[i], 0);
}
}

} catch (Exception ex) {
    System.out.println(ex);
}

btAddress = new String(Settings[0], 0, length[0]);

if (Settings[1][0] == 1) {
    btAuthenticate = true;
} else {
    btAuthenticate = false;
}

if (Settings[2][0] == 1) {
    btEncryption = true;
} else {
    btEncryption = false;
}

if (Settings[3][0] == 1) {
    btMaster = true;
} else {
    btMaster = false;
}

recordPath = new String(Settings[4], 0, length[4]);
recordFile = new String(Settings[5], 0, length[5]);
btFriendlyName = new String(Settings[6], 0, length[6]);

CloseRecordStore();

return true;
}

public void WriteSettings() {
```
byte[][] settings;
settings = new byte[numberSettings][];
settings[1] = new byte[1];
settings[2] = new byte[1];
settings[3] = new byte[1];
settings[0] = btAddress.getBytes();

if (btAuthenticate == true) {
    settings[1][0] = (byte) 1;
} else {
    settings[1][0] = (byte) 0;
}

if (btEncryption == true) {
    settings[2][0] = (byte) 1;
} else {
    settings[2][0] = (byte) 0;
}

if (btMaster == true) {
    settings[3][0] = (byte) 1;
} else {
    settings[3][0] = (byte) 0;
}

settings[4] = recordPath.getBytes();
settings[5] = recordFile.getBytes();
settings[6] = btFriendlyName.getBytes();

try {
    if (recordStore != null) {
        if (recordStore.getName() != null) {
            CloseRecordStore();
        }
    }
} catch (RecordStoreNotFoundException e) {
}

try {
    RecordStore.deleteRecordStore("Settings");
} catch (RecordStoreNotFoundException ex) {
} catch (Exception ex) {
    System.out.println(ex);
}

OpenRecordStore(true);
try {
    for (int i = 0; i < numberSettings; i++) {
        recordStore.addRecord(settings[i], 0, settings[i].length);
    }
} catch (Exception ex) {
    System.out.println(ex);
}
CloseRecordStore();

Synchronizer.java

public class Synchronizer {

    public synchronized boolean Write(OutputStream outputStream) {
        while (newContent == false) {
            if (readFinished == true) {
                writeFinished = true;
                return true;
            }
            try {
                wait();
            } catch (InterruptedException e) {
            }
        }
        try {
            outputStream.write(buffer, 0, nrBytesRead);
            outputStream.flush(); // Just to be sure, even if it looks like it works without it.
        } catch (Exception ex) {
            System.out.println(ex);
        }
        newContent = false;
        notify();
        return false;
    }
}
public synchronized boolean Read(InputStream inputStream) {
    while (newContent == true) {
        try {
            wait();
        } catch (InterruptedException e) {
        }
    }

    try {
        nrBytesRead = inputStream.read(buffer, 0, 10);
        if (nrBytesRead < 0) {
            readFinished = true;
            notify();
            return true;
        }
    } catch (Exception ex) {
        System.out.println(ex);
        readFinished = true;
        notify();
        return true;
    }

    newContent = true;
    notify();
    return false;
}
Bibliography


[3] Mail4Me, open source API that allows J2ME/MIDP devices to access the e-mail service, http://mail4me.objectweb.org


[5] Enciclopèdia catalana


