Android Bluetooth monitoring

Hendrik Mende

Fachbereich Elektrotechnik

University of Applied Science Münster

A thesis submitted for the degree of

Bachelor of Science

2012

I dedicate this thesis to Braximo and the Abraxianer, great friends.

Acknowledgements

Foremost, I would like to express my gratitude to my supervisor Oresti Baños Legrán for the support of my Bachelor studies and research, his guidance, motivation and enthusiasm. His continued advice and encouragement kept me focused.

I must express my gratitude to Prof. Dr.-Ing. Peter Glösekötter from the University of applied science Münster and Ignacio Rojas Ruiz from the Universidad de Granada, who made this thesis possible.

I am grateful for the support from my friends and family, especially Hanna, Cati and Andreas with whom I spend a quality time and was able to explore and enjoy the many faces of Granada.

Finally I would like to thank the Fachbereich für Elektrotechnik at the University of applied science Münster and the Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación at the Universidad de Granada for giving me the opportunity to hold my final semester abroad.

Abstract

The ever progressing technological development of mobile phones, sensor technology and wireless communication systems have made a ubiquitous, permanent mobile monitoring of vital signs of the human body possible in recent times. Sensor devices are now small enough to be worn on the body and are able to provide data via a wireless network. The increasing computational performance of smart phones make processing and analysing of sensor data in a real time manner available. However the technological advance cannot yet foreclose cardiovascular diseases, the number one cause of death globally [1]. With these increasing technological developments and improvements it becomes more appealing to create a mobile system for monitoring vital signs, that can be worn by anyone, anywhere in the world.

This bachelor thesis presents an Android application that works together with a smart phone device and a wearable sensor which transfers obtained data as well as derived parameter values. Information aggregation and analysis of the provided data in real time are made under the coverage of a Bluetooth based wireless network. The system proposed here does not rely on any stationary components such as an Internet connection or a wired technology for processing of data and thus can be applied everywhere. It is capable of storing the recorded data and deposit them onto a remote server. This provides the possibility of an analysis of the data at a later stage.

Die immer weiter fortschreitende technologische Entwicklung von Mobiltelefonen, Sensoren und kabellosen Kommunikationssystemen, haben eine allgegenwärtige, dauerhafte mobile Überwachung der Vitalzeichen des menschlichen Körpers, in jüngster Zeit ermöglicht. Die Sensoren sind mittlerweile klein genug, um sie am Körper zu tragen und sie sind befähigt, Daten über ein kabelloses Netzwerk bereitzustellen. Die steigende Rechenleistung von Smartphones machen eine Verarbeitung und Analyse von Sensordaten in Echtzeit möglich. Allerdings kann der technologische Fortschritt bisher nicht Herz-Kreislauferkrankungen, die häufigste Todesursache auf der Welt, verhindern [1]. Mit den genannten technologischen Entwicklungen und Fortschritten wird es immer reizender, ein mobiles System zur Überwachung der Vitalzeichen zu Erschaffen, welches von irgendjemandem, irgendwo auf der Welt getragen werden kann.

Diese Bachelorarbeit präsentiert eine Android Anwendung, welche mit einem Smartphone und einem tragbaren Sensor zusammenarbeitet, welcher gewonnene Daten, sowie abgeleitete Parameter, bertrgt. Zusammenfassung und Analyse der bereitgestellten Daten in Echtzeit wird im Rahmen eines Bluetooth basierten kabellosen Netzwerks vollzogen. Das hier vorgeschlagene System hängt nicht von stationären Komponenten wie einer Internetverbindung oder kabelgebundener Technologie fr die Weiterverarbeitung von Daten ab und kann daher überall angewandt werden. Es ist dazu befähigt die aufgenommenen Daten zu speichern und auf einen Server abzulegen. Dies bietet die Möglichkeit einer späteren Analyse der Daten.

Contents

List of Figures

Chapter 1

Introduction

In this Bachelor thesis an Android application for monitoring vital signs through sensor data is introduced. First of all the motivation and the context of this work are illustrated, followed by the objectives in which a comprehensive description of the project is given.

1.1 Motivation and context

This thesis is focused on the monitoring of a patient with the help of mobile technology. In the context of health care systems around the world, two situations can be found [43]. On the one hand, health care systems in developed countries are facing budget cuts, due to the current financial crisis, amongst a growing number of elderly patients. On the other hand, health care systems in developing countries are facing challenges in providing health care, due to under-supply. The need for an efficient solution exists. To face the challenge of these conditions, mobile technology could be the solution. With the availability all around the world and popularity of smart phones, a platform for an application could be found in this technology. An additional sensor for acquiring information from the human body, forms together with the smart phone and the application a solution to a world wide problem.

Figure 1.1: On the left hand side, a patient is waiting for treatment in a Hospital in Houston, Texas, USA[44]. On the right hand side, a queue of Sudanese waiting in front of a clinic[45].

1.2 Objectives

The objective of this work was to develop an application that connects a smart phone with a multi-parametric sensor. Between the two devices, data should be exchanged.

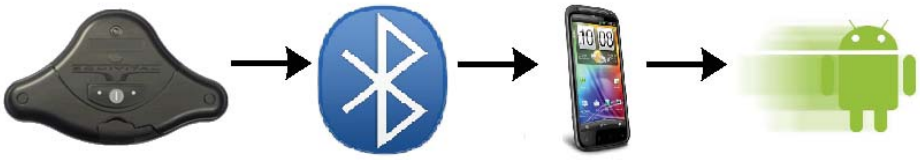

Figure 1.2: From left to right, the incorporated parts in this project are the Equivital EQ01 sensor, a Bluetooth connection, an HTC Sensation smart phone and the Android Software platform.[46][47][48][49]

Furthermore we agreed on the following conditions the project should fulfil:

- A target platform and mobile operating system should be chosen. Even though from the beginning on it was clear that the application would only be useful to users with a vital sign sensor for monitoring, the development process was carried out with the requirements of a commercial application in mind. This would include to develop on a popular software platform and with well performing development tools.

- Incorporate a mobile health care sensor system that is able to connect to a smart phone over a wireless network. It should deliver various information gathered from the human body.
- Develop an application with the ability to connect the two devices with Bluetooth technology. This will ensure that nearly all smart phones can be used for this application. A requirement was to keep a connection throughout the whole application and even outside of it until manually stopped.
- Display numerous values that the sensor provides. These include average data as well as real time data plotted into graphs and waveforms.
- Provide user registration and login to enhance the applications features and user experience. Additional information from the subjects can be further used for risk analysis.
- Collected data should be stored on the smart phone. A structure for a database should be chosen. This way health care experts and supervisors can give a feedback and analyse the monitored subject over a longer lapse of time.
- Upload gathered data to a remote server. This provides the possibility of reviewing data from anywhere around the world. Additionally uploading the stored data can save memory space on the smart phone.
- For ease of use and to cover two possible user groups, two distinct user interfaces should be implemented. In detail the coverage of a group of more experienced users and a group of regular laymen can reach a bigger group of potential users.

1.3 Thesis structure

This thesis is divided into the following chapters:

- In Chapter 1 (the current one) I present the main motivation for this thesis and enumerate the thesis objectives and main contributions.

- In Chapter 2 entitled "State of the art", I will do a review on current developments in health care systems, mobile health care sensors and mobile operating systems for smart devices.
- In Chapter 3 "tools used for this project", I present the hardware and software used to develop this application.
- The application will be explained in more detail in Chapter 4 "Project description". All development steps and functionalities of the application are depicted.
- Chapter 5, "Conclusions and future work", discusses the accomplishments during the development of this work and gives an outlook of future possibilities.

Chapter 2

State of the Art

In this chapter I will present the current state of the art for the most important project parts. These are the theoretical background of mobile health care and its current outlook as well as the state of today's health care systems and its defects. Also presented are the most recent mobile operating systems on the market. In later parts of this chapter I will give an overview of current developments of mobile health care sensors, as well as present network types and a similar project called "Droid Jacket".

2.1 Public health service

Looking at the current health care systems in first world countries and leading economies, the amount of money spend on health care is relatively small in comparison to the expenditure for discharge of debt or national defence [50]. In Germany for example, the federal budget for the year 2012 was 306.2 billion Euro. Only 4.7 per cent were spend on healthcare, whereas nearly 5 times as much money was spend on reduction of debts and defence of the nation. On the other hand, the government of Spain had to cut cost due to the current financial crisis by reducing its budget for public health service by 13.7 per cent in the beginning of this year[51]. Considering the debt of the health care system was already at 16 billion Euros, a challenge towards providing care of patients and supply of medical goods will be foreseeable in the future. Another problem the health care

Figure 2.1: Public health service[2]

system has to face is the ageing population and the involved rise in care-givers efforts. Especially in Germany, where the average age of the population has risen to near 45 per cent, and accordingly one fourth of the population is older than 65 years[52]. A solution to the problem of cutting cost while at the same time raising the effort of care-givers, lies in the increase in efficiency new technologies can deliver. A system that can continuously monitor the health condition of elderly people and share information with remote care providers or hospitals can now be realised with the help of mobile technology. Smart phones are cut out to provide these characteristics of an ubiquitous, permanent monitoring. The range of available smart phones on the market offers a solution in all price ranges and are available nearly everywhere in the world. The devices and their markets are consumer oriented and therefore developed to fit the demands of the customers. The potential remote monitoring technologies has, can be seen in analysis by the Brookings Institution, which estimates the savings for health care in the next 25 years in the United States to be as much as $197 billion[53]. Chronic diseases like diabetes or congestive heart failure were amongst the ones, where the most costs

could be saved. The allocation of improved access to supervision of patients, improved system efficiency and diagnoses accuracy, as well as increasing awareness of health with the help of the remote monitoring technologies can be achieved. Additionally people that had no access to health care because of missing personnel, can benefit from a mobile monitoring system that is widely spread. In the literature and articles that can be found on the Internet regarding a continuously health watch, a lot of differing expressions rave around. From telemedicine over mobile health (mHealth) to pervasive health care or health monitoring, all of them describe the supervision of a patient and their vital signs with the help of computers or smart devices. The application of these theoretical definitions are carried out already in various mHealth initiatives all around the world[43]. Four projects from four different countries are introduced here:

- In Kenia, WelTel provides an mHealth solution that supports HIV patients and care-givers with assistance. They provide information and knowledge, as well as disposal of medicine to regions with little to no medical provision and offer remote data collection[54].

Figure 2.2: WelTel logo[3]

- In India the non-profit organization Apollo Telemedicine Networking Foundation, connects community health care-givers with professionals at the Apollo Hospital in Chennai over mobile phones, to provide assistance. Monitoring of patients in intensive care units (ICU) and mobile devices are provided[55].

Figure 2.3: ATNF Logo[4]

- In Romania, the application SkinVision can identify melanoma on the skin with the help of the in build camera. It provides additional information to the skin disease, suggest a doctor nearby and is capable of identifying changes over time. The application is available for android and iOS[56].

Figure 2.4: SkinVision[5]

- In the United States, a pill bottle cap that is connected with a pharmacy is

developed by the company Vitality and is called GlowCaps. A wireless chip inside the cap reminds the user to take a pill when he has to with sound and light signals, connects to a pharmacy and is used to inform the user with a report via email[57].

Figure 2.5: Vitality GlowCaps[6]

On the flipside, clinicians are worried that the use of mobile applications or technologies build around it, make the patient too independent[58]. They fear the way medicine is practised will change for the worse, as patients would no longer consider the opinion of a professional but rather that of their smart phone. This would the lead to the abolition of the classic occupation of doctors and individual diagnosis.

2.2 Mobile operating systems

With the world wide proliferation of smart phones, the market for mobile operating systems has grown. The choice of mobile operating systems is wide in current markets. First of all I would like to take a closer look at what an mobile operating system is, before looking at the available Software. A mobile operating system, also mobile OS, is a system consisting of programs and data, that is capable of operating digital mobile devices like a smart phone[59]. It is a piece of software that manages operations, controls and coordinates the use of hardware as well as sharing the device's resources. It is part of layers that link applications and

Figure 2.6: Various mobile operating systems[7]

hardware. The abstraction of layers make it possible to have the same behaviour of applications on all smart phones that use the same mobile OS. From former to latter the layers are:

- The Kernel, representing the core Software or the actual operating system. As described above it is directly linked to the hardware via drivers and manages processes, filesystem and memory. The following software layers form the software stack, which sits on top of the OS.
- The middleware layer provides additional services to software applications like for example communication engines or data storage. This layer is not visible for the user but functions as a library to the software system with useful features.
- The application execution environment(AEE) provides the ability for external developers to create their own applications with the help of application programming interfaces (API).
- The user interface framework provides a set of graphics components that make interaction with the user possible.
- The application suite is a collection of software with which the user interacts most of the time.

All these software components combined form a so called distribution. All mobile OS are delivered with already built in applications that can do some of the most desired functionalities like web browser, mail services and phone calls. The manufacturer provides a software development kit (SDK) that is a set of tools for developing applications on the desktop computer. Even though mobile OS are getting more and more closer to the native Desktop OS in terms of functionalities, they differ from them because of their constraints and restrictions in the phones physical size. Rare amount of power provided by small sized batteries, pocket-sized screens and accordingly sized processors and memory are some of the drawbacks a smart phone faces in terms of computational performance. Therefore mobile OS are oriented towards the use of wireless communication and mobile multimedia data.[60]

2.2.1 Comparison of mobile operating systems

The market of mobile operating systems has grown significantly in the last five years. By now more than 3 billion people own a smart phone. In the first quarter of 2012 nearly 420 million smart phones have been sold. With the increasing number of users and availability of smart phones, the choice of mobile operating systems has grown likewise. Taking a look at the market for mobile OS in the second quarter of 2012 (figure x), 94 per cent of the market share was held by only four operating systems[61]. Namely these are Googles's Android, apple's iOS, RIM's blackberry OS and Nokia's Symbian. Microsoft has also gained more popularity and favourable reviews from developers in recent time and is about to launch a new Version of its OS this year. Out of these five, Android is the most popular with a rising number of devices sold with its OS of 81 million units in the first quarter and 98 million in the second quarter. What are the reasons for Android's success over the other OS? In the next section we will take a look at the five OS and go into detail about their strengths and weaknesses until in the end we conclude about them.

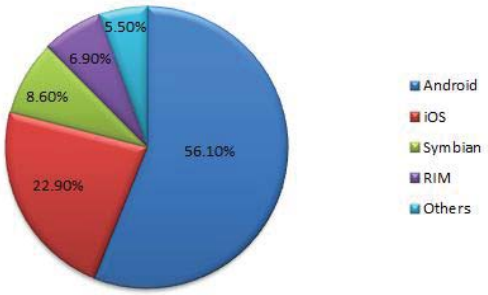

Figure 2.7: Global mobile Operating systems market share 2012[8]

Figure 2.8: Nokia Symbian OS[9]

2.2.2 Nokia's Symbian

Symbian is developed by Nokia in partnership with Accenture. The newest build is called Nokia Belle and is open source licensed under the Symbian foundation. Symbian evolved from a hand held OS to a smart phone OS and its components were specifically designed for this purpose, even though it has similarities to a desktop OS in regards to multitasking, safety aspects and memory management[62]. It is a successor to Pison's EPOC OS and has adapted some of its features. Therefore it supports desktop functions like multi threading. For the development of applications on this platform, c++ is used but it can use other programming languages as well. A software development kit is available which

has an emulator for testing purposes build in and can be made use of with other programs. Native applications can be brought onto the smart phone via one of the many app stores it supports or can also be directly downloaded onto the device. For web applications there is the possibility of putting them directly onto the Symbian platform. One big disadvantage of Symbian is that it is only capable of running on ARM processors[63]. While ARM processors are widely spread in the manufacturing sector, it still restricts the Symbian platform to specific requirements. Another thing that has been criticized by users is the complicated procedure of switching the language to ones that use non-Latin alphabets such as the cyrillic script which is used in Russian or Ukranian language for instance[64]. Taking a closer look at its architecture reveals different implemented layers. From top to bottom the five layers of the Symbian platform are[65]:

- The UI framework is the direct layer between the Symbian OS and the smart phones UI. It lies on top of the OS and provides the tools needed for user interface construction like libraries, plug ins and collections. For application support and user interface customization, the framework defines basic general user interface behaviour which is provided by five components. These are namely the Uikon, the control environment (CONE), the FEP Base, the UI look and feel and the Uikon Error resolver plug in.

- The application services layer is the second layer beneath the UI framework. For launching and installation of applications on the smart phone or other functionalities, this layer provides the needed services. It delivers basic application architecture relationships that are also used by the JavaME components.

- The JavaME Subsystem offers an optional Java implementation to the Symbian Platform. This implementation consists of configurations, profiles and optional packages that make use of a Java Runtime Execution (JRE) environment, a Java Virtual Machine (JVM) and class libraries to provide the basic features of a Java platform.

- The Os Services layer represents the middleware layer of the Symbian OS as it provides frameworks, servers and libraries to complete the subjacent

layers to form an OS. Some of the services are for communications or multimedia purposes as well as generic operations and connectivity.

- The Base Services layer extends the lowest layer, the kernel, to a basic software platform. This basic software platform is a bootable. Runnable and programmable system that can be embedded onto real hardware. These functions are provided by low-level libraries, servers, and frameworks and can be used by the system itself or by applications.

- The kernel services and hardware interface is the lowest layer in which the kernel and drivers for hardware are incorporated. Since Symbian OS v9 the EAK2 real time kernel is incorporated, where a single core processor smart phone can be used as the nano kernel provides enough functionalities. This has an advantage of saving manufacturing costs in smart phones, because single core processor are cheaper to manufacture.[66]

Even though Nokia announced going into partnership with Microsoft to support windows phone OS in their own smart phones as of February 2011, the Symbian OS is particular popular in third world countries. It has also been popular in the past, hence still many smart phones run on Symbian. In addition to these arguments, Nokia stated that Symbian and its development environment QT will still be supported and are therefore future proof. With the background of this thesis and the thought of having a wide spread everywhere to use application, Symbian was taken into consideration under these aspects[65].

2.2.3 Windows phone

Windows phone's first Version called Windows phone 7 was released in October 2010 as a successor to the Windows mobile OS released in 2002. Both are based on the Windows CE that was developed for embedded systems and hand held computers[67]. Windows phone is a closed system, much like iOS from apple meaning that the user or developer has no access to the source code. On the basis of the Microsoft .net software platform and the Microsoft Silverlight add-on for web browsers applications are developed. For a development environment or IDE

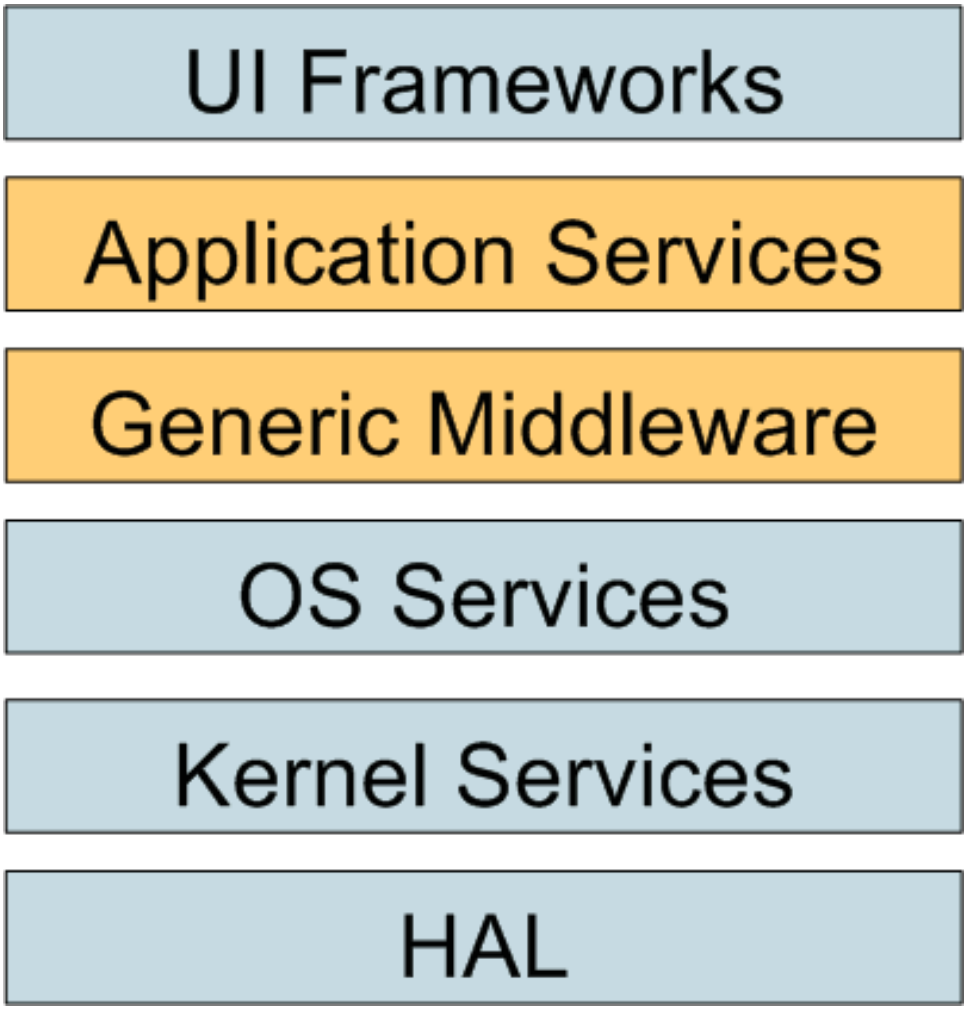

Figure 2.9: Nokia Symbian OS Layers[10]

visual studio is used in combination with an SDK provided by Microsoft. Applications are distributed over Microsoft's own marketplace the Windows mobile marketplace in addition to other marketplaces. The only possibility to connect the smart phone with the computer for data transfer is via Microsoft's software called Zune. It provides the possibility to synchronize your multimedia data with the device but does not allow for a direct access of folders or files. [68] During the process of developing, Microsoft tried to independently build up the operating system from its predecessor. The most recent version of this mobile OS is

Figure 2.10: Windows phone logo[11]

Windows phone 7.5. It offers multitasking Some drawbacks of the newer Version are for instance the inability for switching SD cards or using the SD card as an external storage device. Windows phone recognizes the available storage as a whole and is not able to give the user or developer the ability to independently address the storage on the SD card which limits the possibilities of feature rich applications. When switching SD cards the system will hand out a fatal error. Applications that were developed on older OS version are not compatible to run or be executed on the newer versions. This restriction for applications not to be applicable for future releases makes it a time consuming procedure to update and administrate older versions[69]. The platforms architecture is made up of four main components, from top to bottom these are:

- The Application layer has the frameworks, that give access to the platform functionality and to the components to write applications. Silverlight, XNA and HTML Javascript are used for this process. XNA is used for gaming development on Windows, XBOX and Windows Phone 7. Silverlight on the other hand is used for conventional applications and uses XML for the applications surface and C for the implementation of logics. Additional

contents can be displayed on the web browser using HTML Javascript. The Common Language Runtime (CLR) is the .NET environment for developing in C.

- The middleware layer consists of three components. One of the them is the App Model. It administrates the licences of the applications, does Software updates and does management of applications. It only allows applications to be installed which have a valid licence for the marketplace. The UI Model provides basic ui functions like 3d imaging. UI are managed in layers meaning a new view will be laid on top of the older ones. This is managed by the shell frame and provides an order of pages that are accessible through the session manager. For communication with services the cloud Integration provides APIs like Xbox live or Bing as the third component. The use of other self-made services is also possible.

- The Kernel is the connection between the underlaying hardware and the software stack above it providing access between the two. Responsibilities like storage management, networking functionalities and security models are embedded into this layer. The Kernel used in Windows phone 7 is the same as the one in Windows CE. In combination with the middleware layer this composes the core of the OS.

- The Hardware Foundation is the basic hardware requirements for the OS to run on. Microsoft is the only software developer that has requirements for their mobile operating systems and smart phones.

In the near future Microsoft will launch windows phone 8. Smart phones that have been manufactured for the older version will not be able to run on the new OS because Windows phone 8 uses a different kernel. The new Kernel will be similar to the one used in the desktop OS windows 8 and differs from the older Windows CE kernel. Regarding to Microsoft, there will be an update for older smart phones called 7.8 which will be compatible with Windows 8 stuff. [70]

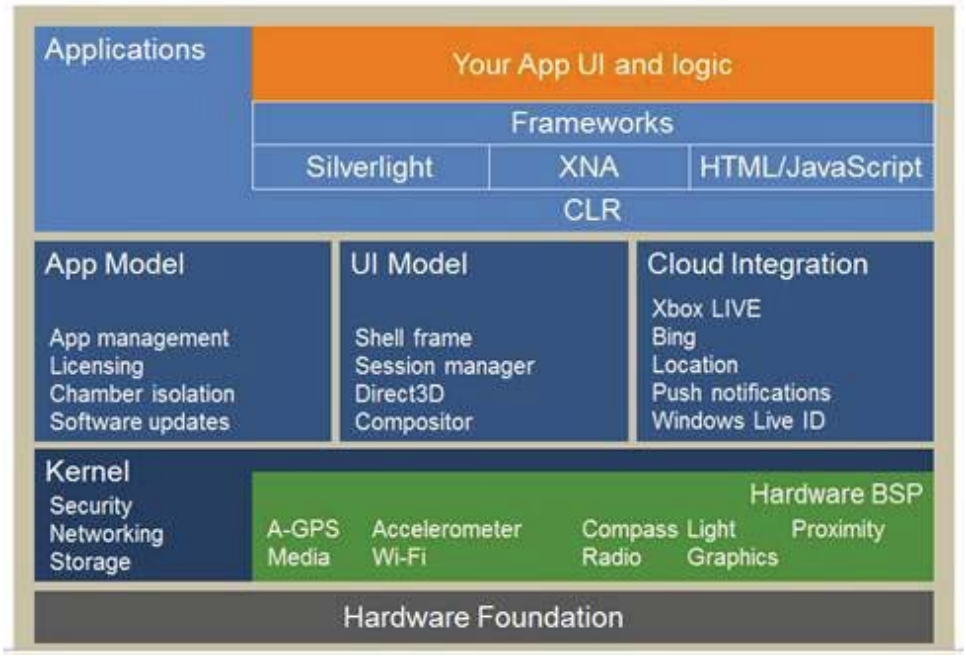

Figure 2.11: Windows phone architecture[12]

Figure 2.12: RIM Blackberry logo[13]

2.2.4 RIM's BlackBerry

The first Blackberry was released in 1999 by the manufacturer Research In Motion(RIM) and comprised the identically named mobile OS which still exists in its newest version 7.1. It was initially developed with a push mailing system and

focus on email services. These functionalities were well regarded by people in the business sector as it provided the user the usual functionalities of an organizer and communication service in one portable device with internet access[71]. The operating system itself was platform independent developed using C++. Newer Blackberry smart phones support the Java2me platform. This includes a Java Virtual Machine which was developed particularly for Blackberry and makes use of Connected Limited Device Configuration (CLDC) and the Mobile Information Device Profile (MIDP). The MIDP is used for developing so called MIDlet applications. These applications run on any device supporting the Java2me platform. A special feature to this is the secure access to applications on a remote server through the Mobile Data System (MDS)[72]. After a connection to the internet is made via WAP or Http the device links to RIM's Network Operating Centre(NOC) which connects to the BlackBerry Enterprise Server (BES) or BlackBerry Internet Service (BIS). The BES is a server solutions protected by an organizations firewall making a secure exchange of emails possible. It incorporates the MDS. Connecting to the BES makes the access to corporate email services or the MDS possible, therefore BES acts as the middleware layer, providing additional services to applications. MIDlet applications, as said earlier run on smart phones supporting the java2me platform or running the blackberry OS, these smart phones do not haver permissions to use the bes from blackberry though. Initially the OS was specifically designed around the user input types like trackball, scrollwheel and keypads that the blackberry smart phones were manufactured with. Today the blackberry OS also supports touch screens and has been adopted to modern technologies providing multitasking for instance.[72] For developers, Blackberry provides their own IDE. Applications can be programmed in Java. For the upcoming Blackberry 10 OS its also possible to use a variety of development languages like Java, HTML5 or even C/C++. Native applications are distributed over RIM's own applications store called app world. Even though there is support for many different languages, a big variety of smart phones to choose from and quite a big community of users, some developers have given up on Blackberry[73]. The biggest complain being its too costly and complex to develop for that many smart phone variants. Some smart phones still have scrollwheels and keypads while newer models only or additionally have a touch

screen. To make an application available to all users is too complex and would consume too much development time and money. Since the Blackberry os is not open source and software and hardware are both manufactured by rim, there is not much information about the underlying processes and theories behind the mobile OS.[71]

2.2.5 Apple's iOS

Figure 2.13: Apple iOS logo[14]

Apple has revolutionized the smart phone market with the launch of the iPhone and corresponding iOS on 9.1.2007[74]. Moving away from the solely business oriented smart phones like blackberry produced and making it more user interactive with just a single button and a big capacitive touch screen. At the first presentation and until 7.6.2010 the os was called iPhone OS as it was specifically designed for the iPhone. Later on it changed its name to iOS with the release of the fourth version, as other products from apple also used this mobile OS, like the iPad and the apple TV[74]. It has always been and still is restricted to apple products, which curtails the use of devices but provides Software that was developed close to the hardware. The very first version was based on the mac OSX desktop OS but has been optimized for touch pad devices and since than further improved with features like multitasking. Consequently both OS share the same frameworks which makes porting applications from OSX to iOS possible. With the delivery of the iPhone the user is supplied with readily installed applications like an email service, web browser and other programs. It

is also possible to run web based applications through apple's own web browser Safari. The newest version of the iOS the iOS 6 was released to the public on September the 19th this year with the disclosure of the iPhone 5[74]. Applications are developed utilising the programming language objective c. Apple is the only manufacturer and Software developer that uses this language to program applications. Therefore it is difficult to provide applications across other platforms, as there is no uniformly used programming language. Native applications are developed using the program called Xcode, which is only available for mac OSX. Completed applications can only be carried onto the smart phone with the help of apple's own app store. A one year submission will cost 99 Dollars, besides apple charging 30 per cent of the revenue from sold applications[75]. Before an application appears on the app store, apple reserves the right of censoring after checking uploaded applications. Some people have reported of causeless censorship of their applications by apple. The source code of the iOS is closed thus it is not possible to write new functions for the SDK. In this regard, developing for the iOS is in comparison to other platforms very cost intensive and cumbersome. The software architecture of the iOS is divided into four layers. Every layer provides features and frameworks that compose a complete mobile OS. The layers are the Cocoa Touch Layer, the Media Layer Core, the Services Layer Core and the OS Layer[75]:

- The Cocoa Touch Layer is most important and frequently used Layer. It provides key technologies like multitasking and touch-based input as well as many other so called high level system services like peer tp peer services and push notifications. The frameworks incorporated in this layer are the most commonly applied and outline the basic application infrastructure.

- The Media Layer provides the media services just like the name suggests. If you want to incorporate video contents, audio contents or 2d and 3d graphics you can find the technologies and corresponding frameworks in this layer.

- The Core Services Layer has key services that are used by all applications directly or indirectly without the user knowing as they are build up on top of them. Examples are storage, object and data management and data

protection services. Integrated in this layer are also frameworks for user accounts, advertisements and system configuration to name a few.

- The Core OS Layer is the base Layer of the iOS. It consists of frameworks and features that are accessed both from applications and frameworks in other layers.

Figure 2.14: Apple iOS architecture[15]

Some of these frameworks contain interfaces for data processing, Bluetooth connections and security management. For special cases where an application uses external storage this layer provides security and communication assistances. As typical for layers in a software architecture that are sitting right on top of the

hardware, this layer also features the system. The system comprehends the kernel, that manages threads and network communication, the hardware drivers, that act as an interface between hard- and software, and low-level UNIX interfaces. Only a limited set of applications and frameworks of the upper layers can access the System for security reasons.[75]

2.2.6 Google's Android

Figure 2.15: Android logo[16]

On the 5.11.2007 google announced Android, which remarks as today's most popular mobile operating system[61, 76]. It was developed in cooperation with the open handset alliance, which was founded to tackle the market of mobile smart phones with mobile OS. The OHA consists of 33 companies from 5 distinct sectors of the technology and business sectors. Mobile operators, semiconductor, software and commercialization companies including handset manufacturers are among them[77]. As of now the Android Open Source Project (AOSP) is governing the Android maintenance and the development cycle. The OHA and the AOSP have set goals to accomplish optimizing the infrastructure for the resources available on mobile phones. Jelly Bean or Version 4.1 is the most recent edition of Android. By increasing alphabetically order, the Versions are named after English sweets. Providing the code to everyone by making it open source and free makes it easy for developers to implement their own libraries. For the

development of applications the programming language Java is used, which is one of the most popular languages. The provided Software Development Kit (SDK) and Native Development Kit (NDK) can be downloaded from the android developers page. NDK is a tool set that provides driver development or other feature libraries programmed in a native language like c or c++. This gives additional flexibilities and possibilities to android developers. SDK includes tools for performance profiling and debugging as well as an emulator, making it necessary for the development of applications. This process is carried out using the Android Development Tool kit (ADT) provided by google, a development environment plug in for eclipse that boosts performance and is necessary for compiling[78]. Eclipse runs on nearly all available desktop operating system and is free. Applications can be tested on various emulators that are already integrated in the ADT plug in. Before distributing applications over the Android market, a registration fee of $25 has to paid. Once the application is online, google encashes 30 per cent revenue from sells if one decides to charge money for applications. There are also other possibilities of sharing applications through direct downloads or offer them on different application markets.[76] Taking a closer look at Android, it reveals to be more than just an operating system, but rather an open software platform for mobile development. Providing a hardware reference design, a run time environment, an application user interface(UI) framework and a system powered by a modified Linux 2.6 kernel, the Android software architecture is a competent performer. Android has its own c library implemented and a specific Java runtime engine, called the Dalvik Virtual Machine (DVM). The c library provides a set of functions and was only implemented because of licensing issues. Special features of the DVM are the fact that it runs register based instead of the typical stacked based architecture. Dual Core processor capabilities makes Android run fast on mobile devices that have a high technological level. Inside the architecture of Android's Software stack the programs exist and act inside a DVM Instance. With the help of the Kernel and the libraries, the DVM can now provide the hardware capabilities to the Android applications. A detailed description of the Android Software Architecture and its three Layers is given from top to bottom[76]:

- The Applications are not part of the OS and lies on top of the three Layers that form the Software Architecture. Android will be delivered with

a set of core applications that are capable of email services, web browsing, communicating via SMS or phone calls and other applications with helpful functionalities. They are made so that other applications can reuse their functionalities or services. This is accomplished and managed by the underlying Software Layers like the Applications framework.

- The Applications framework makes the upper layer up of the OS and provides services to interact with applications. The frameworks are written in Java. Some of these are for example the Activity Manager that manages all applications running solo or in different processes at the same time and their lifecycle, the package manager that administrates all installed applications and the content providers that is responsible for sharing and accessing data between applications.

- The specially made Libraries are written in native languages like c or c++ and offer services to applications like SQLite for Databases, OpenGL and SGL for 3d and 2d content or WebKit for accessing Websites. During the development process of the c Library a special thread implementation was integrated. It is capable of optimizing every thread's memory consumption and makes the start up of a thread faster. The developers have also set focus on optimizing the Library for the use in smart phones and the associated small processing speeds and limited available memory.

- The Android Runtime consists of two main components, the DVM and core libraries responsible for process management. The DVM is, as noted further up, a Java runtime engine or Java virtual machine that in essence is a Sandbox in which Processes are isolated from the environment. Again, like the Library the DVM is specifically designed for mobile devices and is register based. It executes programs that were written in Java by using a DX tool to convert it into bytecode that is readable by the DVM. The byte code is easier compiled and doesn't require much processing resource which makes it suitable for mobile devices. With Version 2.2 of Android the DVM was modified to become even more efficient. A Just in time compiler was added, that converts the byte code into more efficient machine code. The

performance gain this option delivers is in the ranks of two up to five times the usual speed.

- The Linux Kernel is the lowest Layer of Android's Software Architecture and is the abstraction layer between software stack and hardware. It manages processes and memory, as well as security, networking and energy consumption. For this purpose the Linux 2.6 kernel was chosen, as it is a proven open source design. Some modifications that have been applied to the original Linux kernel are for example the binder which allows for inter-process communications (IPC) or a more assertive power management.[78]

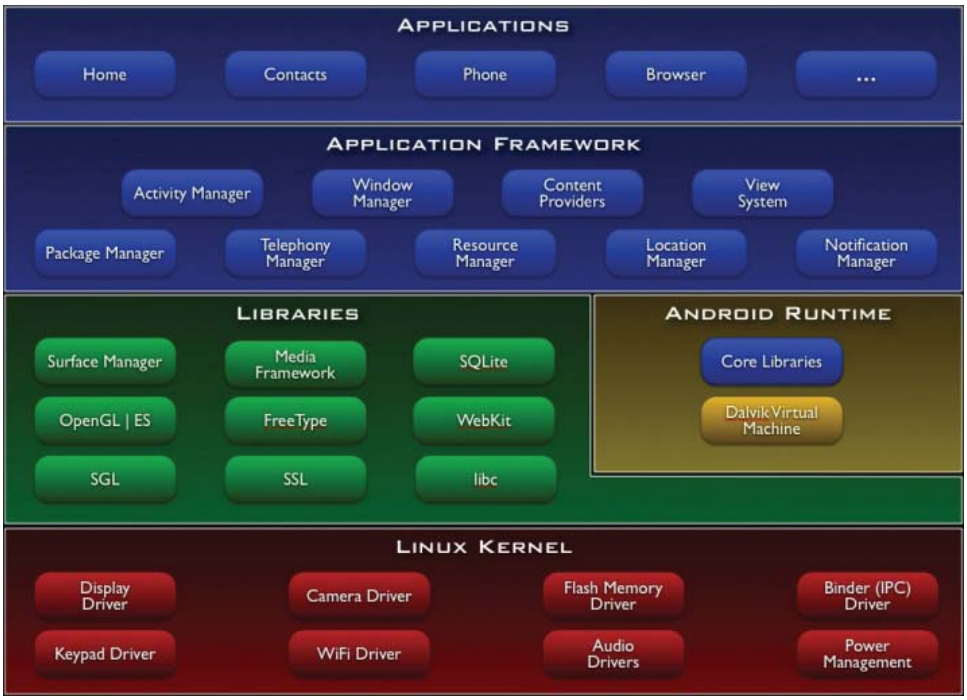

Figure 2.16: Android architecture[17]

2.2.7 Conclusion

Out of the five mobile OS that are presented here, Android delivers the best performance. Android is capable of handling dual core processors for best performances. Since it is wide spread, a lot of applications are available. Features

like the compatibility of between older and newer version's, multi tasking and the open source code provide the best requirements for developing. IOS and Symbian are limited to certain product models, whereas the latter is an obsolescent model with no future support. The proof for this can also be seen in the buyers response and the number of sales. It will be interesting though, to see how windows 8 can compete against Android once it has reached markets.

2.3 Mobile healthcare sensors

The Possibilities given by the technological developments and researches in the area of ultra small electronics could make for a more safer and easier way of life. Doctors no longer have to be visited to get a diagnosis or an examination, they are literally worn around your chest or on your body and notice any irregularities that occur. In the form of the smallest of sensors these devices may not even be noticeable to the person wearing them(user/patient) or are implemented into the body. Additional features like a build in defibrillator or a pump for blood pressure control or insulin injection can be embedded to provide a full package of pervasive health care. This science fiction scenario is to some extend already available with today's mobile health care sensors. Modern technology will probably not completely replace doctors in the foreseeable future but will definitely support them in their daily work routine[43]. The beginning of this development can already be seen in the way physicians use mobile devices like smart phones. A study shows that by this year, 2012, 81 per cent of physicians use smart phones[79]. At the time when the study was published, in 2009, the rate of penetration was 64 percent, which means an increase of 17 per cent in under 3 years. Here we have a development that could be caused by the familiarity to computers and popularity of smart phones by the younger generation. The candidness to explore upcoming technologies in the information sector, might push the popularity of ubiquitous health care furthermore along with the already present ubiquitous entertainment. With a predictable widespread application of mobile health care sensors among the younger generation, especially the elderly could benefit already from the available maturation of this technology. In the simplest of forms, vital signs can be continuously monitored by a caretaker or

Figure 2.17: Mobile Healthcare sensors in practice[18]

health professional to respond fast in case of an emergency. Not only can these devices be useful to make medical work more efficient or permanently but also to make it at least possible. These devices could also become very useful in regions where the trip to a doctor takes several hours or one doctor has to care about a few thousand patients[80]. Especially with the current rate of electronics getting smaller and cheaper at the same time, an affordable device could become reality in the near future. Already available are various medical and physiological monitoring, disability assistance or human performance management. While the choice of operation, analysis or feedback, and application, commercial or research as well as clinical or non clinical, are widely spread, the operation of available sensors are alike. Taking a closer look at the functionalities and the way sensors work, reveals a small network of sensors communicating with a base station or with each other around the body. These sensors can even be implemented under the skin or inside the body. In future thinking sensors could be powered off by

the energy the human delivers and communicate through body parts acting as antennas[81]. To obtain data from the body and distribute it throughout the BAN, health sensors generally obtain data from electrical signals emitted by the body parts and collected by electrodes resting on the skin. By definition a BAN uses little power for operation and communication purposes. One of the more popular available communication standards is Bluetooth or ZigBee regarding the GSMA list[82]. These two communication standards are thoroughly used in the industry but remain to consume too much energy in relation to their data rate to be ideal for a BAN. Still most of today's health sensors use these technologies, but other solutions for communication are provided. Some of them use techniques to store and forward the data, other use real time processing. Some are embedded, others are connected to a smart device. Which one is suited most for the demanded application depends on the requirements given. In our case these are a pervasive monitoring of vital signs with a device that is able to communicate with a smart phone.

2.3.1 Comparison of mobile health care sensors

Looking at the market for mobile health care sensors, an estimated number of 16 million devices will be sold by the year 2016, as predicted by ABI Research[83]. Accordingly, the popularity of these sensors is increasing. The choice of which sensor to use for certain applications boils down to the features and incorporated sensors. Some of the manufacturers already provide Software to examine the collected data on a smart phone or desktop computer, yet others provide supervision of professionals from remote servers. For choosing a suited device, a list of more than 240 different devices is provided by the GSMA list[82]. They can perform health checks and provide monitoring of vital signs simultaneously. The way these devices are worn are very different just as the way they obtain data from the body or communicate with base stations for further diagnosis and presentation. For a mobile health care sensor to be available in the United States and to get a professional certificate, the FDA can approve or disapprove a design. The Food and Drug Administration (FDA) is an agency of the United States Department of Health and Human Services[84]. The main function of the

FDA is to protect the public health of citizens of the United States. They check the safety and effectiveness of pharmaceuticals and products that fall under the category of health care, which are produced in or imported into the United States.

Extracted from the regularly updated GSMA list of commercially available devices in the healthcare sector, some of the more interesting performers are presented in this chapter. We will take a look at the successor of the Equivital EQ01, which was used in this project and as well have a look at some of the more recent and promising publishings of health care devices.

2.3.2 VisiMobile

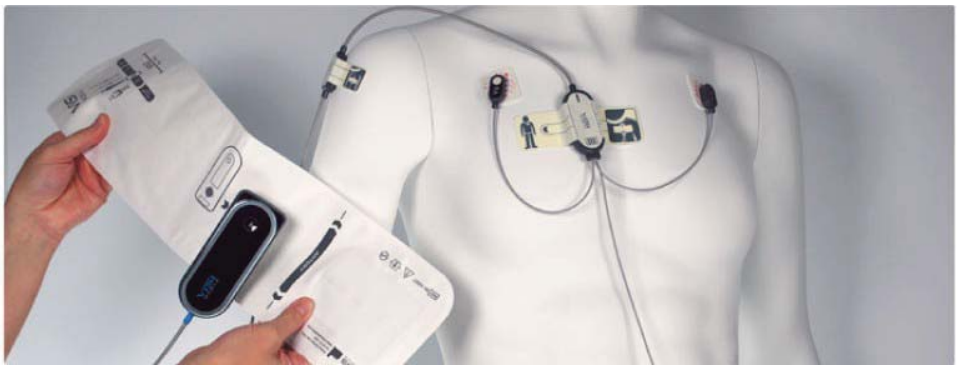

Figure 2.18: VisiMobile[19]

The ViSiMobile mobile monitoring system developed by Sotera wireless, was released in 2012[85]. Composed of three parts, ViSiMobile is delivered with sensors that are applied onto someone's body, a monitoring device that displays the information from the sensors and is worn around the wrist and a charging station. The sensors are the actual necessary parts that collect vital sign informations from the body in places like the thumb, the chest and the arm. A wired connection is made to the monitoring device to communicate the data and visualize it on the screen in a readable format. Connections from the ViSiMobile to other devices than the one worn around the wrist are possible with the help of a wireless network. Implemented into the device are a variety of functionalities, some of which

are not yet approved of by the FDA and are therefore not yet implemented in the available version[86]. The features that are currently monitored by the VisiMobile are a three or five lead ECG, heart rate, SpO2, respiration, non-invasive blood pressure (NIBP) and skin temperature. Awaiting approval by the FDA are for example Continuous non-invasive blood pressure (cNIBP) and Motion, realized by an accelerometer. Measurements for the listed features are based on Pulse-Transit-Time (PTT). Connections to smart devices like smart phones or hand held computers are done via wireless communication. Instead of using a mobile network like Bluetooth, the ViSiMobile uses WiFi which requires a stationary distribution system(router) to be able to make out and ingoing communications. To withstand the daily use in clinical applications the ViSiMobile was developed using robust materials[86]. The outer shell is made out of a special copolyester called Tritan by Eastman that provides a chemical resistive, durable and water proof housing. A water resistant sealant called GLS Versaflex OM 3060 TPE was used to protect the electronics of the device from water seepage. It meets IPX7 requirements of withstanding water submersion for 60 minutes at a depth of 1 meter. Before taking the ViSiMobile into operation an initial cuff-based calibration measurement has to be undertaken for measuring blood pressure. For this procedure a cuff has to be applied to the arm to measure the blood pressure. After calibration the cuff is removed and the measurement of PTT based beat-to-beat blood pressure is continued without it. Sotera is working on a solution, where the cuff won't be necessary to measure the blood pressure[85].

2.3.3 Equivital EQ02 LifeMonitor

Figure 2.19: Equivital EQ02 LifeMonitor[20]

This year Hidalgo announced the launch of the Equivital EQ02 LifeMonitor, a multiparametric sensor that provides physiological monitoring of a humanciteeq021. It is the successor of the Eq01 which was launched in 2010 and that is used in this project. Hidalgo itself was acquired by the Jaltek Group in 2008 which is part of the Cambridge wireless, an amalgamation of companies in the wireless technology sector from all over the world and has since then dedicated its work towards the development of the Equivital product range[87]. The main features of the EQ02 are the monitoring of the ECG, a three axis accelerometer, Respiration and skin temperature data. This data can be logged, transmitted and exported to other smart devices, computers or base stations. Connections to the former can be set up via wired or wireless connections. For wireless applications, Bluetooth is used with the SPP Protocol to manage connections. Additionally the EQ02 is capable of sending information about heart rate, body position or motion status and emergency notifications like alarms and alerts[88]. It can ex-

tract features from the acquired signals, for example the R-R interval from the ECG or recognize falls from the accelerometer. The Equivital EQ02 comes with a belt that is strapped around the chest. Once it is attached to the belt it sits on the left side of the body. Additional accessories can be acquired to extend the prowess of the EQ02. The available software for computer or mobile devices can display monitoring sessions in real time and present data extracted from the sensors. For developers flexible software platforms are available that incorporate development modules available for 3rd party applications[88]. Another customizable feature of the EQ02 is the integration with command and control systems to manage vital signs from a single or multiple patients. To accommodate/acquire more data from the body, other sensors from Hidalgo, wired or wireless can be integrated with the EQ02. This will extend its capabilities to measurements of the oxygen saturation, galvanic skin response, core temperature capsule, dermal temperature patch or GPS depending on which sensor is added. When the EQ02 is actuated with no additional sensors and in full disclosure logging mode, its battery lifespan will reach up to 52 hours. Multiple battery packs can be put in a recharging stack and later hot swap them into the EQ02 to quickly extend monitoring sessions. The internal memory of 8GB will make continuous data logging of up to 50 days possible[89].

2.3.4 Zephyr Bioharness

The Bioharness is a complete harness that is worn around the body. Comprehensive physiological data is captured from the wearer in real time. Connections from the Bioharness to other remote devices can be made with the help of fixed or mobile networks to transmit the collected data. For fixed network applications, a Wi-Fi connection is used. For mobile monitoring, a conventional Bluetooth connection is established, that uses the SPP protocol[90]. Optionally for the more advanced field applications, a VHF(very high frequency) channel can be used as a connection between remote devices and the Bioharness. This is a special feature of the Bioharness ,sending gathered data over VHF channels from connected tactical radio's. Capable to connect to a variety of tactical radios, data is transmitted over the same channel as the voice or an additional channel, so

Figure 2.20: Zephyr BioHarness[21]

that voice communication between radios is still possible. Some tactical radios use Bluetooth to connect to the BioHarness, others use the side data port allowing voice through the top port[90]. To make this feature readily available on the different types of radios, a Radio interface device is available from Zephyr to extend the features of a radio to incorporate an additional channel for data transmission. This makes it especially interesting to the military or firemen, for having an overview of a task force and their vital signs. Accessories that are distributed by Zephyr for the Bioharness are available in Soft- and Hardware form. Software applications for any smart phone that display the vital signs of multiple

users and submit the data to the Zephyr Portal are readily available. The Zephyr Portal is a base application for desktop computers in which the incoming data from the harnesses is displayed and can further be transmitted to remote servers or electronic health records. Software developers also have the possibility of developing their own mobile applications on any smart phones. Zephyr provides a complete free SDK for this purpose. Hardware accessories include tactical radios and RID. As the only manufacturer of sensors that are described here, Zephyr provides detailed papers on the validation of the accuracy of their sensors. Measurements of the ECG and position are accessible through their website. The Bioharness was released in 2010 and has gained FDA approval in the same year during development stages[90].

2.3.5 RS TechMedic DynaVision

Figure 2.21: RS TechMedic DynaVision[22]

Netherlands based company RS TechMedic released their mobile monitoring device, the DynaVision series in 2010. It has obtained the CE certification and

FDA approval. Offered are a variety of Versions which all have a different amount of sensors and features implemented. Here we will have a look at the feature richest version, the DVM-010SG. It incorporates an ECG, heart rate, Plethysmogram, SpO2 and body temperature[91]. The DynaVision is a hand held device with wired sensor leads attached to it that are placed on the body. For measuring the ECG and SpO2, a variety of cabling systems with sensors are available. A three, five or twelve lead ECG cabling system can be attached to the body and hand held device, to acquire the different ECGs. Data may be stored on the device or sent over a wired or wireless connection to a remote server or other smart devices. The internal memory suffices for up to 30 days of recording[91]. For a distribution of data over wireless networks, options for Bluetooth and a built in mobile phone with SIM card to transmit data via GPRS are available. A GPRS transmission to a server will only result in local GPRS costs and provides a wide spread network that is readily available. Alternatively the DynaVision offers a USB port to connect to any computer over a standard USB cable. All data transmissions are processed in real time and can be accessed on servers with the Software RS TechMedic provides. The customer has the option to purchase the Software if a server is in their hands. Otherwise, for a fee of 5 Euro per unit and month, a Server can be rent or if the number of units for this option exceeds the recommended 25, a complete new server for 4500 Euros is offered by RS TechMedic. From the internet and with additional software it is possible to access the collected data. The base Software which is required to set up units and add other programs from RS TechMedic which supply manifold features, is called Monitoring. Other programs include applications for smart phones, 12 lead ECG processing, receive cardiac events or to analyse health and fitness for example. The DynaVision comes delivered with a charger only. Other accessories are available on purchase from the manufacturer. Different operation modes will discharge the internal battery faster or slower. Hence when using a GPRS connection the battery will last for up to 15 hours, with Bluetooth the lifespan will reach up to 24 hours of real-time monitoring[91]. When collecting the ECG in full disclosure mode the DynaVision will last for up to 68 hours active.

2.3.6 SHL's Smartheart

Figure 2.22: SHL Smartheart[23]

The Smartheart from SHL is going to be the first and only wearable, mobile, battery powered sensor that is capable of recording a full 12 lead ECG once it is released[92]. A belt is strapped around the body, so that the device can rest on the breastbone. With stationary devices a 12 lead full grade hospital ECG provides more accuracy than the usual single, three, four or five lead sensors. The very detailed recordings this device delivers make it possible for an in depth analysis of the signal. Irregularities that the Smartheart is capable of detecting are ischematic cardiac events and irregular or abnormal heart conditions[93]. Once these conditions are identified in a check that takes thirty seconds, an alert is send

to the user in real time. Connections between the Smartheart and smart phones or remote devices can be established using Bluetooth. The standardized SPP protocol is used for this purpose. Real time monitoring sessions of the ECG can be stored on the smart phone or mobile computing device for later examinations. Via Email the results of these sessions, for example a detection of an irregularity or identification of abnormal heart conditions, can also be send to a physician. This way it is possible to provide the patient with a supervision of a professional to undertake necessities to keep a healthy condition. SHL is going to provide this supervision from a professional once the Smartheart is going to be released. With a subscription of "way under $20 a month" , regarding to Shay Leibovitz, Vice President of SHL, further data analysis, reports, and other medical advice will be made available. The Smartheart product gained FDA approval in 2012 which gives it reputation and approval of release in the United States[93]. In the FDA 510(k) paper the use cases for the Smartheart is summarized as followed: "The Smartheart device is intended to condition an electrocardiographic signal so that it can be transmitted digitally via Bluetooth technology and cellphone or communication device to a remote location. The Smartheart device is designed to be used by a patient to transmit a 12 lead ECG and rhythm strip in real-time to enable review at a physician?s office, hospital or other medical receiving center." A date for disclosure of the Smartheart is not set yet, but fall 2012 is said to be the time of release[92].

2.3.7 Conclusion

After looking at the five mobile health care sensors, a clear superior device cannot be spotted. All of them have unique features that make them useful for an application as developed here. For example, the Equivital EQ02 offers the possibility of plugging additional sensors into the monitoring device, the SHL Smartheart can record a full 12 lead ECG and the Zephyr Bioharness can connect to a tactical radio to make use of a VHF channel to transmit data. There are also devices like Philip's IntelliVue and Zoll Medical's LifeVest which take the whole idea of monitoring a patient remotely a step further. They can also interact with the patient themselves when identifying problems or irregularities. The LifeVest for

example has a defibrillator incorporated, that takes actions when a cardiac arrest is detected. These devices are developed for the professional clinical market and too expensive and too advanced to be applicable for this project.

2.4 Body area networks

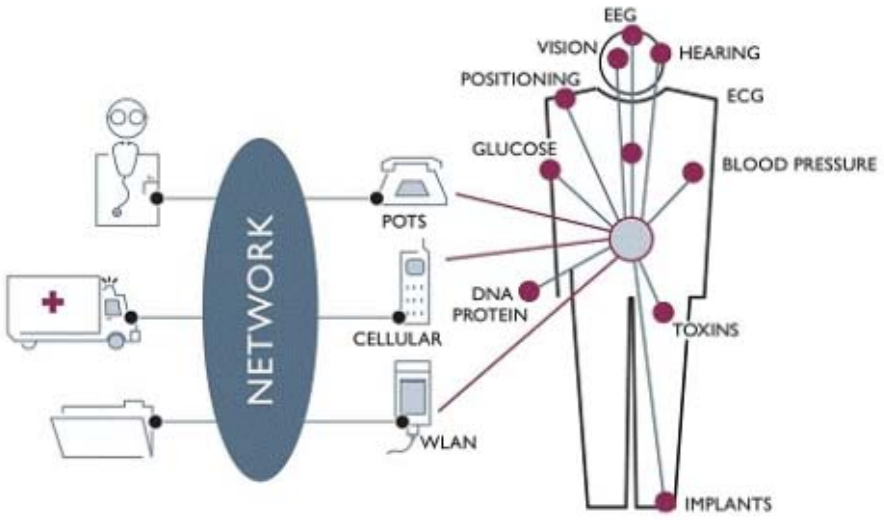

Figure 2.23: Example of a BAN[24]

For the purpose of transmitting data wireless over a short distance, the IEEE 802.15 working group came together in 2007 to develop a miniature system for data transmission around the human body. Defining the principles behind the idea of a small network is done with the so called Body area network(BAN). A Body Area Network is defined by IEEE 802.15 as, "a communication standard optimized for low power devices and operation on, in or around the human body (but not limited to humans) to serve a variety of applications including medical, consumer electronics / personal entertainment and other"[94]. In other words a BAN is a network of electronics around or inside the body to support the user. The applications that can be realised with this technology are very manifold[95]. Entertainment systems that interact with the body to control or react to an input

given to the user, sports applications or the use of a sensor system in a medical or clinical environment are all possible use cases of a BAN. For this project the BAN is established between the smart phone and the mobile health care sensor. The distance between these two is very small and in close proximity to the human body. Which communication standard is best suited can not only be determined by the requirements of low power consumption, but also by the availability throughout commercial devices. This aspect comes to mind, when thinking about the possibilities that are in scope of this project. If a ubiquitous vital sign monitoring should be carried out, it ideally has to be available to as many users as possible. Therefore the choices for networks to choose from is already limited. As specified by the IEEE working group, a BAN has to fulfil the requirements as described earlier. Not only do these technical requirements define a BAN but also some practical requirements like privacy and security, guarantee quality of service and bandwidth as well as being future proof. Especially in medical applications a guaranteed quality of service and reliability of the connection is very important. Today a number of communication standards for short range and low power consumption, fall into the range of possible candidates for an application based on a BAN. Namely these standards are ZigBee, ultra-wideband (UWB), Bluetooth and Wi-Fi.

- Bluetooth is based on a wireless radio system and is defined under the IEEE 802.15.1 standard. Its main design intentions were to replace cables with a wireless radio system. It establishes a network called piconet to connect a device with one or several others. The use of frequency hopping on the radio connection provides an interference free transmission of data. Bluetooth is mainly used to connect computer peripherals and smart device accessories like head sets[96].

- Ultra-wideband (UWB) is a short range indoor wireless communication standard that is capable of transmitting very big amounts of data over a radio communication. This is related to its very high bandwidth of up to 480 Mbps. Its high data rates makes it interesting for use with transmission of multimedia data. The communication is carried out as defined in IEEE 802.15.3[96].

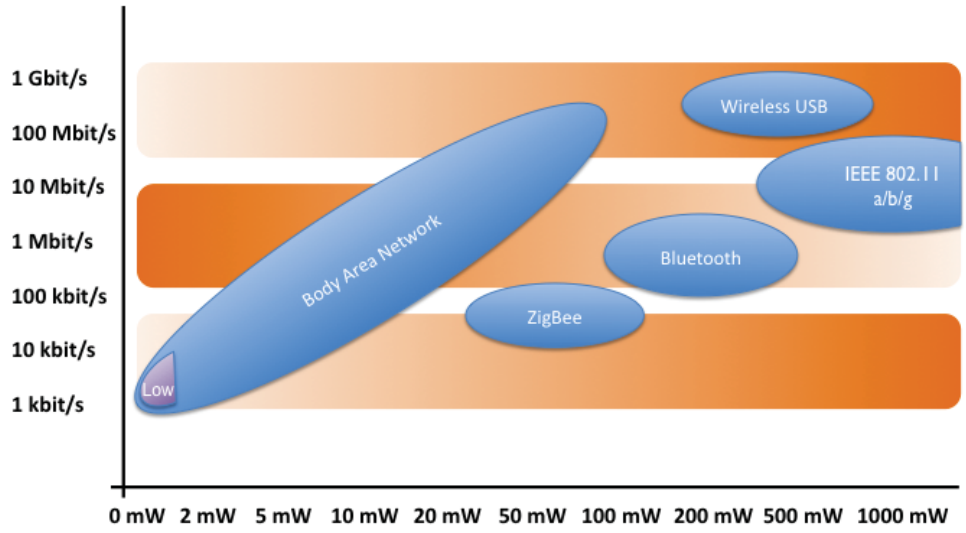

Figure 2.24: Power versus data rate in a BAN.[25]

- ZigBee is a small signal radio communication standard as defined in IEEE 802.15.4. The intended use is to control or communicate small amounts of data between or with household aids, which leads to very little power consumption. ZigBee organizes its own multi-hop network[96].
- Wi-Fi is a standard for radio communication. As defined in IEEE 802.11a/b/g it is used for wireless local area networks (WLAN). Wi-Fi is essentially a certification for WLAN products to ensure interoperability. With the help of a Wi-Fi a device can surf and make use of the internet[96].

As can be seen from the information given above, all standards might be used for establishing a BAN, considering their range. While Bluetooth and ZigBee have a more limited range, they also have less power consumption in comparison to the fixed station based Wi-Fi and the very high bandwidth UWB. Therefore UWB and Wi-Fi overshoot the goal of a low power consumption communication and not applicable for BANs. Between Bluetooth and ZigBee, does the former technology provide higher data rates, whereas the latter can establish a network with 65000 participants. Depending on application both technologies can be applied to a BAN. ZigBee is very useful for connecting a very high amount of for example

sensors incorporated all over the human body, while Bluetooth provides a high bandwidth to be used for a little number of devices around the body with a lot of data transmissions. A big advantage for Bluetooth over ZigBee is that it is already incorporated in most smart phones today and is very popular[97].

2.5 Droid Jacket

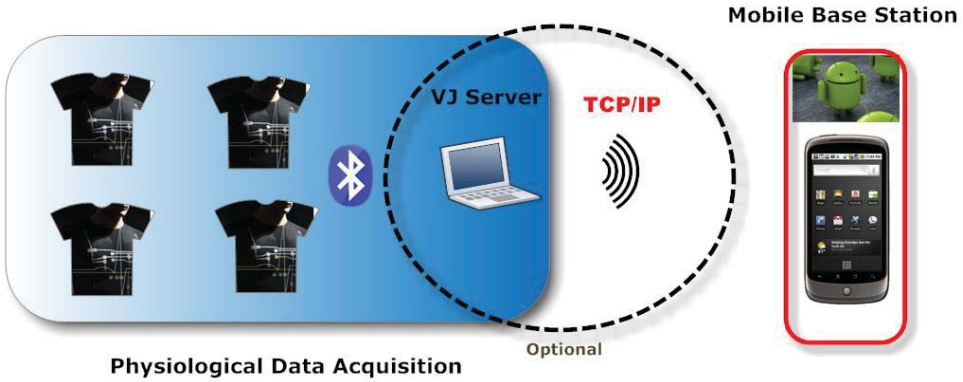

Figure 2.25: Droid Jacket overview[26]

The project Droid Jacket was developed to support first responders in their daily work routine by checking the vital signs for irregularities occurring from the influence of stress. If the evidence of a high stress level is confirmed, actions to get the first responders back into a save environment can be undertaken by analysing the extracted location data. Physiologic signals such as ECG, respiration, SPO2, galvanic skin response (GSR) and body temperature are monitored with the help of a sensor jacket called VitalJacket. These vital signs are an important indicator of stress levels that the extreme conditions and situations demand from first responders. The whole project is made up of three parts that were developed[26].

- The TeamMonitor is a system architecture that defines generic coordination modules and roles for a solution that will provide transmission, processing and storage of data. The modular system is made up of three parts, the application for an Android smart phone, the BIOsal framework and a Storage Provider responsible for database operations.

- The Biological Signal Acquisition Layer (BIOSal)is a modular framework written in Java. It has a session management and communication service incorporated to connect to several first responders and supports the development of additional plug ins. BIOSal can be deployed in a mobile device or desktop computer.

- The DroidJacket is made up of the application, a server and the hardware behind the two. The application enables the user for monitoring of vital signs, data aggregation, processing,visualization and optionally relaying services. Additionally the ECG can be reviewed online, as well as the simulated location of the first responders team. BIOSal is the layer on which DroidJacekt relies on for most of the reception, gathering and processing of the team acquired information. Three detectors are implemented into DroidJacket to support decision making, a QRS detector, a beat detector and an arrhythmia episode detector.

Chapter 3

Tools used for this project

In this chapter I will describe the Software and Hardware tools used for this thesis project. The project tries to link two Hardware components. Software is used to manage connections and display exchanged information. As the software layer is build upon the hardware layer, I will first concentrate on the hardware. I will describe the two Hardware components used and the linking Software with its development programs. In the Hardware section the Equivital EQ01 and Bluetooth are described as well as the HTC Sensation smart phone, for which the application was developed. Regarding Software, these are the Android application anatomy, the integrated development environment Eclipse and the programs to configure and control the Equivital EQ01.

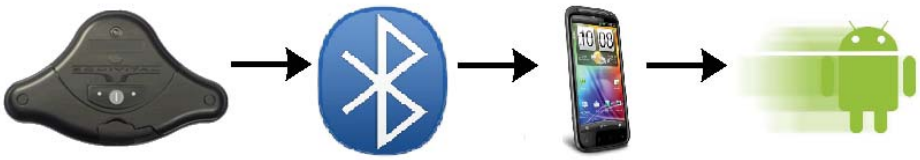

Figure 3.1: From left to right, the incorporated parts in this project are the Equivital EQ01 sensor, a Bluetooth connection, an HTC Sensation smart phone and the Android Software platform.[46][47][48][49]

3.1 Hardware

The Hardware used in this project is comprised of two devices and the connection between them. First of all there is the Equivital EQ01, a multi-parametric sensor. This device collects and transmits vital sign information from the body of the person wearing the device to a smart computer, server or base station. With the help of a Bluetooth connection, the information is send over a network in close proximity to the device. In the following we will take a closer look at the EQ01, its components, application to the body, specifications and the structure of the acquired data. Furthermore the connection to the smart phone via Bluetooth, the Rfcomm socket and the SPP protocol are described in detail. The second device used in this project is the HTC Sensation smart phone. It has the newest version of Android installed and was used to debug and test applications under real life conditions. A closer look at its specifications, benchmarks and consumption will be taken in the end of this section.

3.1.1 Hidalgo Equivital EQ01

The Hidalgo Equivital EQ01 is the predecessor of the EQ02 that was described in greater detail in the chapter "state of the art". Hidalgo launched its first incarnation of a wearable, mobile monitoring system in 2010[98]. The EQ01 is applied to the body with the help of a belt which is strapped around the chest. When the device is attached to the belt, it rests on the breastbone on top of the heart. Inside the belt are sensors that rest directly on the skin to measure vital information with impedance measurements[98]. The Cambridge based company delivered the EQ01 along with a loading station, the belt, Bluetooth adapter, cable and instructions and manuals on CD. Taken from the specification sheet, the EQ01 senses ECG, respiration, motion, temperature and a few other measurements. Very helpful can be the extraction of R-R interval etc to further analyse the signals. Alarms and status information is also sent over Bluetooth, making it ready for bi-directional communication. Battery status and other features are additional ones of a few. Communication is handled using a serial connection over either Bluetooth or a serial cable. For Bluetooth connectivity the RFCOMM protocol along with the SPP profile is used for transmissions between the EQ01 and a

Figure 3.2: Hidalgo Equivital EQ01.[27]

base station or other mobile smart device. Symbol rate is denoted as 38,4 kbps in the specification sheet and a data length of 8 bit with one start and one stop bit. Through the transmitted data, information is formatted in message frames with a fixed size and syntax. Message frames on the application layer contain bytes of information. It can consist of three byte characters, with message type and two characters of data, or four character, same as three characters with added message sub type, depending on the type of the message. By reading the first byte from

Message Type	Data 1	Data 2

Message Type	Msg sub-type	Data 1	Data 2

Figure 3.3: Message frame types[28]

a message, the receiving party knows what kind of data follows and therefore if

it has three or four characters. This ensures that with every message send, the

Data Item	Message Type	Data Prior to Coding
Raw Disclosure Output	-	
Primary ECG1	e	10 bit raw A/D reading + sequence no. (see 0 below)
Secondary ECG2	c	10 bit raw A/D reading + sequence no. (see 0 below)
Belt Sensor Respiration	f	10 bit raw A/D reading
Accelerometer Vertical Axis	v	10 bit raw A/D reading (see below for conversion to milli-g)
Accelerometer Lateral Axis	l	10 bit raw A/D reading
Accelerometer Longitudinal Axis	w	10 bit raw A/D reading
Skin Temperature	t	calibrated value in 0.1 °C, ie 0 to +204.7 °C
Impedance Respiration	i	10 bit raw A/D reading
R-R Interval	r	9 bit value in 1/256 s units i.e. 0.000 to 1.996 sec + sequence number (see 3.4 below)
Summary Disclosure Outputs:	-	
Belt Respiration Rate	u	12 bit value in 0.1 bpm ie 0.0 to 409.5 bpm
Heart Rate	h	12 bit value in 0.1 bpm ie 0.0 to 409.5 bpm
Impedance Respiration Rate	j	12 bit value in 0.1 bpm ie 0.0 to 409.5 bpm
EDR Respiration Rate	d	12 bit value in 0.1 bpm ie 0.0 to 409.5 bpm
Configuration/reporting:	-	
Indication	n	see indications table
Indication Clear	o	see indications table
Report configuration or calibration value	z	value as configured
Sensor ID Part 1	{	ls 12 bits
Sensor ID Part 2	}	ms 12 bits
Battery voltage	x	12 bit value in millivolts
Fault	q	see below
ECG signal quality	a	0-100%
Heart Rate Confidence	g	0-100%
Belt Resp signal quality	k	0-100%
EDR signal quality	m	0-100%
Breathing rate confidence	p	0-100%
Impedance resp signal quality	~	0-100%
VSDS Algorithm Confidence	|	0-100%
Core Temperature	y	temperature value in 0.01°C (see below for format)
Core pill serial number	s	unique serial number (0-4095)
Four-character message	b	See section 3.8 below

Figure 3.4: Message characters[28]

receiver already knows what to expect. Four character messages are only used as outgoing information from the EQ01, whereas three character messages can be either incoming messages from other connected devices to the EQ01 or outgoing messages. The data bytes are encoded, usually unsigned integer values, with a maximum length of 12 bits. Taken from the specification sheet, the encoding is

best explained as followed: “The first transmitted data character is formed by taking the least significant 6 bits of the 12 bits and adding 32 decimal (20 Hex) to avoid non-printing characters. Similarly the second transmitted data character is formed by taking the most significant 6 bits of the 12 bits and adding 32 decimal. This means that all data characters lie in the range 32 to 95 Decimal (20 to 5F Hex). So that message types can be distinguished from data characters, message types are deliberately restricted to be in the range 96 to 126. This means that all lower case letters are available plus a few more characters. This approach allows the receiving party to synchronise by ignoring data characters until it receives a message type.”[28] The additional sub type character in four-character messages ensures that older receivers, that were designed for three character messages only, are compatible by letting the receiver ignore four character messages. Sub type characters are send in place of the first data character in the range of 32 to 95 decimal. Older receivers recognize this as an unknown data message after reading the first character, the message type. It will ignore the following two characters to move to the second data character of the message which also will be ignored, because it is not a valid message type. Afterwards the next messages' first character will be read and if it is a three character message will be a valid message type and so the receiver will recognize the data characters correctly. The rates for the data characters in which new data is generated by the sensors and transmitted over the physical connection varies for the different types of data. Primary and Secondary ECG waveforms are both a size of 6 chars and are generated with a rate of 256Hz, resulting in 1536 chars per second. All chars combined from all items bring forth a rate of 1928 chars per second. Accelerometer and both Respiration measuring techniques are generated at 25.6Hz. Three chars are used for data size resulting in 230 per second for the accelerometer, because of the three axis, and 77 for both Respiration methods. R-R interval and the two ECG waveforms are checked for drop outs with the help of a three, for the R-R interval, and four bit, for the ECG waveforms, sequence number. For the ECG waveforms, the sequence number is split in half and transmitted over the most significant two bits of the two 12bit data messages, one from each channel(primary and secondary). If both data messages have been transmitted successfully, the sequence number will be reset with the 16th transmission, allowing a detection of drop outs of up to 17/256 s or 62.6

Data Item	Size (Chars)	Data Volume (Chars per second)
Raw Disclosure Output:		
Primary and Secondary ECG @ 256Hz	6	1536
3 Axis Accelerometer @25.6Hz	9	230
Belt Respiration @25.6Hz	3	77
Impedance Respiration @ 25.6Hz	3	77
R-R Interval - irregular, say 1/3 Hz	3	1
Disclosures every five seconds:		
Body Position	3	0.6
Motion	3	0.6
Apnea state	3	0.6
Fault status	3	0.6
ECG saturation status	3	0.6
Short term heart rate	3	0.6
Summary Disclosure Outputs every 15 seconds:		
Sensor ID	6	0.4
Battery measurement	3	0.2
Core Pill status	3	0.2
Core Pill serial number	3	0.2
Core Pill temperature	6	0.4
ECG Quality	3	0.2
Belt Quality	3	0.2
EDR Quality	3	0.2
Impedance Resp Quality	3	0.2
Heart Rate	3	0.2
Belt Respiration Rate	3	0.2
Impedance Respiration Rate	3	0.2
EDR Respiration Rate	3	0.2
Heart Rate Quality	3	0.2
Breathing Rate Quality	3	0.2
Life signs detection	3	0.2
Life signs quality	3	0.2
Total		1928

Figure 3.5: Data volumes[28]

ms. Similarly the R-R interval sequence number is sent within the three most significant bits from the 12 bit data message. It will be reset already after only 8 transmission, detecting possible drop outs every 8 seconds if one considers a heart rate of 60bpm. The maximum range in which an R-R interval can be detected is 1.996 seconds, originating from the lower rest 9 bit of the 12 bit message data and the 256 Hz data rate from the ECG, by calculating

$$((2^9) - 1)/256$$

seconds. If the value exceeds the maximum range, which corresponds to a heart rate of 30 bpm, and the heart rate is lower, instead a value of 0 is sent. Readings from the Accelerometer can be converted from the raw A/D values to nominal values in mg (force). For this 628 has to be subtracted from the reading and be multiplied by 5.69[28]. This stems from the sensitivity and zero point of the sensors from the accelerometers in the EQ01. There are two operating modes available for the EQ01. First one is the full disclosure mode which was used for this project and the second one is the partial disclosure mode. In full disclosure mode the device behaves and delivers information as described here. When in partial disclosure mode "raw respiration data, raw ECG data, raw accelerometer, and battery voltage readings are suppressed. RR interval data is also suppressed, unless the RR in partial disclosure calibration parameter is set to 1."[28] Switching from one mode to another, as well as generally configuring the EQ01, can be done using messages in form of message character similar to the receiving message characters. A list of messages characters that can be utilised and its effects is presented in figure 3.4. Logging data onto the SD card can also be handled with the help of messages. If full logging is turned on, regardless of which mode is set it will log all data. When data is logged in partial disclosure mode and full logging is turned off, data transmitted over the serial Bluetooth or wired connection will be logged. Data can be saved for up to 50 hours in disclosure mode with the 8GB memory size, the incorporate SD card has.[99]

3.1.2 Bluetooth

The Swedish company Ericsson invented Bluetooth in 1994[100]. It is a technology that imitates a wired connection over a radio transmission. Later in 1998 the Special Interest Group (SIG) was founded by a group of companies to maintain and administer Bluetooth. The short range communication technology uses profiles and protocols to handle data transmission between a wide variety of devices. Synchronisation for PCs and mobile phones, printing and fax capabilities as well as headsets for voice calls or monitoring in health care systems and many more applications are employable. The networks established can have a range from under a meter to an unlimited range depending on specification. Usually these have

Figure 3.6: Bluetooth[29]

a ubiquity of about 10 meters and are ad hoc networks, also called piconets[101]. A big advantage over other techniques is that the profiles used in Bluetooth are independent from the transmission methods. Therefore it is possible to use wireless LAN for the Bluetooth 3.0 technology for example. Security is an important part about Bluetooth highlighted by the SIG. It is provided by the methods used to handle connections. Other features are the low power consumption and low cost that make it favourable for hardware manufacturers and software developers. This results from the fact that Bluetooth is license free, standardized and the hardware can be build in a very small package. Today it is available in many devices because of its amenity[102]. In the following I would like to go into more detail about the Bluetooth technology, the SPP protocol and the Rfcomm socket.

Piconets which Bluetooth networks are, can contain eight active devices for communication and another 248 that can be connected but only passively, not taking part in the communication. Every Device has its own universally unique identifier (UUID) composed of 128 bits from which 48 bits form the device specific address[103]. It is used to identify devices in a network without a central

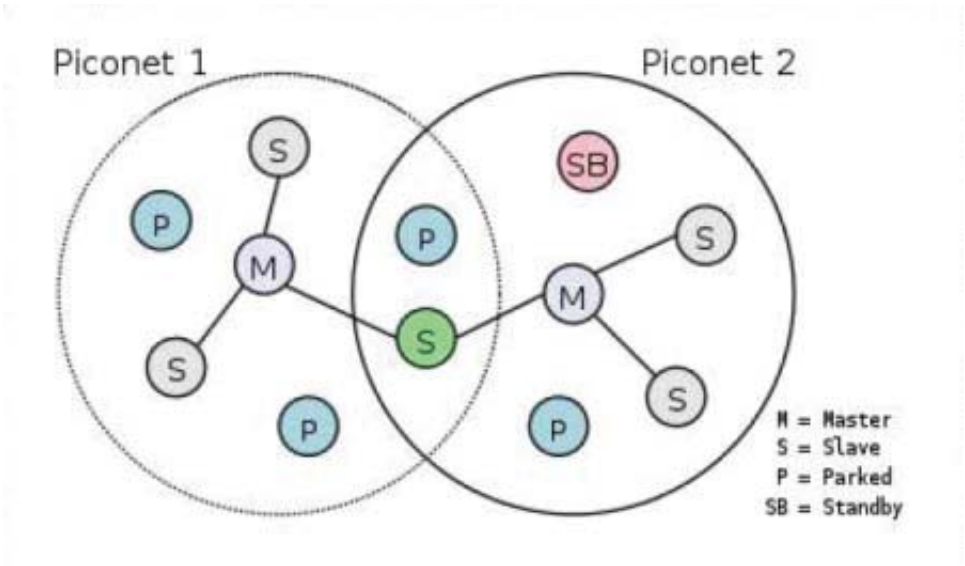

Figure 3.7: Piconet[30]

registration process. The frequency spectrum that is used is the one from the unlicensed Industrial, Scientific, Medical (ISM) band. It is widely available in countries all over the world and lies between 2.402 GHz and 2.480 GHz which is a 79 MHz wide band. The ISM band is used by other communication methods like wireless LAN or in the kitchen by the microwave. In order to maintain a stable and interference free connection, Bluetooth uses Frequency-hopping spread spectrum with 1600 hops per second[101]. Meaning that the connection switches 1600 times a second the frequency in the 79 MHz spectrum of the ISM band by chance. Generally, Bluetooth connections are derived into two links, the synchronous (SCO) and asynchronous connection oriented (ACL) links. The former is used for transmitting speech, the latter for transmitting data. Normally radio standards can only handle one of the two methods, synchronous or asynchronous, which makes Bluetooth special in this regard. A data transmission rate of 1 MBit per second for Bluetooth and up to 3 MBit/s with Enhanced Data Rate (EDR) or the newer Bluetooth 3.0 standard are possible[101]. The net values are a bit lower, 720 kBit/s download and 57 kBit/s upload for an ACL link and 432 kBit/s for both down- and uplink for an SCO link. Up to which ranges a connection is still reached depends on the power a radio transmission is established and the

sensitivity of the receiving antennas. As long as the power ratings are within the bound of the law, Bluetooth can be send with any power rating. Defined classes provide a standardized power rating and network range. Class 3 radios transmit data in the range of up to one meter, while class 2 radios establish a range of 10 meters with a power rating of 2.5 mW. Class 2 radios are mainly used in consumer electronics, whereas class 3 radios are mostly used in the industry, which reach devices in Bluetooth networks up to 100 meters away. A low power specification for Bluetooth, specifically developed for devices requiring low data rates but long batter lifespan, consumes as little as 1/100 of the power of standard Bluetooth technology. The range decreases significantly through walls or ceilings, even though visual contact between devices is not required. As described in the Introduction of this section, Security plays a big role in Bluetooth connections. Three safety modes are implemented beginning with the safest, these are the link level enforced security, the service level enforced security and the non-secure mode. Highest security is guaranteed with the link level enforced security, where devices initiate security procedures like authentication before the connection is established. The service level enforced security mode establishes a connection before the security procedures. In the non-secure mode no security will be initiated, just like the name suggests. Taking a closer look at the architecture reveals the Bluetooth Protocol Stack. The protocol stack is a package containing software and drivers that enable connections with Bluetooth devices and services to use the Bluetooth profiles.[100, 102] Profiles are used to specify communication between devices and define composition and content depending on application. They bear on the application layer, which lies on top of the Bluetooth core specification and the protocols. An application does not have to surpass every layer, but is able to do so with many of one vertical slice. The protocols are arranged in the following order. On the bottom of the stack resides the Bluetooth radio, on which the Baseband is put on. Above that the Link Manager Protocol (LMP) for managing authentication and connection establishment and termination is rested. With the Host Controller Interface (HCI) these four layers make up the physical connection of the radio communication embodying the hardware abstraction Layer and the Data Link Layer. Overlying are the Logical Link and Control Adaptation Protocol (L2CAP) that procures logical links between devices and multiplexes

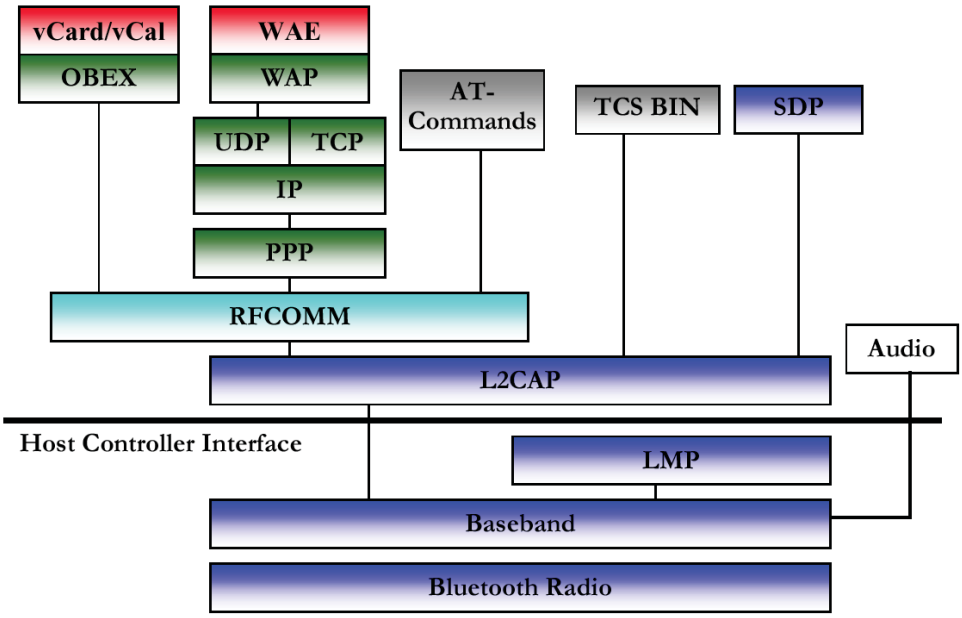

Figure 3.8: Bluetooth Protocol stack[31]

multiple connections. Based on L2CAP are a variety of other protocols except for Audio which is directly connected to the Baseband. Some of these protocols are the Radio Frequency Communications (RFCOMM), Object Exchange (OBEX), Telephony Control Protocol Binary (TCS BIN) and the Service Discovery Protocol (SDP). The SDP facilitates identification of a service offered by other devices in the network with the help of the UUID. TCS BIN is a control layer for telecommunication applications. An adopted protocol is OBEX, which allows for exchange of objects and data. RFCOMM is a cable replacement protocol. It emulates a RS-232 serial port with up to sixty simultaneous connections to a device at the same time with the help of L2CAP. The protocol was established by the SIG but is based on the ETSI standard TS 07.10. Especially for its downwardly compatibleness and its data transfer capabilities for Bluetooth profiles, RFCOMM is used. A connection is established by first setting up the underlying L2CAP connection. After RFCOMM control frames are exchanged between devices, transmission of data over following channels. The byte streams of a Message are send over the Baseband without any ability to correct errors in the process. The streams contain messages. Basically the frames for RFCOMM

are the same as for the GSM 07.10 standard, except that they do not have a start and stop flag. This is because the data is stored into L2CAP packets of fixed size and no partition is needed. The size of data is limited to the size of the L2CAP packets by the Maximum Transmission Unit (MTU). As described earlier, Bluetooth profiles embody defined rules and specified behaviours for a communication between devices. Profiles are defined and standardized by the SIG and are put on top of the Protocol stack. They describe applications and allow the devices to only use features used for these. The consequences are that a device can be specifically designed for one or a few profiles, needing little components and making it very small in size and energy consumption. More than one profile can be used at the same time. In this project the sensor and the smart phone used the SPP profile to handle communication. It is based on the RFCOMM Protocol and is intended for devices with data transmission over an emulated RS-232 serial cable. SPP can establish only a one-slot packet distribution over the underlying protocols with data rates of 128 kbps or more if specified. Fortunately Bluetooth provides the functionality to simultaneously use multiple profiles. In this case multiple SPP profiles are run at the same time to ensure higher data rates. Security features are not required but features such as authorization, authentication and encryption can be added. The path of the communication through the protocol stack goes from below to above. From the Baseband over the LMP and L2CAP, to RFCOMM and SDP. A communication process is established in three main steps. First of all a link to an emulated serial port in another device in the network is established and a virtual serial connection is set up afterwards. This includes the use of the SDP protocol to find the RFCOMM Server channel and an RFCOMM session on a requested L2CAP channel. Once this is done a data link connection on the RFCOMM session is established. The second step follows with accepting the data link and establishing the virtual serial connection. For this, all previously made establishments of L2CAP, RFCOMM session and data link have to be accepted by the remote device. The last and third step is to register the service record in the local SDP database. Now the Connection between two devices with the help of the SPP profile is made. Currently available is Bluetooth in Version 4.0 which was adopted by the SIG in June 30, 2010. Version 4.0 evolved from Nokia's Wibree, which was intended to be an addition to the

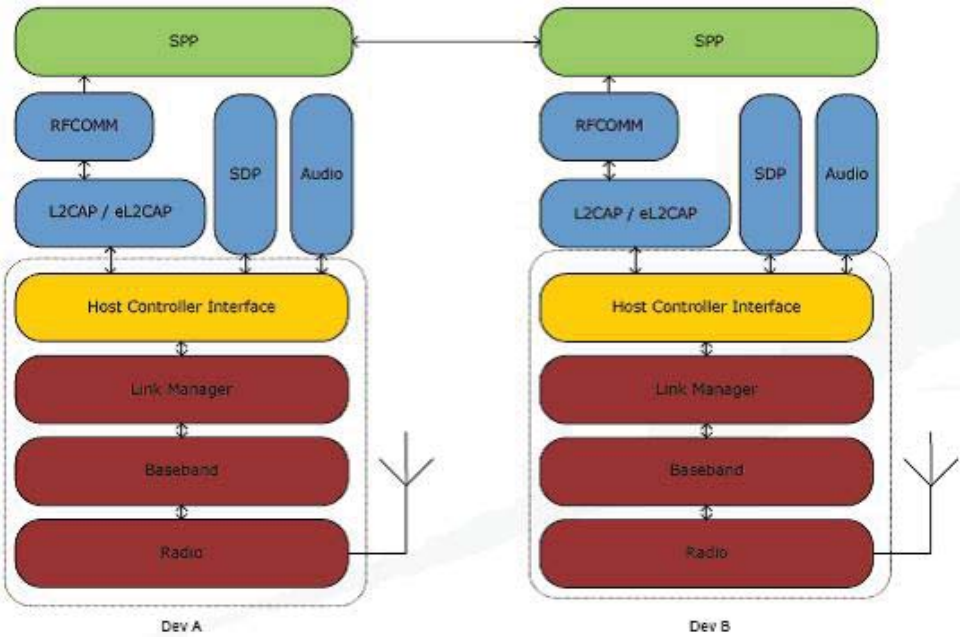

Figure 3.9: Bluetooth communication path[32]

existing Bluetooth technology, but instead running on very low power. Wibree specifications hava been adopted into the Bluetooth low energy (BLE) subset, that is part of version 4.0. The BLE has become a trimmed down version of the original bluetooth with simplified protocols and specifically designed for small low energy chips. The previous Bluetooth specification was introduced in April 2009 under the name Bluetooth v3.0+HS. HS connoted HighSpeed mode which can be switched on if needed. The Base for version 3.0 is a 3 Mbit/s connection for exchanging big amounts of data like multimedia data, which for previous versions were too large. With the HighSpeed option theoretically up to 24Mbit/s connections are possible. An ad-hoc network over the WLAN standard IEEE 802.11g is established for this purpose. WLAN works within the same frequency spectrum, but Bluetooth does not have to surpass access points like computer clients.[101, 104, 105]

Figure 3.10: HTC Sensation[33]

3.1.3 HTC Sensation

The Sensation is a touch-screen smart phone manufactured by HTC. It was launched in the second quarter of 2011 and comes with Android OS already installed[106]. There is active Support for this Smart phone from HTC, making it possible to install the newest Android version. Particularly striking to the eye is the 4.3 inch capacitive touch-screen which nearly takes up the whole front. The screen has a resolution of 16M colours and 540 by 960 pixel. For fast processing a dual core CPU with a clocking speed of 1.2 GHz is built in. As described in the chapter state of the art android is capable of running on devices featuring a dual-core CPU. Connectivity options range from Bluetooth, over Internet, to

telephone networks. Bluetooth in Version 3 is available with support for A2DP for wireless stereo headsets, FTP and OPP (object push) for file transfer and SPP. Other supported Profiles can be extracted from the table. Internet connections can be set up using 3G, Wi-Fi, EDGE or GPRS. Networks for phone calls are either HSPA/WCDMA or Quad-band GSM/GPRS/EDGE with varying bands, depending on location. The lifespan of the 1520 mAh battery depends greatly on the processes and connections that are active. If the smart phone is used for telecommunication services, the battery lifespan during talk time with WCDMA is estimated to be 420 minutes and 500 minutes with GSM. When in standby the time increases to 525 hours and 285 hours respectively. The consumption of battery power increases with the use of an active internet or Bluetooth connection. An internal SD card with a memory of 1 GB and a ram of 768 MB are built in for short or long term storing of data. A number of sensors are embedded into the hardware, making it possible for applications to make use of them. Gyro sensor, G-Sensor, Digital compass, Proximity sensor, Ambient light sensor and an Internal GPS antenna make the environment accessible to the smart phone. The four buttons below the screen can be used to navigate back to the main menu, choose settings, go back and use the search function. To synchronize data and install third party applications on the Sensation, the Software HTC Sync can be downloaded and installed on a desktop computer.

Figure 3.11: HTC Sensation specifications[34]

Using the HTC Sensation for debugging and testing purposes with Eclipse or a computer, HTC Sync is necessary. After installing it, the Smart phone can be connected to the Computer for tethering via Wi-Fi or a USB cable. Some of the problems that we encountered during development processes were related to Bluetooth connections. In order to connect to Bluetooth devices, regardless of other smart phones, computers or the Equivital, firstly the devices had to be paired with the Sensation. After several connections and disconnections with other devices, an application under the name Bluetooth share in the list of applications had to be cleared. Clearing this application's memory, solved disabilities to connect with Bluetooth.[106]

3.2 Software

The Software used in this project can be assigned to the two Hardware components that make up this project. On the one hand we have the Software related to the development on the smart phone. This is the Android platform and the development environment Eclipse used to program applications for Android. As described in the chapter state of the art, Android has a number of benefits over other available mobile OS that lead to the use in this project. We will take a closer look into what components make developing for android possible, like the software development kit (SDK), the android virtual device (AVD) and the Eclipse integrated development environment (IDE). Moreover the basic application architecture and structure will be presented, to give an overview about how applications are executed. On the other hand we have the Software that is related to setting up and configuring the Equivital EQ02 mobile health care sensor. For configuring the EQ01 on a desktop computer, the SEM customization utility has been applied to this project and is described in more detail. Furthermore the LiveLink Standard Edition is the base Software for the EQ01 that displays the transmitted data and was used for validation and testing purposes.

3.2.1 Android Application anatomy

Applications created for the android platform can generally be divided into distinct use cases[76]. Foreground applications are used when permanent user interaction is requested or important information is constantly updated on the screen. On the contrary, apps are developed to stay in the background, when a service or action should be carried out, while no user interaction is necessary or when these functions should provide their capabilities to other apps. A combination of the two use cases could also be possible, with an app set to work in an intermittent mode, where only little foreground interaction is done but a lot of processes are carried out in the background at the same time. An example could be this project, where the user can see the real time data on the screen, while in the background Bluetooth connection and data processing are carried out. Another option is to design applications to appear on the home screen as a widget or as a Live wallpaper[78]. This means that with a short graphical component, the user can interact with functions of the app from the home screen, for example message notifications or cpu usage. The catchword cpu usage leads us to another aspect that has to be considered when developing for android, the hardware limitations and varieties between devices. Since Android is capable of running on a wide span of devices applications have to be able to work on them all with varying capabilities regarding processing power, screen resolution and battery sizes. These factors, that make for a challenging aspect of developing an application to look and feel the same on every device, have to be well planned. Unfortunately, especially power consumption is an issue when transferring data over the wireless connections to and from the smart phone[78]. Though, Bluetooth is the only wireless connection which is accessible without any additional devices or services like Wi-Fi etc. That's why it was chosen to be the connection between a sensor and the smart phone. It provides enough bandwidth capacities for the sensor data and has protocols incorporated which service the connection. The provided APIs for establishing a Bluetooth connection are wrappers around RFCOMM. For more information about this protocol and Bluetooth in general, please refer to the section Bluetooth. To take all previously mentioned considerations into account, it is of vital information to know how an applications' anatomy looks

like. Consequently we will take a closer look at the lifecycle of activities, processing, the components and resources. Before we proceed to this part, a look at how applications are handled by the android OS is of vital information. Every application represents a user in the Linux system, android is based on and has a unique user ID, which is only accessible for the system. A single application has its own instance of the Dalvik virtual machine (DVM), which is a sandbox[107]. This provides high security as only the code runs in this Sandbox, isolated from other applications. Lets focus on the parts an application actually consists of. An application can consist of one or more main components out of a group of core elements. This group is composed of activities, services, content provider and broadcast receiver. An activity represents a single screen or view in Android. Its main purposes is to manage menu items, text elements or buttons. Usually an application consists of a single activity to reduce complexity and implements other core elements to provide functions and features. Multiple activities can help making very feature rich applications or very clearly arrange the navigation and look. To communicate between activities or to start activities, services and broadcast receivers, an asynchronous message on the application layer called intent is used[76]. Intends can be addressed to a specific receiver inside one or throughout multiple applications and be explicit or be implicit and addressed to no particular receiver. The receiver of an implicit intent is identified from the content or context. For communication between multiple instances in an application or across processes, broadcast receivers are used. Essentially, these are system sent intents or messages on the application framework layer using the method sendBroadcast. Android itself uses broadcast receivers to inform applications of events like connection statuses or use of resources(battery consumption). Applications implement broadcast receiver to listen and respond to broadcast intents sent by the system. Likewise activities can perform broadcast intents throughout an application. Identification of intents and its corresponding components is provided by the intent filters defined in the manifest file. Every application has one manifest file written in .xml code, where all components are declared. Intent filters can be applied to only accept certain intents for components. If a component is not declared it will not exist or be visible to the system and cannot be used. It is assigned to an application by the unique user ID given to the application by

the system. The manifest file does more than only declaring components, it also can set requirements, declare features and apply permissions. Such as enabling permissions for the application to read from an SD card, enable Bluetooth or internet connection and set hardware requirements like a touch screen. Setting minimum versions of Android OS as requirements for applications, as some parts are only compatible with newer versions. The requirements set in the manifest file are not checked for compatibility by the system itself but by Software like google play before the user downloads or installs an application on a device. Switching between views and the corresponding activities is handled with the help of the so called back stack and various callback methods[108]. The back stack manages activities, when they have to appear or reappear and be destroyed, in a last in first out (LIFO) behaviour. The operations, whether to push or pop an activity from the stack, are carried out upon the three states an activity is in. If an activity is in the foreground and is in interaction with the user, it remains in the state "resumed". Once another activity moves into the foreground or an activity is only partially visible, it switches to the ?paused? state. Paused activities remain alive, meaning their object is stored in memory to maintain its informations, however, if the system requires more memory they are deleted, respectively killed. If an activity has been killed by the system, it has to be created all over again if it is called to reappear. Lastly activities can enter the state "stopped", being in the background still alive but not visible. This makes it prone to being killed by the system desiring memory[76]. Generally speaking, the chance of a process to be killed is determined by its priority. Processes that are active have the highest priority, while phantom processes, empty processes used to start activities faster, and background processes have lowest priority. In the middle lie the running processes, which are more likely to be killed after the ones with lowest priority. Switching between states is notified by the lifecycle callback methods that android activities implement and which are part of the activity lifecycle. The methods that an activity receives, allow the user to perform actions before or after a state has been changed. Between onCreate() and onDestroy() the activty is alive and experiences its entire lifetime. The milestones are used to initiate and release general, fundamental parts, like resources and layouts. From the method onStart() to onStop() the activity is visible and experiences user interactions.

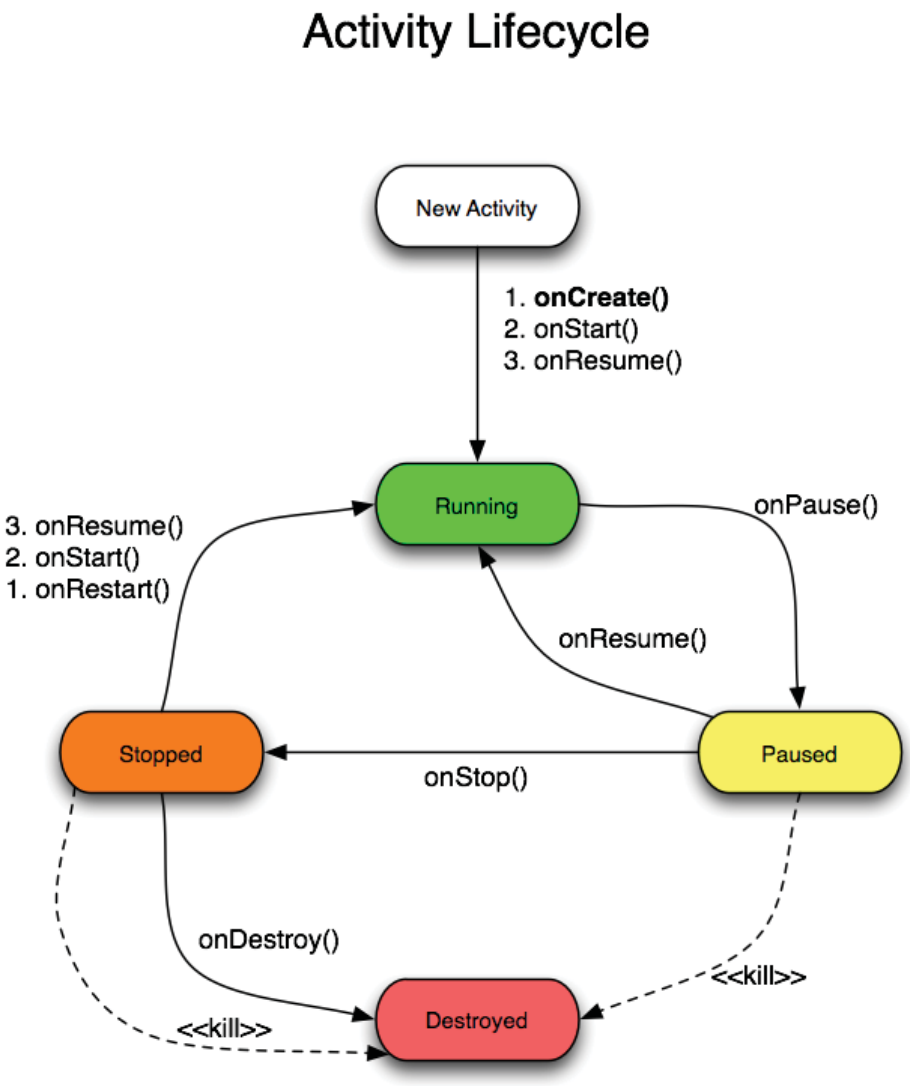

Figure 3.12: Activity back stack[35]

This period can be called many times as well as the foreground lifetime. The appendent methods are onResume() and onPause() in this very order of appearance. The lifecycle of activities is always the same as defined by the Android OS, independent of the structure of the components of an application. Complexity and structure on the other hand are determined by the purpose, performance and desired look and feel of an application. For having a background task running

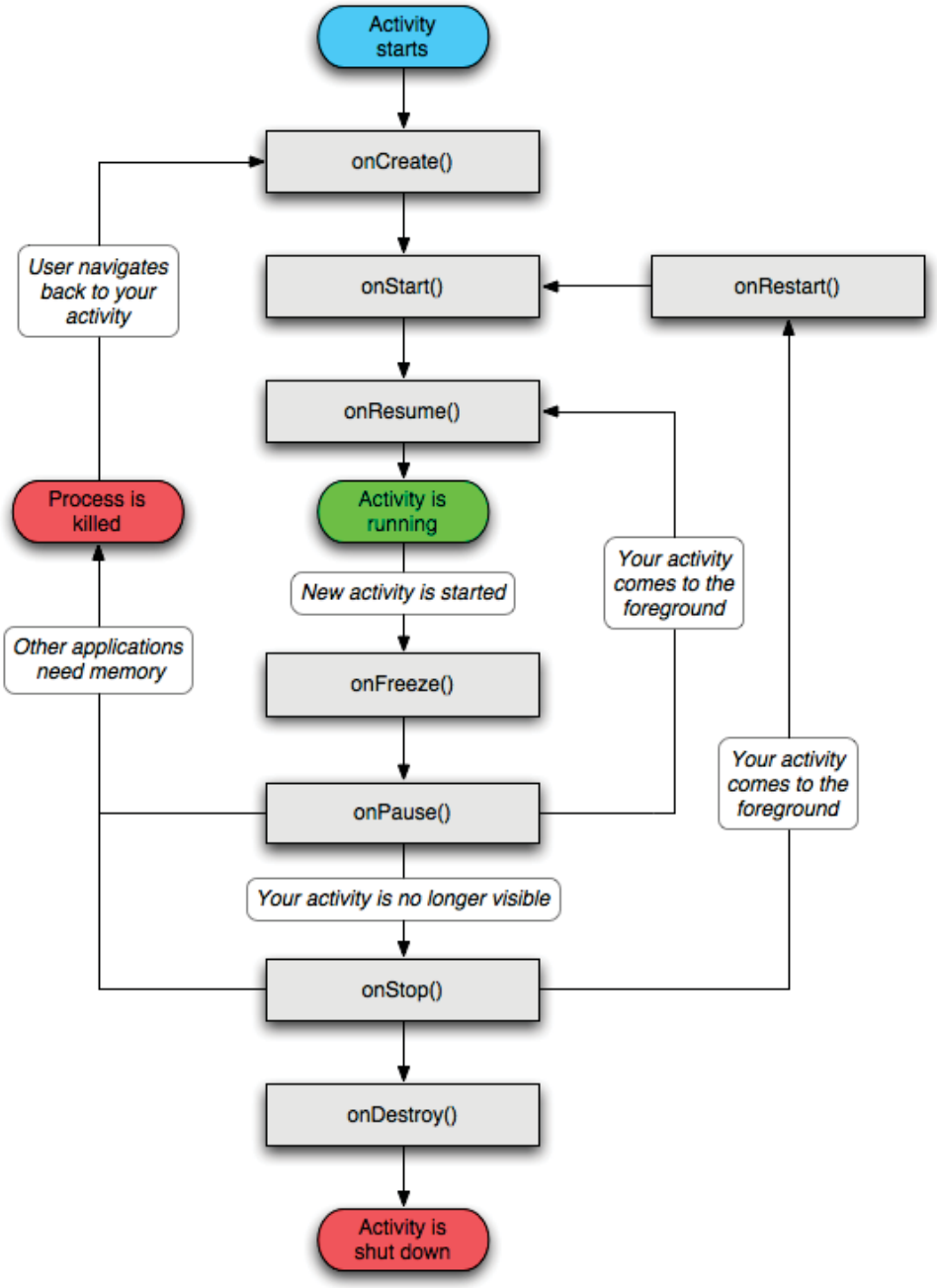

Figure 3.13: Activity lifecycle[36]

across the whole OS (outside and throughout apps), which purpose is for example to establish an internet connection or provide email services, a service is used.

Services have no interaction with the user over the user interface and can run on the same or different process to execute a task. They are either called local service if they run on the same process as the application that started it or remote service when run on their own independent process. To start a service the method startService() is called by a component in an application[78]. It will continue to run independently, meaning that when the component that started the service is destroyed, the service will still run. If the operation it is performing is finished, the service itself has to call stopSelf(). Another possibility to stop a service is to call the method stopService() from another component, which will afterwards destroy the service by the system. Interprocess communication (IPC), or simplified the use of a service by multiple applications, can be carried out by binding an applications' component to a service. This can provide network transactions, music playback or file management, for example. To bind to a service it has to be created beforehand by an application component. When it is created, the call of the method bindService() by a client component will bind the component to the service and allow for a communication similar to a server- client interaction. BindService() is also capable of creating a service, therefore startService() might not be needed. Bound services run until all components throughout the applications are unbound from it by calling unbindService(). Communication is carried out using an Ibinder interface. Restricting the use of a service to only one application can be defined inside the manifest file by declaring it as private. As already addressed the lifecycle of a service starts with onCreate() and ends with onDestroy() returns. The methods onStartCommand() or onBind() will initiate the start of the active lifetime, meaning an activity is bound to the service. Once the active lifetime has started, the service is declared to run in the foreground and the bound activity has user focus, it is highly unlikely for the service to be force stopped by the system, requiring memory. Likewise, when the service runs in the background but remains the other attributes, being force stopped is very unlikely. The active lifetime will end with the return of the method onUnbind(). Once leaving the active lifetime, a service is prone to being killed by the system. As soon as more memory and resources become available the service will be restarted by the system. The last of the four core components presented here, is the content provider. A content provider is an interface between the instances of

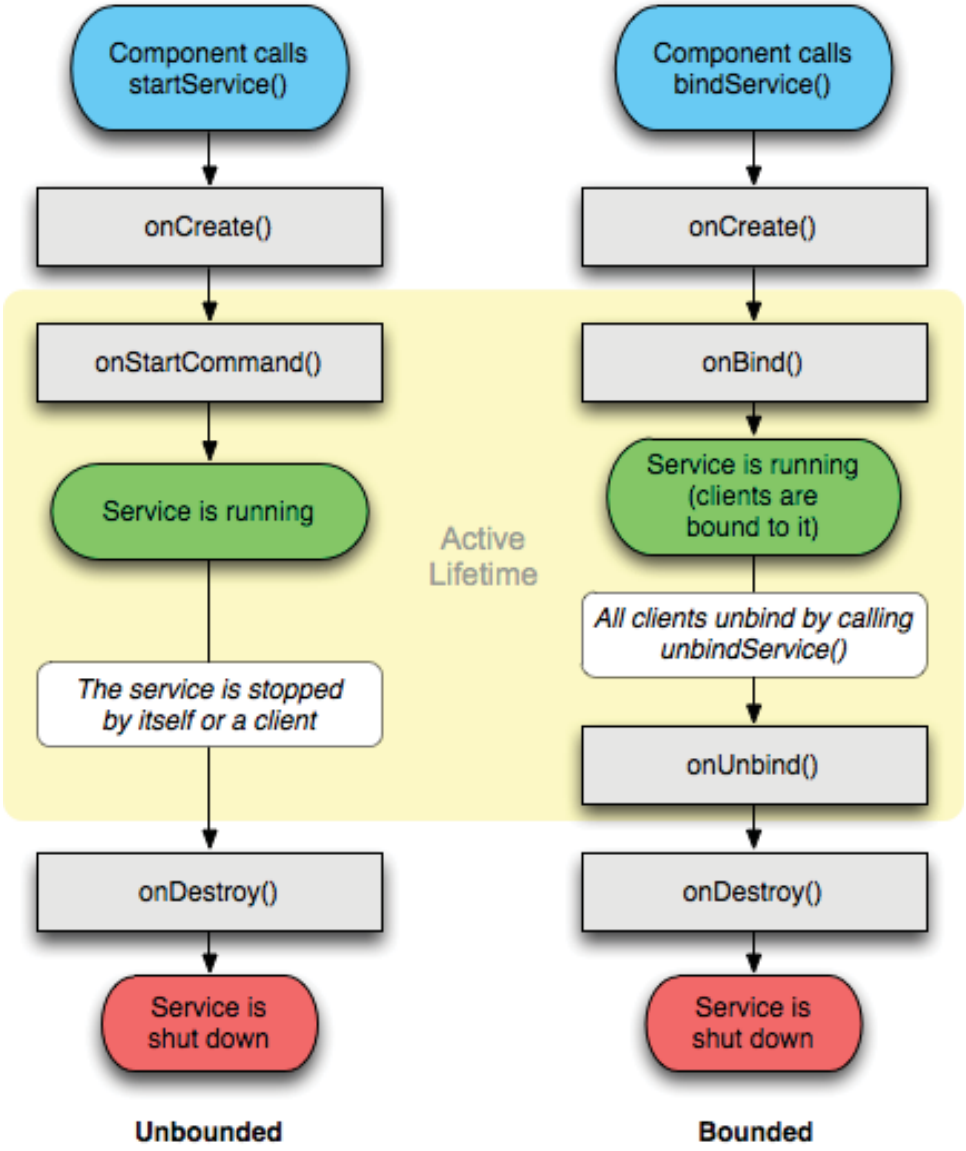

Figure 3.14: Service lifecycle[37]

the DVM or Sandbox, every application has[76]. With the help of this interface, content, as the name already suggests, can be reached throughout applications. Content providers are run on a different process than the code, to provide data security and encapsulate the data. Similarly to broadcast receivers, Android itself includes content providers to manage data such as contact information or

multimedia. For an application to be able to access the content it has to have the required permissions enabled inside the manifest file. Data is offered between applications by the content provider in either file format or a structure, table like, format. Multimedia data is generally stored in files into the application's private space for the content provider to offer a handle to the file. For other data types a database, in SQLite format for example, or similar structure is used to store the data in the form of a table. The rows in this table represents the type of data, whereas the columns represents the actual data peace. The stored data as seen from the system is kept in its initial format, whereas the data seen from an external point of view is stored into a table. This means the ContentProvider, responsible for these actions, acts as an abstraction layer between these two appearances. Content resolver offer basic crud operations (create, retrieve, update and delete) onto the database or storage system. For these operations an instance of the class ContentProvider processes requests from applications and returns the results. Access and modification of the data is done with the ContentResolver object which communicates with the ContentProvider. A customization of the provider allows the user to implement copy and paste operations of complex data from one application to another or custom search functionalities[109]. Instead of accessing data inside an application with the help of content providers, the use of preferences can be made to access a small amount of primitive data stored as key value pairs. Mostly configuration data or user informations are stored here and can be exported or imported from .xml files. The actual data which is stored in a back-end, whose format depends on the implementation, is associated with two hierarchy trees. One is for system preferences, whereas the other is for user specific preferences. Configuration data inside the trees can be accessed with the help of an instance of the preferences class. Each instance represents one node in a tree, with a name and a patch to its ancestors with the same conventions as directories in a file system. To make sure methods invoked with operations onto the data do not exit with exceptions and errors, the developer has to make sure to provide default values in case the back end is not reachable. Along data for exchange and preferences, data that is used as resources for applications to build a user interface with images, buttons, text or the like are incorporated in the resource files. These files and the core components, define the user interface and

behaviour of an application. From layouts and animations to menus and colours, all visible user interactive parts of an application are discarded files inside the resources folder. XML files administer the resources, which makes modifying of visible components possible without alternating the code. By addressing every resource with a unique integer ID, the SDK tools provide instances that can be referenced from the code or XML files. This feature provides the possibility to prepare an application for various hardware situations with different screen sizes for example. Multiple definitions of the same resource in different formats in the xml files enable the use of these resources in different environments. A picture, for example, can exist in three distinct resolutions, one uniquely in the hdpi, mdpi and ldpi folder, but with the same file name. This way it can be referenced by code or an XML file with the same name, respectively ID, but various resolutions. The qualifiers hdpi, mdpi an ldpi refer to the high, mid and low density of a screen. A drawable resource, for example a background image called background, exists in the three density folders. It is addressed by its directory, for example res/drawable/hdpi or res/drawable/mdpi and file name R.drawable.background. R refers to the R class that contains all resource Ids from all resources folders. The R class is automatically created by the system and manages all resources. To do this all resources are compiled into binary data during runtime. A software development kit (SDK) is used to create applications. All previously mentioned components and resources which can form an application are incorporated into a single executable file. These files and the corresponding code are compiled by the android SDK tools into a file with the suffix .apk. This file can be executed by an android OS only, after being installed onto a device. Android provides an SDK Manager that is able to download the SDK tools, which are distributed in package form. With the help of the SDK Manager it is also possible to download example projects, framework SDK libraries, documents, plug-ins and add-ons. Beside these tools it incorporates service programs for testing and debugging applications. Some of these are the Dalvik Debug Monitoring Service (DDMS) or an Android Emulator which requires an Android virtual device, which is described in the next section. The SDK is supported for desktop operating systems such as Windows XP or later, Mac OS X 10.5.8 or later and Linux. To start developing for android the SDK along with a JDK is necessary. The Android virtual

device is needed to debug or run applications on an Emulator, which is a handy tool to test code for different devices virtually. AVD is an implementation of the DVM, so it is decoupled from the computers' hardware, because the DVM acts as a Sandbox. This allows for simulation of all kinds of smart phones or physical devices on the desktop. Some emulators are integrated in the ADT and can be used with Eclipse or the console. For a faster access through eclipse, the emula-

Figure 3.15: Emulator[38]

tor can be started from a snapshot. Their functionality is limited but they offer many features that are available for their physical counterparts. Specification settings can be adjusted or set to the default of the skins, existing corresponding devices, with the help of the so called name value pairs. The AVD manager lets the user administer and choose from a list of devices that hold an instance of the emulator. For testing purposes internet connectivity, SMS and telephony services are available. Additional features that can be adjusted for the desired applications are the screen pixel density, Maximum VM heap size, SD card support, as well as Accelerometer, GPS, and proximity sensor support, just to name a few.

Unfortunately, emulators do not provide Bluetooth support, as the emulation of a virtual serial connection is not supported. Therefore the smart phone played a significant role during testing and debugging stages[76].

3.2.2 Integrated Development Environment

Integrated Development Environments (IDE) are used to develop software on a desktop computer[110]. For this purpose usually a normal text editor is enough, but an IDE provides a few advantages and necessities. The most basic application it allows for is writing and editing code, browsing class structures or providing help during programming. Knowledge of the code provides ability the user in the form of highlighted functions or keywords to make the code more clear to the reader. It can also detect errors in the code and suggest corrections, add missing code automatically or offer macros and abbreviations for a more user friendly and fast working experience. Generally speaking, IDE's provide and manage the necessary resources for programming languages. Files like libraries are kept in the right folder or missing packages are pointed out. Most of the time a compiler, tracing and debugging utilities as well as running code tools are implemented , necessary to convert code into a working program. In this project the Eclipse IDE was used to develop for Android. It was chosen because it is the recommended IDE by the developers at google. In the following we will take a closer look at it[110].

3.2.2.1 Elipse

For the development of applications in Java that are devoted to the android platform the use of Eclipse IDE is recommended by google[76]. As described before, an IDE is not necessary, as programming can be done with the help of a text editor and the SDK, to compile, test and debug an application, but the Android developer tools (ADT) plug in provided for the Eclipse IDE offers many advantages. But before we take a closer look at the android specific features Eclipse is examined. The beginnings of Eclipse reach back to 1998 when IBM started to develop it with the idea in mind to create a program that can work with existing and upcoming tools for software development. In 2001 the code

Figure 3.16: Eclipse[39]

was released open source to convince third party developers to work with eclipse, as scepticism towards IBM came up before, who controlled the development of Eclipse as owners. As many people started to improve and work with the open platform, IBM turned Eclipse over to the non profit eclipse foundation to maintain and manage it[111]. Throughout the years, the eclipse foundation turned out to become a community for organizations and individuals contributing to its development. Today Eclipse resembles to be the most popular IDE on the market for Java development, having gathered 65 per cent market share with over a million downloads. The community has grown to a 1000 committers, who are allowed to alter code directly and review changes made by the 10000 contributors before releasing code. It is estimated, that more than a few million user exist, who apply eclipse in their Software development. Reasons for the popularity are the open source code and cooperation of many companies and developers in the eclipse foundation, as well as the expandability and modularity of the program itself. A variety of programming languages and IDEs are available for Eclipse, like c/c++, cobol, html, php, python and Java, in which it was developed itself. Among developer tools for software applications, it also offers business applications or

is implemented into devices like SkiData card readers. In the background of all processes of Eclipse works Equinox, a system runtime that was based on Open Services Gateway initiative (OSGi), an independent industrial standard. On top of the runtime is the architecture of eclipse build up in the form of plug-ins that are structured as bundles of code. The equinox framework, which is the core of eclipse and base for all plug-ins that allows for interoperability between components, was introduced with version 3.0. Eclipse is readily available for Windows, Mac and Linux with support for plenty of languages. Provided with

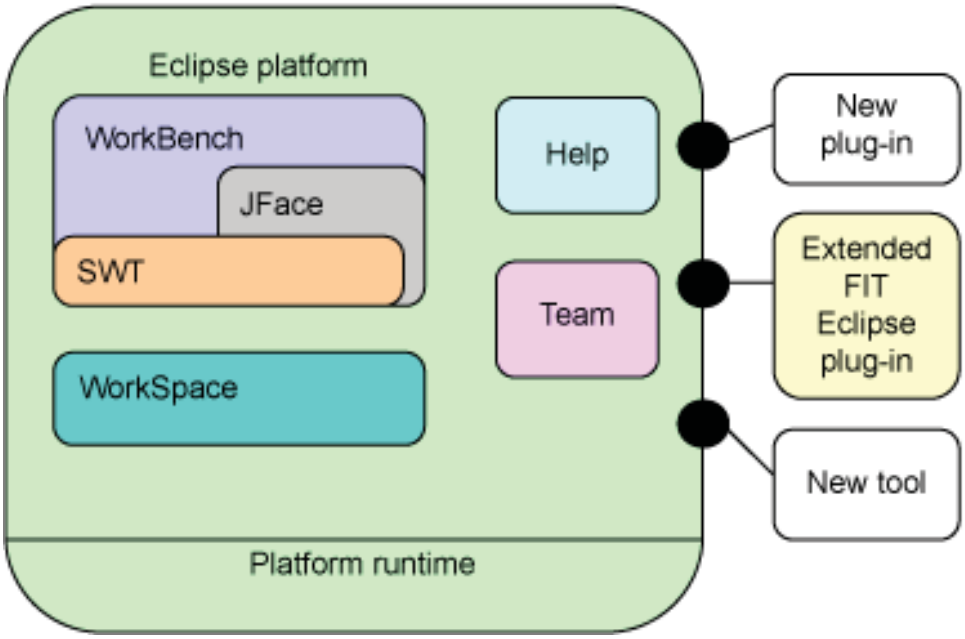

Figure 3.17: Eclipse Architecture[40]

an installation are a code editor, compiler, debugger, gui builder and other tools to make development more affable. In addition, the android developer tools (ADT) plug-in is available that offers a variety of features for the development of android applications. It requires Eclipse 3.5 or above to work and includes he developer tools, along with the Emulator and .class-to-.dex converter. The ADT plug-in integrates the Android virtual device AVD manager, to administer virtual devices, Runtime debugging, to set breakpoints inside the code, and the android emulator. Android emulators can be very useful when debugging and testing

applications, as you can set up basically any smart phone virtually on the desktop computer, with the same functionalities. Unfortunately Bluetooth usage is not possible with virtual devices[111]. Other implementations are an android project wizard, to simplify creation of a new project, editors to work with XML resources for the manifest file, layout and and other resources and automated building of projects, including conversion and installation on virtual devices or smart phones for testing. The Dalvik Debug Monitoring Service (DDMS) provides features like thread viewing or process details, and log or console outputs of Android and Dalvik exist as well. With the help of the log output, runtime errors or exceptions that occur during debugging can be further analysed. Once set up with all its plug-ins and add-ons, the development of an application can begin.[112]

3.2.3 LiveLink Standard Edition

The LiveLink Standard Edition (SE) is alongside the Live Link Professional Edition (PE) one of two Software applications that are part of the Equivital SDK by Hidalgo[41]. LiveLink SE is the off the shelf application delivered with the Equivital EQ01, whereas the LiveLink PE is optional. The Professional edition offers more advanced options for monitoring the Equivital, whereas the Standard edition presents basic functionalities regarding presentation and communication of data. The Standard edition was used in this project for testing functionalities and analysing settings. It is developed for Windows XP or Vista and requires the desktop computer to have at least 1.99 GB of ram and 1.99 Ghz CPU speed. Parts of the LiveLink SE system is the application itself and the so called monitoring system, containing the Equivital EQ01 along with the belt and optional sensors, to transmit data over a Bluetooth connection to the computer. Four optional sensors are compatible with the SE and can be connected wireless or wired to the EQ01. An O2 Saturation Probe, a Core Temperature Capsule, a Galvanic Skin response (GSR) sensor and a dermal Temperature Patch are available. Before using the SE, it has to be installed with all its required additional software, that are Microsoft .NET Framework 3.5 and VEE Run-Time. The Bluetooth interface in the computer needs to be at least Class 1 or else performance issues or limitations might occur. After installation of all components and the SE, an Equivital

or several have to be paired with the computer before starting the program. Once launched the user can choose between two sessions of monitoring, either replay with data from a file or Live in real time with data from the device. After se-

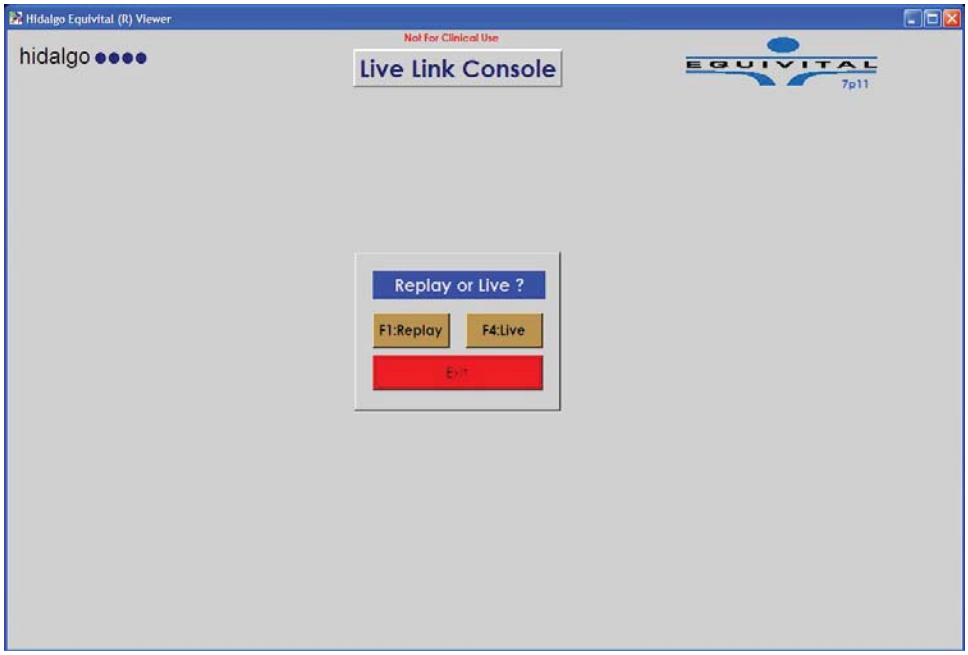

Figure 3.18: LiveLink start screen[41]

lecting the Live option, the user is directed to the Vee LiveLink console in which several device, connection and session related options can be configured. Among them are adding and deleting devices from a permanent list, connection status in three stages, connected and with or without subject id as well as not connected, end session and logging. When a session is ended, all data is archived in a text file which can be found under My Documents\Equivital\LiveLink SE\Discharge. Inside the LiveLink SE folder, files from the logging process are saved which can be selected under the option Replay in the main menu for reexamination. Following the configuration, the summary screen is reached. Here basic information gathered by the EQ01 on the body are presented with the option to go to the Waveform screen. The information is presented in a row for every subject connected with a sensor. From left to right the readings are the subjects' name, Heart rate, Respiration rate, SpO2, Skin and Core temperature, Motion, Posture, Fall

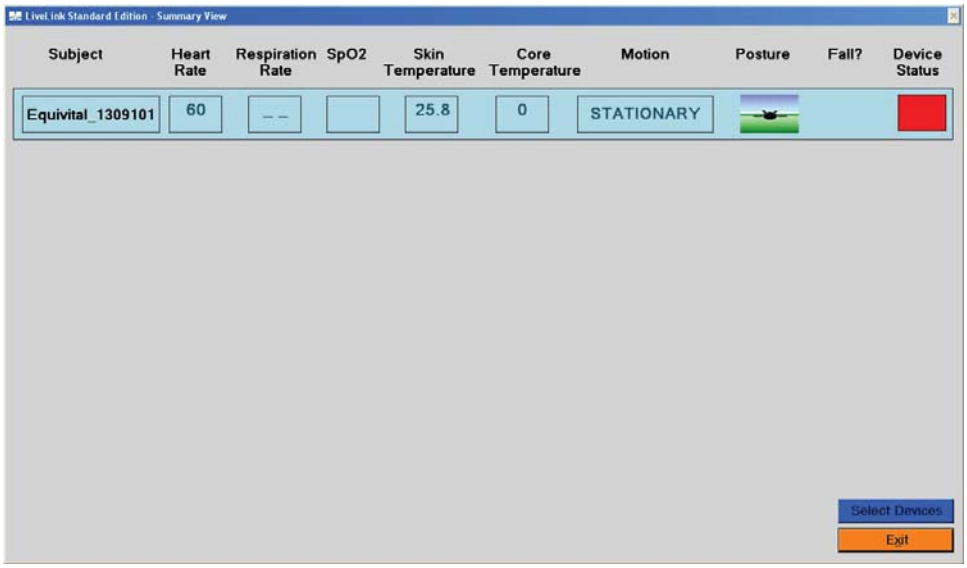

Figure 3.19: LiveLink summary screen[41]

detection and the device status. By clicking on a subject the waveform screen as specified earlier is approached for the corresponding sensor. Physiological data among other useful status notifications about the EQ01, either from a recorded replay or live session, are illustrated. The screen is made up of a variety of panels that contain different informations. If a value inside the panel reaches a critical high or low or is in a specific state, the border of the panel changes it colour. In the top right corner of the screen the heart rate panel is positioned. Among the heart rate in beats per minute, confidence and signal quality level in a range from 0 (the worst) to 100 (the best) are implemented. Signal quality refers to the physical connection of the heart rate recording, whereas confidence level is calculated from signal quality, noise pulses and large scale variations in the interbeat interval. Below Heart rate panel, the SpO2 panel can be found on the right hand side. Two values are presented, one is the oxygen saturation (O2) level from 0 to 100 per cent and the derived pulse from the SpO2. Similarly to the heart rate panel, the respiration rate panel which is third from above on the right hand side, shows signal quality and confidence level. The breathing rate in beats per minute can be chosen from the three available measurement methods, which are ECG derived, from the chest impedance or the chest belt. At last the temperature panel

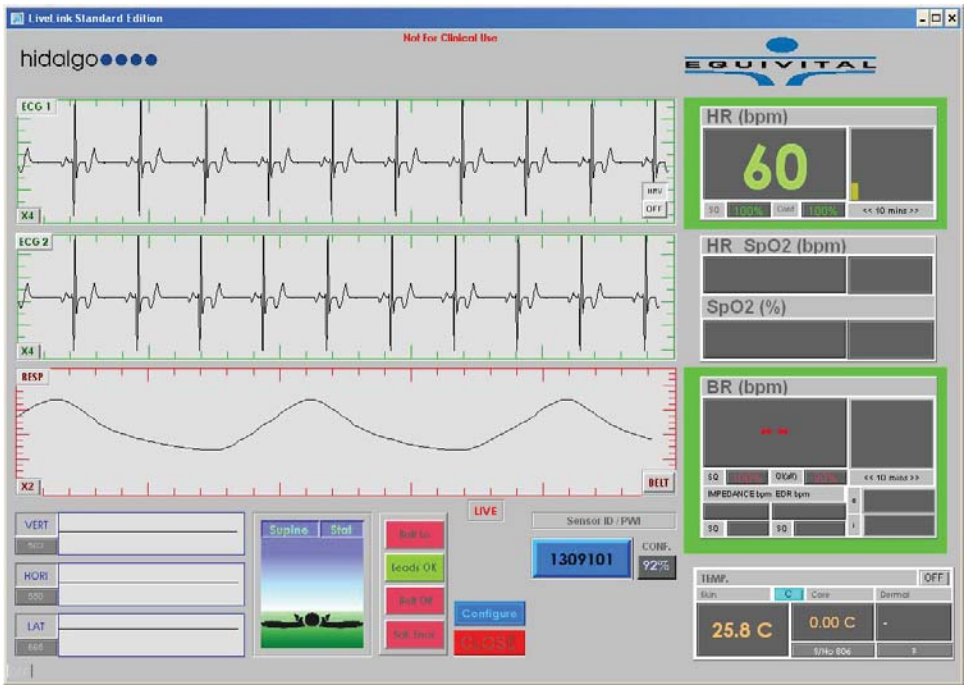

Figure 3.20: LiveLink Waveform screen[41]

is positioned at the bottom right corner to fill up the right side of the screen. Skin temperature measured from the EQ01, core pill temperature and dermal patch temperature can be displayed in a resolution of 0.1 degrees C every 15 seconds. Values can be switched between units of Celsius, Fahrenheit and raw values. The left and middle part of the screen is made up of three waveform panels and one panel containing a variety of information along configuration options. On the top of this part of the screen are two panels for each of the ECG leads. Scaling of the waveforms can be adjusted with a click on the button in the bottom left corner of the panel, additionally switching to the heart rate variability panel, which plots the R-R Interval on the second panel, is implemented in the HRV button in the lower right corner. The third panel from above is the respiration panel, displaying the current respiration waveform. Again a button for scaling appears in the lower left corner and a button in the lower right corner allows for switching between Belt or impedance source for respiration data. The bottom panel is made up of a number of smaller panels and indicators. On the left side

the accelerometer waveforms for the three axes vertical, lateral and horizontal are depicted. To the right of this panel is the activity panel with a stick figure imitating the body posture and text describing the posture and motion. In case of fall detection the border colour of the panel turns to red with short text below the top corner saying "fall confirmed". Next up alongside the activity panel are the alarm indicators in form of status boxes, that change colour to red if an alert occurs. Alerts for Battery status, Lead contact, Belt connection to the EQ01 and ECG Saturation of the range. Finally the last indicators in this bottom panel are the Physical Welfare Indication PWI that changes colours of the Equivital device ID depending on status, Stop button to go back to the main menu, a Live indicator that flashes when a data block has been read and a configuration button. When clicking the configuration button, a screen with settings to adjust is displayed. Among these settings are operating mode, which has to be set to full disclosure mode for the software to be able to display waveforms, an ECG filter for ambulant or diagnostic filters and options to open and stop replay data. There are a few more but these turned out to be the most important settings during development.[41]

3.2.4 Equivital SEM Customize

Hidalgo provides a software utility for Windows XP or Vista to adjust the Equivital EQ01 to the users preferences or application requirements. With the help of this application, the behaviour and adjustments can be customized[42]. During installation the user is asked to install SEM either as an engineering user or standard customer. For this project the standard customer option was sufficient and is explained in further detail to illustrate the use of this software and its integration. The engineering user configurations are explained further below, to describe the possibilities provided. An overview and introduction of the installation process is given in the instructions by Hidalgo. Here we will concentrate on the options and its effects as well as the settings used to achieve the projects' results. After the SEM Customise has been successfully installed, the EQ01 has to be connected to the computer with a wired serial port connection. This can be done with the delivered USB adapter or the programming lead, connected to a USB port or

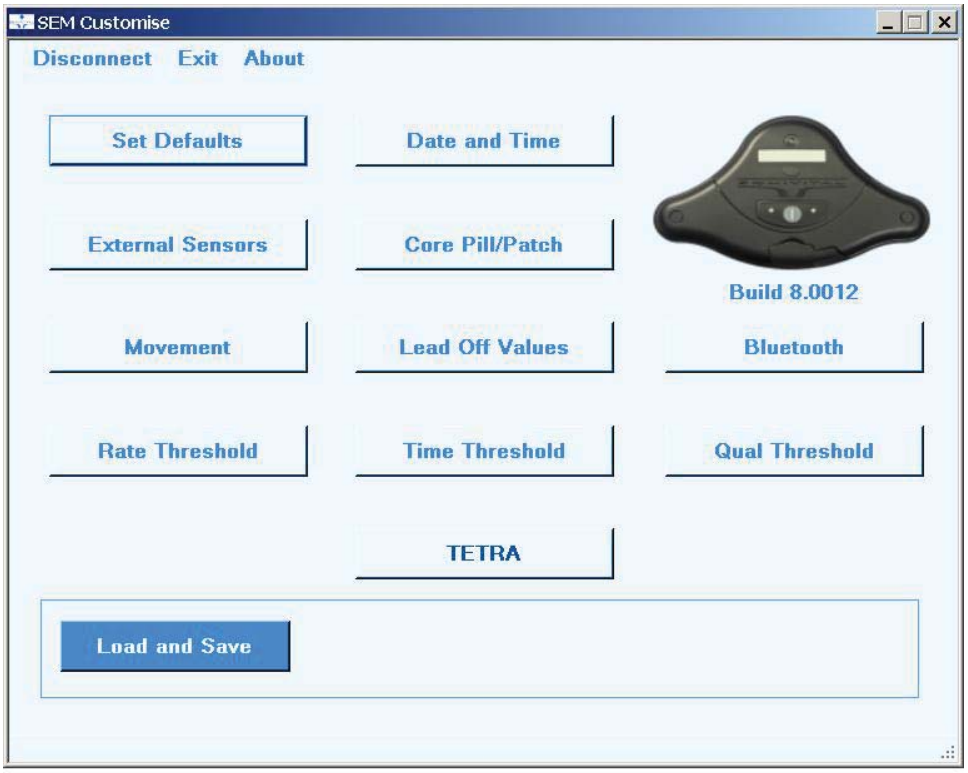

Figure 3.21: SEM main screen[42]

RS-232 port. Once the EQ01 is connected to the computer, all functions of the SEM Customise are enabled and can be edited in no particular order. Instead of changing every option by hand each time, a number of pre set configurations can be chosen as well as loading and saving edited or newly constructed configurations in .xml format files. All changed settings and options must be saved to be permanently on the device. From the main menu, when choosing the first item "set defaults" the sub menu with options for general settings, that are used when the device is switched on, comes up. These are the most important configurations when setting up the EQ01 for use with other devices, as these are general operating options. The menu item Date and Time will guide you to a sub menu which displays computer and EQ01 time that can be saved onto the device. Additionally wake ups can be set for the device once or twice a day to keep it active. To manage and configure external sensors the identical named button has to be

pressed to get to the sub menu. Core Pill/ Dermal Settings will ?allow the user to read and update the core capsule pill / dermal patch start-up behaviour and the values of various parameters associated with the core capsule pill or dermal patch. Configuration of the Accelerometer calibration values and fall sensitivity settings can be done under the menu item Movement Settings. The Accelerometer can be auto calibrated with the device put in different positions as ordered on screen or manually with maximum and minimum values. Next up on the main menu screen are the lead off values settings. It detects the quality of the signal, with low values corresponding to a good connection and high values to bad connection, and partitions it in three states. One is the green state, where the connection to the EQ01 is good for ECG and impedance respiration measurements. Another is the amber state, in which only the ECG measurements are good enough with the present connection. And finally the red state in which both measurements cannot be undertaken because of the low quality of connection. Transitions between the three states can be set in ms in both directions from amber to green and vice versa and from amber to red and vice versa. Bluetooth settings allow the user to display and update settings and parameters related to the wireless connection. Bluetooth sniff mode parameters, to save energy, the Bluetooth PIN, a pass key for connection, and the Bluetooth module power, switched off when no power should be used to the radio module, can be configured. Finally the user is able to chose from three distinct Threshold settings menus. The rate Threshold settings will allow for setting values when alerts should be raised for heart and breathing rates, time threshold for time values and quality threshold for qualoty values. Under the button load and save, older configurations or ones delivered with the EQ01, can be loaded to the device or saved to the computer, all in .xml format files. The SEM customize will inform the user if this transaction between EQ01 and computer has been successfully carried out. The Engineering feature settings allow the user to apply some advanced settings that are only accessible if during installation of the SEM customization utility the option for engineering user is selected. To present the customizability of the Equivital sensor and its applicability to future projects, a short description of this function is given. Once readily installed the user can choose between two ways to configure the engineering settings on the EQ01 by selecting either auto or manual configuration. Both selections

will offer the same options to be configured, the auto configuration though will be presented in a sequence where the user is guided through all options. With the manual configuration a selection of options can be made. From the main menu, nine sub menus can be entered with a click on the corresponding button. First in line is the Interface type option with the selection of radio types used with the EQ01. Temperature Calibration will let the user modify the temperature calibration values in 17 calibration points within the range of 0 to 1023. Another menu item is the option to set bluegiga serial speed. The modules' baud rate can be set to either 38400 baud or the default 115200 baud, which might be useful for receiving devices with limited performance. Likewise options for adjusting the lead off values, selecting the battery type, changing serial number and testing and adjusting the vibrator exist. A safety feature of the EQ01 can also be accessed by entering the crystal comparison menu. When the device executes the Power On Self test it also performs the crystal comparison test, where frequencies of two resonators from the microprocessor are compared. If the watch crystal and ceramic resonator differ in their frequencies by more than say one per cent, the device might be faulty. This test can also be undertaken inside this menu to additionally check for correctness of operation. [42]

Chapter 4

Project description

This project is about the design and implementation of an Android based sensor application on a smart phone. Smart phones offer more advanced computing ability and connectivity recently. They have a lot of integrated sensors, such as GPS, accelerometer, microphone, camera etc. as described in the chapter "tools used for this project". In addition, a smart phone application can employ remote wireless body sensors, such as the Hidalgo Equivital EQ01. This way, the smart phone can access the data shared by the sensors in the EQ01 over a wireless connection. The EQ01 is capable of providing information about the subjects' electrocardiogram (ECG), skin temperature , position and movement, as well as activities of the lungs through its sensors. This provides useful information that can be used to describe or give a status on the subjects' health. Again, please refer to the chapter tools used, that covers all hard- and software components utilized to develop this application. This application is designed for exchanging data via Bluetooth between a HTC Sensation Smart phone and the Hidalgo Equivital EQ01, a portable health monitoring sensor. Use cases for this whole system, containing of smart phone, sensor and application could be wide spread from health monitoring to movement recognition. All received data is presented in real time on the smart phone, where the user can check and react to data with for example, an emergency call. These additions help making the application more user friendly and safer. The user interaction has been designed with two scenarios in mind, the presentation of data for the experienced analyst and the average user. Collections of data can be stored in a database and be uploaded to

a remote server with the help of an internet connection. As the EQ01 provides more than just a few sensor values extracted from the body, a selection of values that are presented has been chosen. Amongst others the ECG, Accelerometer and Respiration waveforms are displayed as well as the position, motion and temperature. A Bluetooth connection can be adhered throughout the application, even when the home screen is reached. Every session is managed with login of user and the corresponding information. More on all the details will be revealed in the following sections. This project can be fragmented into two parts, an application oriented and research oriented part. The application oriented part is the development of an android application that displays health status information of a subject from the Hidalgo Equivital EQ01 on a smart phone, supporting the android mobile OS. The basic functionalities are applied to a newer version of the Android API level and makes use of the best practices recommended by Google in order to increase user experience, reduce power consumption and improve performance. Functionalities like signal acquisition, data storage, data analysis, data transmission to subsystems like a stationary server and other useful additions are incorporated. In theory it is also possible to let the system automatically alert medical stations through Short Message Service, based on the analysis' results. Added possibilities for a continuously monitoring of the subject from a stationary server can allow for an early detection of potential illness or general irregularities. The system therefore can consists of three or only two parts. The wearable sensor that records different types of data on a subject and transmits it via Bluetooth. A smart phone that acts either as a gateway by forwarding, or as the diagnostic device by analysing the data received from the sensor. Finally an optional server for processing or reviewing the data, when the smart phone is acting as a gateway. All classes have been stored into purpose based packages to organize the structure of the classes better and to access certain parts of the application faster. There are seven packages called add ons, cache, drawing, login, monitoring, upload and storage, which are all stored inside the directory com.project.bluetooth. The classes inside the packages serve the same purpose as described by the packages' name. For instance the add ons package contains classes that are not essentially needed for the applications core functions and features, but make user interaction more pleasant and provide additional features like a help screen to support the

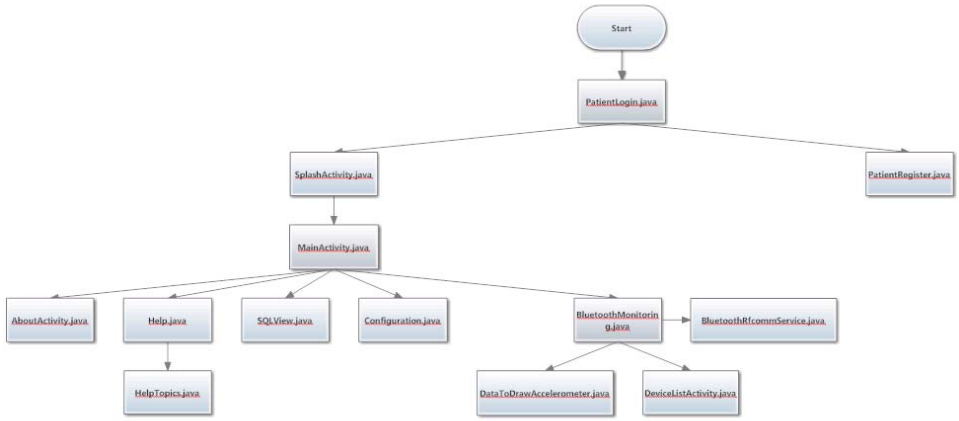

Figure 4.1: Class tree

user. Inside the cache package folder, prototype classes for a cache based buffer are residing, to be used in the future to preform data processing without the help of the sd card. The drawing package contains classes that plot the data which was acquired with the help of the classes inside the monitoring package. Former package also contains classes to manage and administer Bluetooth connections. Finally there are the login package, which contains classes that handle login and registration of a user, the storage package, with classes to manage storing processes, and the upload package, which as the name suggests contains classes to handle uploading processes. Inside the packages are classes that not only extend the activity class but also Services or SurfaceViews. An overview of all classes can be seen in figure 4.1 and the order in which the activities and service are executed is depicted as arrows. The manifest file defines the minimum required Android API level as 11 with the target level of 15. Additionally permissions have been set in the manifest file in order to make use of all features the components offer. For internet and Bluetooth connection, emergency calls and furthermore permissions for the application and smart phone to reside in a state. The manifest file with permissions, android version and classes is in the appendix. In the following, all application components are presented in the way the user is experiencing them visually upon the use of the application. After installing the application it will

appear on the smart phones list of applications with a custom made icon, as in figure 4.2 shown. Once the user has touched the icon, the application called Bluetooth monitoring will begin. The sections of this chapter are divided into application peripherals, Information, Bluetooth and data storage and upload.

Figure 4.2: App icon

4.1 Application peripherals

Figure 4.3: Application peripherals

After initiation of the application, the user is forwarded to the class PatientLogin.java. The login screen invites the user to type in the user name and password in order to be logged in and be forwarded to the application contents. At the first time the application is used by someone or a new user should be registered, a click on the text "I don't have an account. Register me" will bring the user to the registration screen, part of the PatientRegister.java class. Here additional informations are requested to enter in order to create a new account. These entries are related to the user's personal information and vital signs, like full name,

password, user name, but also weight, height and an additional entry for miscellaneous notes. Only the user name, password, height and weight are necessary for the user to type in. If an entry of a user name already exists or the inserted format is not compatible, a notification pops up with an error message. In the background, the entered data is processed to calculate the BMI of the user, in order to use this for further evaluation on the health status. All registered informations along with the user name and password needed for logging in are stored inside an SQL database. The database is stored locally on the smart phone and is accessed by every login for verification or registration to enter new user data. A registration of users provides features for the future development of the application. Data can be accessed and used to measure the likelihood of critical conditions the user can fall into during monitoring of the vital signs. When the application is deleted from the smart phone, the SQL database and its stored data are also deleted from the smart phone. By pressing the button Register in the register screen and the button login in the login screen, the main menu of the application is reached and the user is logged in. Beforehand a splash screen is shortly displayed that does not make use of any user interaction. The splash screens class' SplashActivity.java purpose is to load resources from the internet on a background thread. Actual loading of the parts is carried out with the help of the LoadingTask.java class. An approximate loading time of two seconds allows for a few websites to be loaded, that are used later on to provide information about the application, its components and general background knowledge, in the help screens. While the background thread is executed, the user can check the progress by following the progress bar on the bottom of the screen. In the centre the two involved Universities logos, the Universidad de Granada and the University of applied science Münster, are displayed. Inside the main menu, the screen is made up of the menu items, with name and icon of the classes, and the actionbar at the top of the screen. As shown in figure 4.3 the menu icons with the corresponding screens to which the user can switch to from the main menu are monitoring, configuration, database, help and about from top to bottom. We will elaborate on these classes and their features in the next sections. Behind the icon monitoring, the main purpose class of this application can be found, including Bluetooth connection to the EQ01 and presentation of its data. The

configuration menu has a few settings incorporated to configure the behaviour of the application. For having a look at the collected data during monitoring the database screen can be entered, where the option to upload the recorded data to a remote server can also be performed. Information about the parts of the application and the involved parties is provided by clicking the buttons help or about. At the top of the screen is the actionbar incorporated, instead of clicking on the smart phones configurations button. The actionbar is made up of quickly reachable buttons that perform actions or navigates the user to certain parts of the application. It is incorporated not only on the main menu. From left to right, the first thing to notice is the applications icon, which can be pressed to go back to the main menu. To the right of the menu icon the user name of the currently logged in user is displayed. Further on the right side are the four buttons that provide enabling and disabling Bluetooth and Wi-Fi as well as navigating to the database and configurations screen. If the user decides to leave the application from any screen, with the main menu button on the smart phone, a logout will automatically be performed. With every start up, the user has to log in again. The underlying code of the classes behind the screens of this first section of initial application start up is presented in the tables below.

4.2 Information

This section covers the configuration.java settings and the two informal classes help.java and about.java. They can all be reached from the main menu, in addition the configurations menu can be reached from the actionbar as already specified in the previous section. Plain text is displayed when opening the about screen from the main menu. Information about the involved parties and the developers can be found here. The user is welcomed to the application, below the same image as displayed in the splash screen, composed of the logos from the involved universities. When the bulb icon is selected from the main menu, the user will be directed to the main help screen.

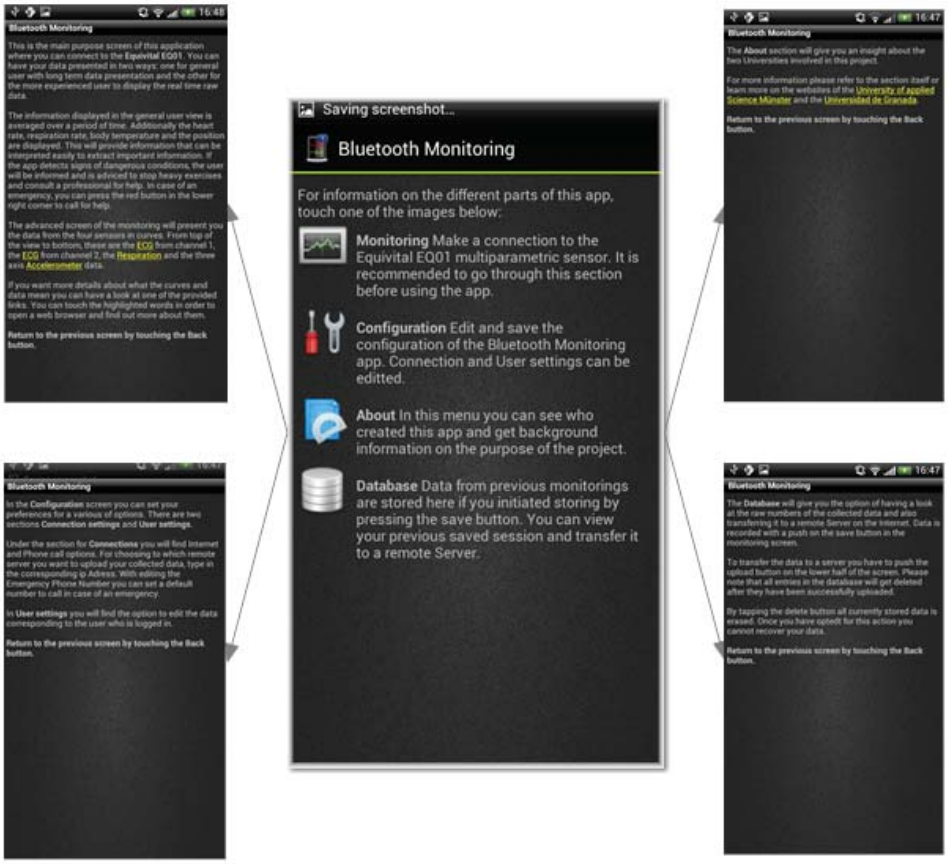

Figure 4.4: Help screens

This screen is build similar to the main menu. It displays all menu icons from the main menu except its own, with an explanatory text next to it. The icons are as well clickable, to forward the user to a more detailed explanation and guidance about how to use the features of the applications classes. Incorporated are links to internet pages where the user can gather more background knowledge about the displayed information, like the ECG, if desired. These links have been loaded on a background thread during start up, while the splash screen has been shown. By clicking the highlighted parts, the android web browser will open automatically with the internet pages loaded. Going back to the main menu, the user will find an icon called configuration. In here settings can be edited that will be saved with the help of shared preferences. As illustrated in the chapter tools used, the android shared preferences store the application specific data, as edited in the configuration class, as long as the application is installed on the smart phone. The saved settings are not user specific and remain the same, if not edited, throughout multiple sessions. Shared preferences can be accessed by any component inside the application and therefore a variety of settings have been incorporated. One of them is the checkbox enable rfcommservice. If this checkbox is disabled, a connection over Bluetooth will not be established. The checkbox is figurative of a boolean that is checked before initialising a Bluetooth connection inside the methods of the bluetoothmonitoring.java class. If it is enabled and therefore set to true, the connection will be established, otherwise the user is directed to the configurations screen to edit the settings. Another option the user has is to type in a telephone number that is called, when the user decides to push the emergency call button inside the bluetoothmonitoring.java class. The emergency call number has a default number that is called when no number has been inserted by the user. Finally the IP address of the remote server can be edited, for the data to be uploaded to.

4.3 Bluetooth

The classes described in this section form the main part of the application, including handling of Bluetooth connection, collecting and processing data, as well as presenting it appropriately. Upon choosing the icon monitoring in the main

menu, the class bluetoothmonitoring.java is reached. What is presented to the user in this screen is also called the regular user view. To use this part of the

Figure 4.5: Regular View

application the item enable rfcommservice has to be checked in the configuration. If the smart phone does not have Bluetooth enabled, a dialogue will appear on the screen informing the user that an application wants to turn on Bluetooth. The user has two options, one is to deny permission and be directed back to the main menu or confirm and stay inside the view with Bluetooth being switched on. Thereupon Bluetooth is actually switched on on the smart phone, the view with

its panels, buttons and text is displayed. To make a connection to the Equivital, which by now should be strapped around the body and switched on, the button settings on the smart phone has to be pressed. A pop up menu with a tab to scan for devices and to store data comes up. Pressing the former will result in a dialogue displayed with the option to connect to already paired devices or scan for devices. Behind this dialogue is the DeviceListActivity.java class that listens to discovered devices to connect to with the help of a broadcast receiver. Once discovered, the devices are added to the list out of which the user can select a device to connect the smart phone to. To connect to a device, it has to be chosen from the list. For handling connections the service BluetoothRfcommService.java is responsible. It also has a thread that listens for incoming connections, a thread for connecting with a device, and a thread for transmitting data over Bluetooth. The activity BluetoothMonitoring.java starts the service automatically when the request to enable Bluetooth occurs. From this point on the activity is bound to the service as long as it is not killed by exiting the activity with the back button, changing the boolean in the configurations or having the system kill the service due to low memory. When the smart phone and a device like the EQ01 are finally connected, a toast is displayed informing the user that the connection is established. Now the data transmitted over the bluetooth network from the EQ01 to the smart phone is presented in real time. For this the BluetoothRFcommService transmits data to the bluetoothmonitoring activity where it is processed and displayed. As described in the chapter tools used for this project, the incoming raw data from the EQ01 is converted. The converted data from the sensors is than displayed in three basic presentations. Temperature, heart rate and respiration rate values are categorized into four parts. The temperature for example is fragmented into sections in the ranges from zero to 34C, 34C to 36C, 36C to 40C and 40C to above. For the heart and respiration rate also exist these categorisation of values. If the value of the temperature lies within one of these parts, a number of bars will light up from green to red in the middle of the screen. A lower part corresponds to a lower number of bars light up and vice versa. From top to bottom of this screen part, the bars belong to the heart rate, the respiration rate and to the temperature. At the bottom of the screen the user can get information about the battery status, apnea occurrence, vital signs and irregular heart beat.

These four panels turn red if the EQ01 sends the corresponding data messages over Bluetooth. Same goes for the panels that reside on the right hand side of the screen. They turn red if the values exceed a certain maximum or minimum which can be dangerous for the human body. Here we have the Pulse, breathing rate, temperature and position displayed from top to bottom. The actual numbers are displayed within the first three panels, while the position panel displays the current status of the body posture with the help of images. A stick figure imitates the position in real time. If the user happens to have an emergency or is in the need of help, the emergency call button is available. This will set out a call to the European emergency call number 112 by default. Changing this number can be done so by entering a telephone number inside the configuration class. To proceed to the expert view the user has to push the buttons for either of the three data waveforms. Three classes can possibly be reached either the DataToDrawRespiration.java for plotting the respiration waveform, the DataToDrawAccelerometer for the three axes accelerometer waveforms or the DataToDrawECG fot the ECG waveform graph. These classes are basically the same, except that the DataToDrawAcceleromter is constructed for three waveform plots at the same time. By entering one of these classes, they are bound to the BluetoothRfcommService automatically and upon exiting are unbound from the service. This provides all functionalities of the service for the DataToDraw classes to draw the waveforms. These activities extend a SurfaceView where drawings can be carried out on. An example is given in figure 4.6 with the DataToDrawAccelerometer activity drawing all three axes with the help of the ServiceView.

4.4 Data storage and upload

The data from the sensors of the Equivital EQ01 can be saved to a database and uploaded to a remote server. For this purpose a number of classes are implemented in this project. Collecting data and storing it in an SQL database is carried out by the DatabaseHelperClass.java class and its methods. The Database and its stored entries can be viewed as a list inside the SQLView.java class. This is the only class that has a user interface from this group of components. Additionally the JSONParser.java and NewProductActivity.java classes provide uploading

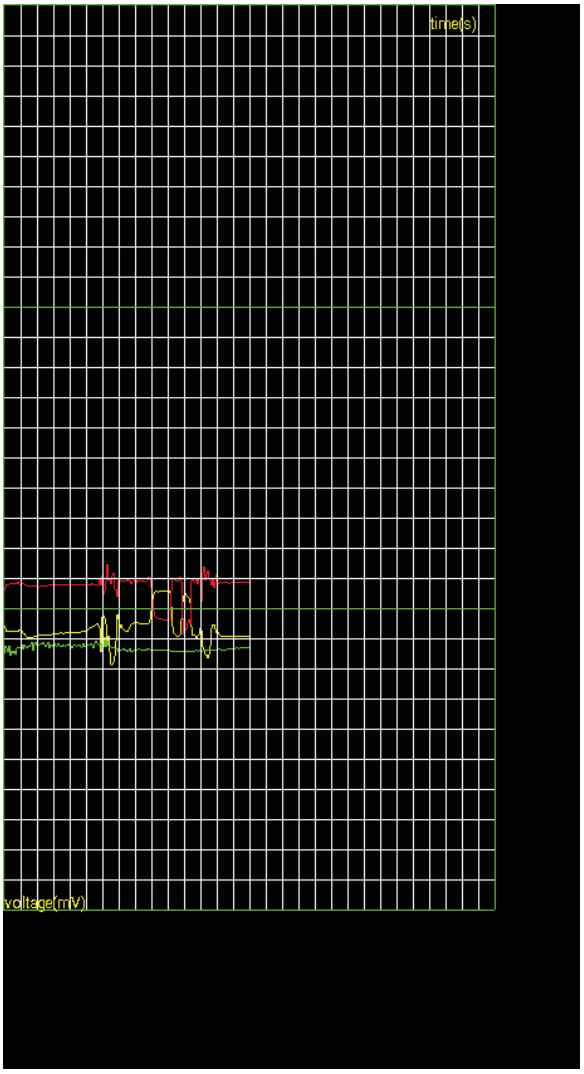

Figure 4.6: DataToDrawAccelerometer.java

of the data to a remote server. They check for the status of the internet connection, parse data JSON format, similar to XML, and notify the user about the status. Inside the Bluetoothmonitoring.java class the storage of data can be initiated and stopped with a click on the expert view button, residing on the lower right hand sight. This way, the user will reach the DataToDrawAccelerometer.java class where storing data to the database starts automatically and is cancelled when the screen is exited. If the user can see the data waveforms and

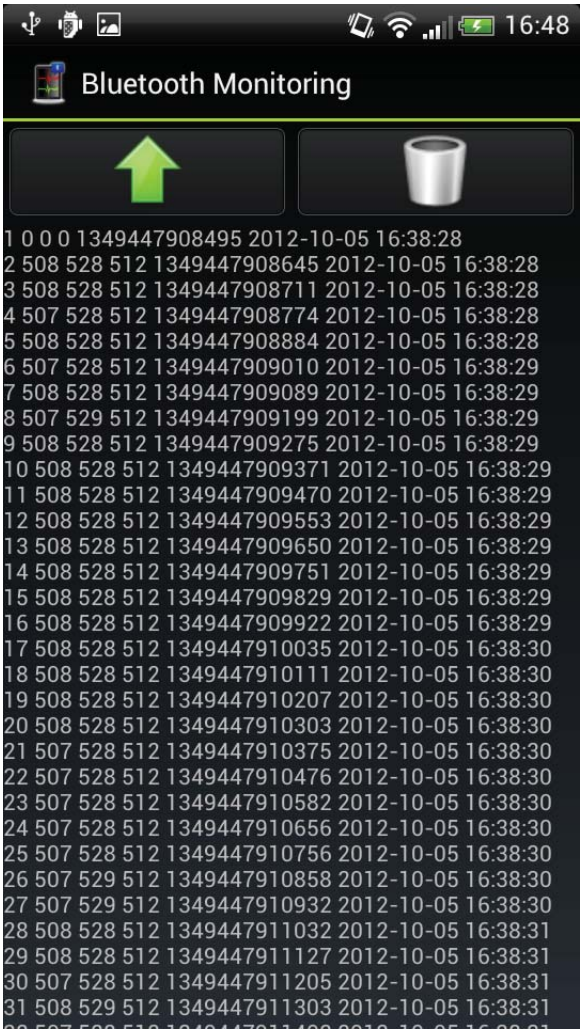

Figure 4.7: SQLView.java

the storing progresses, the data from the class will be stored in a table which makes up the SQL database. For the Accelerometer, the three axes sensor data, are stored into the first three columns of the table. Next to the data of the sensors, the time stamp is stored, one in date format of yyyy-MM-dd HH:mm:ss and the other in milliseconds from the first January of 1970 to now. Every time new data from the sensor has been processed by the smart phone, with the help of this application, a new row is entered into the database. Figure xx displays the view from the class SQLView.java. Above the table with entries from the sensors

reside two buttons next to each other. The arrow on the left is for uploading the data to a remote server and the dustbin on the right is for clearing the SQL database. If the user decides to push the upload button to transfer all data from the local database into a database on a remote server, the process is carried out row by row. The class responsible for uploading is a subclass of AsyncTask, which allows the user to perform actions, like going to another view, at the same time as this means that the upload is carried out on a background threat. Every time a row has been successfully uploaded it will be deleted from the table before the next row will be uploaded. This has the big advantage that when an error during uploading occurs and the application is killed, all data is save as it is either on the server or still on the smart phones database. To provide this function the connected server has to have the same SQL database with the same parameters already installed. For this a number of php files were used to set up a server with the XAMPP program for windows.

Chapter 5

Conclusion and future work

In this chapter I will conclude this thesis by summarizing our project and discussing future work.

5.1 Conclusion

This thesis has presented an Android application that is capable of connecting a smart phone to the Equivital EQ01 mobile health care sensor. The EQ01 is strapped around the body and transmits data over a Bluetooth connection to the smart phone. Our application on the smart phone communicates with the EQ01 in order to gather data, present it on screen, store it locally and upload it to a remote server. These essential characteristics of the resulted application are completed by an additional set of features. Incorporated is a user interface that is consistent and well-arranged all over the application. Help on how to use the application is provided in addition to recommended further reading of background information. An expert user view for advanced data waveforms is available as well as a regular user view that presents more basic data during monitoring. The log in and registration feature of the application provides a session based monitoring, with advanced user information. Decoding of the raw data that is provided by the EQ01. A Bluetooth connection that is established from a service which can run throughout the application and even outside of it. And finally the application is capable of providing information about the heart rate, respiration rate, temperature, position, as well as accelerometer, ECG and

respiration waveforms. In case of an emergency the user is able to get help with the implemented emergency call option.

5.2 Future work

he application introduced in this thesis can be further developed in many different directions. Possible modifications could make it applicable for supervision of athletes, permanent monitoring of chronic diseases or daily living support of elderly in the form of fall detections. In which direction future steps are leading depends solely on the requirements and desired finished product. In the following a few future steps are presented.

A logical step in further developing the application could be to make it available for more people to benefit from it. The number of sensors that can be used for the application could be increased, by adapting to their technical specifications. To provide a low cost system that maybe aims to extract only information of a single vital sign like the ECG waveform, a sensor could be developed. Another way to increase the number of users is to make the application available on other mobile operating systems.

To further analyse and process the collected data from the EQ01, the implementation of an Activity recognition chain on Android could be made. This can give the subject under test a diagnosis of its health status and reveal diseases and diagnoses. The work would mainly deal with the development of an efficient detection algorithm using the signals provided by the Equivital EQ01, so that detecting diseases at initial stages is possible. The proposed arrhythmia detection algorithm may therefore be helpful to the clinical diagnosis. For this, different steps of an activity recognition chain would have to be implemented. The data would have to go through the preprocessing stage, segmentation, feature extraction and finally a classification technique. Preprocessing the data is meant for converting the raw data into a usable format. This is achieved by applying different filters onto the data stream. It then would be undergoing the

segmentation process. Here the filtered signal is sliced into divisions of a certain time or frequency base. Applying these techniques achieves small packages of important information that is easier to extract certain information from. The last step of preparing the signal for the classification technique would be the feature extraction. By extracting certain values which correspond to different features of the signal, the classification of the signal can be made. The signal for the classification process is left with only the extracted features, which are the most important characteristics of the data stream for the different types of classifications. The result of this four step process would be the recognition of an activity, which in this process for example can be the recognition of a certain heart disease. The decision logic encoded may be complex, yet the resulting application would provide a correct and simple to understand advice or diagnosis.

Another possible future step could be to extend the supported Bluetooth protocols and profiles. The specifically for health applications developed Bluetooth Health Device Profile HDP would make the application able to connect to a variety of additional devices. The general idea of expanding connection variety could be taken as far as to develop a software layer that is capable of connecting to any device over the available communications standard on smart phones. Without informations of the sensors specifications, like the data types and message structure, the layer could act as a registration instance, where upon the first few seconds of operation all required data to establish a connection are exchanged. Within these first few seconds, information that are needed to process, plot and handle data in general could be exchanged. This way the application would not have to be developed for a specific device, but rather be able to connect to any device with the given communication standards and exchange all needed data in the initial connection.

Maybe in the very far future, sensors could be implemented under the skin after childbirth. They would grow with the body, be powered by it and use it for transmission of data, as an antenna for example. This may sound fictional, but BANs already are established with the help of sensors that reside under the skin.

In which direction the future development of the application will go to is uncertain. The version presented in this thesis provides basic functionalities, that can easily be extended or modified to suit the needs of another application.

Appendix

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.project.bluetooth.monitoring"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.CALL_PHONE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission
    android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"
    />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.UPDATE_DEVICE_STATS"
    />
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />

    <uses-sdk
        android:minSdkVersion="11"
        android:targetSdkVersion="15" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- com.project.bluetooth.addons -->
        <activity
            android:name="com.project.bluetooth.addons.AboutActivity"
            android:configChanges="orientation|keyboardHidden"
            android:label="@string/app_name"
            android:screenOrientation="portrait"
            android:theme="@android:style/Theme.NoTitleBar.Fullscreen" />
        <activity
            android:name="com.project.bluetooth.addons.SplashActivity"
            android:configChanges="orientation|keyboardHidden"
            android:label="@string/app_name"
            android:theme="@android:style/Theme.NoTitleBar.Fullscreen" />
        <activity
            android:name="com.project.bluetooth.addons.MainActivity"
            android:configChanges="orientation|keyboardHidden"
            android:label="@string/app_name"
            android:theme="@android:style/Theme.NoTitleBar" />
        <activity
            android:name="com.project.bluetooth.addons.Help"
            android:configChanges="orientation|keyboardHidden"
            android:label="@string/app_name" />
        <activity
            android:name="com.project.bluetooth.addons.HelpTopics"
            android:configChanges="orientation|keyboardHidden"
            android:label="@string/app_name"
            android:theme="@android:style/Theme.Light" />
        <activity
            android:name="com.project.bluetooth.addons.Configuration"
            android:enabled="true"
            android:label="Settings"
            android:theme="@android:style/Theme.NoTitleBar.Fullscreen" />

        <!-- com.project.bluetooth.drawing -->
        <activity
            android:name="com.project.bluetooth.drawing.DataToDrawECG"
            android:label="@string/app_name"
            android:screenOrientation="portrait"
            android:theme="@android:style/Theme.NoTitleBar.Fullscreen" />
        <activity
```

```
            android:name="com.project.bluetooth.drawing.DataToDrawAccelerom
            eter"
            android:label="@string/app_name"
            android:screenOrientation="portrait"
            android:theme="@android:style/Theme.NoTitleBar.Fullscreen" />
        <activity

            android:name="com.project.bluetooth.drawing.DataToDrawRespirati
            on"
            android:label="@string/app_name"
            android:screenOrientation="portrait"
            android:theme="@android:style/Theme.NoTitleBar.Fullscreen" />



        <!-- com.project.bluetooth.login -->
        <activity
            android:name="com.project.bluetooth.login.PatientLogin"
            android:label="@string/app_name"
            android:theme="@android:style/Theme.NoTitleBar.Fullscreen" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
                />
            </intent-filter>
        </activity>
        <activity
            android:name="com.project.bluetooth.login.PatientRegister"
            android:label="@string/app_name"
            android:theme="@android:style/Theme.NoTitleBar.Fullscreen" />

        <!-- com.project.bluetooth.monitoring -->
        <activity

            android:name="com.project.bluetooth.monitoring.BluetoothMonitor
            ing"
            android:configChanges="orientation|keyboardHidden"
            android:screenOrientation="portrait" />
        <activity

            android:name="com.project.bluetooth.monitoring.DeviceListActivi
            ty"
            android:configChanges="orientation|keyboardHidden"
            android:label="@string/select_device"
            android:theme="@android:style/Theme.Holo.Dialog" />

        <service

            android:name="com.project.bluetooth.monitoring.BluetoothRfcommS
            ervice"
            android:exported="false" />

        <!-- com.project.bluetooth.storage -->
        <activity
            android:name="com.project.bluetooth.storage.SQLView"
            android:label="@string/app_name" />

        <!-- com.project.bluetooth.upload -->
        <activity
            android:name="com.project.bluetooth.upload.AllProductsActivity"
            android:configChanges="orientation|keyboardHidden"
            android:label="@string/app_name" />
        <activity
            android:name="com.project.bluetooth.upload.NewProductActivity"
            android:configChanges="orientation|keyboardHidden"
```

```
AndroidManifest.xml                                                05.10.2012

            android:label="@string/app_name" />
        <activity
            android:name="com.project.bluetooth.upload.EditProductActivity"
            android:configChanges="orientation|keyboardHidden"
            android:label="@string/app_name" />
        <activity

            android:name="com.project.bluetooth.upload.AndroidDetectInterne
            tConnectionActivity"
            android:configChanges="orientation|keyboardHidden"
            android:label="@string/app_name" />
    </application>

</manifest>
```

3

Figure 1: Manifest file

```
/*
 * Copyright (C) 2009 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package com.project.bluetooth.monitoring;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.text.SimpleDateFormat;
import java.util.Date;

import com.project.bluetooth.addons.Configuration;
import com.project.bluetooth.drawing.*;
import com.project.bluetooth.monitoring.BluetoothRfcommService.LocalBinder;
import com.project.bluetooth.storage.DatabaseHelperClass;

import android.annotation.SuppressLint;
import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.ActivityNotFoundException;
import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.ServiceConnection;
import android.content.SharedPreferences;
import android.content.SharedPreferences.OnSharedPreferenceChangeListener;
import android.net.Uri;
import android.os.Bundle;
import android.os.Environment;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.preference.PreferenceManager;
import android.util.Log;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.WindowManager.LayoutParams;
import android.widget.Button;
import android.widget.ImageButton;
import android.widget.TextView;
import android.widget.Toast;

/**
 * This is the main Activity that displays the current chat session.
 */
```

```
public class BluetoothMonitoring extends Activity implements
OnClickListener,
        OnSharedPreferenceChangeListener {
    // Debugging
    private static final String TAG = "BluetoothMonitoring";
    private static final boolean D = true;

    // Message types sent from the BluetoothRfcommService Handler
    public static final int MESSAGE_STATE_CHANGE = 1;
    public static final int MESSAGE_READ = 2;
    public static final int MESSAGE_WRITE = 3;
    public static final int MESSAGE_DEVICE_NAME = 4;
    public static final int MESSAGE_TOAST = 5;
    public static final int MESSAGE_GET_DATA = 6;

    // Key names received from the BluetoothRfcommService Handler
    public static final String DEVICE_NAME = "device_name";
    public static final String TOAST = "toast";

    // Intent request codes
    private static final int REQUEST_CONNECT_DEVICE = 1;
    private static final int REQUEST_ENABLE_BT = 2;

    // Name of the connected device
    private String mConnectedDeviceName = null;
    // Local Bluetooth adapter
    private BluetoothAdapter mBluetoothAdapter = null;
    // Member object for the chat services
    private BluetoothRfcommService mRfcommClient = null;
    // Area for the Waveform view

    // Button for emergency call:
    private Button button_call;
    private Button button_nextview;
    private ImageButton imageButton_position;

    private TextView panel_heart_rate;
    private TextView panel_respiration_rate;
    private TextView panel_temperature_c;
    private TextView panel_bottom1;
    private TextView panel_bottom2;
    private TextView panel_bottom3;
    private TextView panel_bottom4;

    // Maximum values for the vital sensed data
    private double maximumBR = 50;
    private double minimumBR = 8;
    private double maximumHR = 180;
    private double minimumHR = 40;
    private double maximumT = 40;
    private double minimumT = 34;

    boolean mExternalStorageAvailable = false;
    boolean mExternalStorageWriteable = false;
    String state = Environment.getExternalStorageState();

    File file;
    FileReader filereader;
    // public static BufferedReader bufferedreader_respiration;
    // public static BufferedReader bufferedreader_ECG2;
    BufferedReader bufferedreader_ECG1;

    String s = new String();

    String p = new String();
    static int test = 0;

    private boolean mBound = false; // are we bound to the
```

```
                                // BluetoothRfcommService?
    private SharedPreferences preferences = null;

    // DBAdapter db = new DBAdapter(this);

    String dataMessage;

    // IncomingBuffer RingBuffer;

    // Strings used for Storage
    public String bluedata;
    private String position;
    private String heartRate;
    private String respirationRate;
    private String temperature;
    MyReceiver myReceiver;
    private Button button_ind_1;
    private Button button_ind_2;
    private Button button_ind_3;
    private Button button_ind_4;
    private Button button_ind_5;
    private Button button_ind_7;
    private Button button_ind_6;
    private Button button_ind_8;
    private Button button_ind_9;
    private Button button_ind_10;
    private Button button_ind_11;
    private Button button_ind_12;
    private Button button_ecg_go;
    private Button button_resp_go;
    private Button button_acc_go;

    @SuppressLint("NewApi")
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (D)
            Log.e(TAG, "+++ ON CREATE +++");

        preferences = PreferenceManager.getDefaultSharedPreferences(this);
        preferences.registerOnSharedPreferenceChangeListener(this);
        // Set up the window layout
        setContentView(R.layout.monitoring_main);

        // bind to bluetooth service
        // mServiceIntent = new Intent(this, BluetoothRfcommService.class);
        // bindService(mServiceIntent, mConnection,
        Context.BIND_AUTO_CREATE);

        // Keep the device's screen turned on and bright
        getWindow().addFlags(LayoutParams.FLAG_KEEP_SCREEN_ON);

        setupUI();

        // Get local Bluetooth adapter
        mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

        // If the adapter is null, then Bluetooth is not supported
        if (mBluetoothAdapter == null) {
            Toast.makeText(this, "Bluetooth is not available",
                    Toast.LENGTH_LONG).show();
            finish();
            return;
        }

        // check to see if storage is available to read/write
        if (Environment.MEDIA_MOUNTED.equals(state)) {
            // We can read and write the media
            mExternalStorageAvailable = mExternalStorageWriteable = true;
```

```
        } else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
            // We can only read the media
            mExternalStorageAvailable = true;
            mExternalStorageWriteable = false;
        } else {
            // Something else is wrong. It may be one of many other states,
            but
            // all we need
            // to know is we can neither read nor write
            mExternalStorageAvailable = mExternalStorageWriteable = false;
        }

        // Set up filestreams..
        try {
            file = new File(Environment.getExternalStorageDirectory()
                    + "/project/data.txt");
            filereader = new FileReader(file);
            bufferedreader_ECG1 = new BufferedReader(filereader);
        } catch (FileNotFoundException e) {
            e.printStackTrace();

        }

    }

    /**
     * /** Next up the emergency call
     *
     */
    // create an AlertDialog with side-by-side buttons
    public void onClick(final View v) {
        int[] array = { test };
        if (v == button_call) {
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setMessage("Are you sure you want to do an emergency
            call?")
                    .setCancelable(false)
                    .setPositiveButton("Yes",
                            new DialogInterface.OnClickListener() {

                                public void onClick(DialogInterface dialog,
                                        int id) {
                                    p = preferences
                                            .getString("phonenumber", "");
                                    if (p.equals("")) {
                                        Toast toast = Toast.makeText(
                                                BluetoothMonitoring.this,
                                                "Please enter a phone
                                                number",
                                                Toast.LENGTH_LONG);
                                        toast.show();
                                        Intent i = new Intent(v.getContext
                                        (),
                                                Configuration.class);
                                        startActivity(i);
                                    }
                                    emergencyCall();
                                }
                            })
                    .setNegativeButton("No",
                            new DialogInterface.OnClickListener() {
                                public void onClick(DialogInterface dialog,
                                        int id) {
                                    dialog.cancel();
                                }
                            });
            builder.show();
```

```
        }

        if (v == button_nextview) {
            Intent i = new Intent(this, DataToDrawECG.class);
            Bundle e = new Bundle();
            e.putString("key1", s);
            e.putIntArray("key2", array);
            i = i.putExtras(e);
            startActivity(i);
        }

        if (v == button_resp_go) {
            Intent i = new Intent(this, DataToDrawRespiration.class);
            Bundle e = new Bundle();
            e.putString("key1", s);
            e.putIntArray("key2", array);
            i = i.putExtras(e);
            startActivity(i);
        }

        if (v == button_acc_go) {
            Intent i = new Intent(this, DataToDrawAccelerometer.class);
            Bundle e = new Bundle();
            e.putString("key1", s);
            e.putIntArray("key2", array);
            i = i.putExtras(e);
            startActivity(i);
        }
        if (v == button_ecg_go) {
            Intent i = new Intent(this, DataToDrawECG.class);
            Bundle e = new Bundle();
            e.putString("key1", s);
            e.putIntArray("key2", array);
            i = i.putExtras(e);
            startActivity(i);
        }



    }

    // action call
    public void emergencyCall() {

        try {
            Intent intent = new Intent();
            intent.setAction(Intent.ACTION_CALL);
            intent.setData(Uri.parse("tel:" + p));
            startActivity(intent);

        } catch (ActivityNotFoundException e) {
            Log.e("Error: ",
                    "failed to make a call. Please check Call settings in
                    Configuration menu",
                    e);
        }
    }

    @Override
    public void onStart() {
        super.onStart();
        if (D)
            Log.e(TAG, "++ ON START ++");

        myReceiver = new MyReceiver();
        IntentFilter intentFilter = new IntentFilter();
        intentFilter.addAction(BluetoothRfcommService.MY_ACTION);
        registerReceiver(myReceiver, intentFilter);
```

```
        // If BT is not on, request that it be enabled.
        // setupMonitoring() will then be called during onActivityResult
        if (!mBluetoothAdapter.isEnabled()) {
            Intent enableIntent = new Intent(
                    BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
            // Otherwise, setup the chat session
        } else {
            // if (mRfcommClient == null)
            if (!mBound)
                setupMonitoring();
        }
    }

    @Override
    public synchronized void onResume() {
        super.onResume();
        if (D)
            Log.e(TAG, "+ ON RESUME +");

        // Performing this check in onResume() covers the case in which BT
        was
        // not enabled during onStart(), so we were paused to enable it...
        // onResume() will be called when ACTION_REQUEST_ENABLE activity
        // returns.
        if (mRfcommClient != null) {
            // Only if the state is STATE_NONE, do we know that we haven't
            // started already
            if (mRfcommClient.getState() == BluetoothRfcommService.
            STATE_NONE) {
                // Start the Bluetooth chat services
                mRfcommClient.start();
            }
        }
    }

    private void setupMonitoring() {
        Log.d(TAG, "setupMonitoring()");

        if (preferences.getBoolean("syncEnabled", false) && !mBound) {
            // Bind to the TexTabService
            Intent intent = new Intent(this, BluetoothRfcommService.class);
            bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
        }

    }

    @Override
    public synchronized void onPause() {
        super.onPause();
        if (D)
            Log.e(TAG, "- ON PAUSE -");
    }

    @Override
    public void onStop() {
        unregisterReceiver(myReceiver);
        super.onStop();
        if (D)
            Log.e(TAG, "-- ON STOP --");
        if (mBound) {
            unbindService(mConnection);
            mBound = false;
        }
    }

    @Override
    public void onDestroy() {
```

```
        super.onDestroy();
        // Stop the Bluetooth chat services
        // if (mRfcommClient != null)
        // mRfcommClient.stop();
        if (D)
            Log.e(TAG, "--- ON DESTROY ---");
        if (mBound) {
            unbindService(mConnection);
            mBound = false;
        }

    }

    /**
     * Sends a message.
     *
     * @param message
     *            A string of text to send.
     */
    @SuppressWarnings("unused")
    private void sendMessage(String Message) {
        // Check that we're actually connected before trying anything
        if (mRfcommClient.getState() != BluetoothRfcommService.
        STATE_CONNECTED) {
            Toast.makeText(this, R.string.not_connected,
            Toast.LENGTH_SHORT)
                    .show();
            return;
        }

        // Check that there's actually something to send
        if (Message.length() > 0) {
            // Get the message bytes and tell the BluetoothRfcommService to
            // write
            byte[] send = Message.getBytes();
            mRfcommClient.write(send);

        }
    }

    private void setupUI() {


        // Initialize panels
        panel_heart_rate = (TextView) findViewById(R.id.heart_bpm);
        panel_heart_rate.setText(this.getString(R.string.test3));
        panel_respiration_rate = (TextView) findViewById(R.id.
        respiration_bpm);
        panel_respiration_rate.setText(this.getString(R.string.test2));
        panel_temperature_c = (TextView) findViewById(R.id.temperature_C);
        panel_temperature_c.setText(this.getString(R.string.test1));
        panel_bottom1 = (TextView) findViewById(R.id.testtext1);
        panel_bottom1.setText(this.getString(R.string.bottom_test1));
        panel_bottom2 = (TextView) findViewById(R.id.testtext2);
        panel_bottom2.setText(this.getString(R.string.bottom_test2));
        panel_bottom3 = (TextView) findViewById(R.id.testtext3);
        panel_bottom3.setText(this.getString(R.string.bottom_test3));
        panel_bottom4 = (TextView) findViewById(R.id.testtext4);
        panel_bottom4.setText(this.getString(R.string.bottom_test4));

        // Set up buttons
        button_nextview = (Button) findViewById(R.id.nextbutton);
        button_nextview.setOnClickListener(this);
        button_call = (Button) findViewById(R.id.button_call_emergency);
        button_call.setOnClickListener(this);
        imageButton_position = (ImageButton) findViewById(R.id.
        imageButton_body_position);
        imageButton_position.setOnClickListener(this);
```

```
        button_ind_1 = (Button) findViewById(R.id.indicatorbutton1);
        button_ind_2 = (Button) findViewById(R.id.indicatorbutton2);
        button_ind_3 = (Button) findViewById(R.id.indicatorbutton3);
        button_ind_4 = (Button) findViewById(R.id.indicatorbutton4);
        button_ind_5 = (Button) findViewById(R.id.indicatorbutton5);
        button_ind_6 = (Button) findViewById(R.id.indicatorbutton6);
        button_ind_7 = (Button) findViewById(R.id.indicatorbutton7);
        button_ind_8 = (Button) findViewById(R.id.indicatorbutton8);
        button_ind_9 = (Button) findViewById(R.id.indicatorbutton9);
        button_ind_10 = (Button) findViewById(R.id.indicatorbutton10);
        button_ind_11 = (Button) findViewById(R.id.indicatorbutton11);
        button_ind_12 = (Button) findViewById(R.id.indicatorbutton12);

        button_ecg_go = (Button) findViewById(R.id.button_ecg);
        button_ecg_go.setOnClickListener(this);
        button_resp_go = (Button) findViewById(R.id.button_resp);
        button_resp_go.setOnClickListener(this);
        button_acc_go = (Button) findViewById(R.id.button_acc);
        button_acc_go.setOnClickListener(this);

    }

    private final Handler mHandler = new Handler() {

        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {

            case MESSAGE_WRITE:

                break;
            case MESSAGE_READ:
                byte[] readBuf = (byte[]) msg.obj;
                // construct a string from the valid bytes in the buffer
                String readMessage = new String(readBuf, 0, msg.arg1);
                // String readMessage = (String) msg.obj;

                displayTemperature(readMessage);
                displayRespirationRate(readMessage);
                displayHeartRate(readMessage);
                displayPosition(readMessage);
                displayMisc(readMessage);

                // bluedatatransfer(readMessage);

                // ---add to db---
                // String phoneNumber = readMessage.substring(0, divider);
                // String message = readMessage.substring(divider + 1);
                // String time = now();
                // db.open();
                // long id;
                // id = db.insertSms(time, message, phoneNumber,
                // DBAdapter.RECEIVED);
                // db.close();
                break;
            case MESSAGE_DEVICE_NAME:
                // save the connected device's name
                mConnectedDeviceName =
                msg.getData().getString(DEVICE_NAME);
                Toast.makeText(getApplicationContext(),
                        "Connected to " + mConnectedDeviceName,
                        Toast.LENGTH_SHORT).show();
                break;
            case MESSAGE_TOAST:
                Toast.makeText(getApplicationContext(),
                        msg.getData().getString(TOAST), Toast.LENGTH_SHORT)
                        .show();
```

```
                break;
            case MESSAGE_GET_DATA:
                test = msg.arg1;
                break;

            }
        }

        // private String bluedatatransfer(String readMessage) {
        // bluedata = readMessage;
        // return bluedata;
        // }
    };

    private void displayPosition(String Message) {

        for (int i = 0; i < Message.length() - 3; i++) {
            if (Message.charAt(i) == 'n') {
                char msg1 = Message.charAt(i + 1);
                char msg2 = Message.charAt(i + 2);

                byte bmsg1 = (byte) (msg1 - 0x20);
                byte bmsg2 = (byte) (msg2 - 0x20);

                int sensorValue = (0x000000ff & bmsg1)
                        | ((0x000000ff & bmsg2) << 6);
                positionTransfer(sensorValue);
                switch (sensorValue) {
                case 18:// upright
                    //
                    setCompoundDrawablesWithIntrinsicBounds(orientation_upr
                    ight_img,
                    // null, null, null);
                    imageButton_position
                            .setImageResource(R.drawable.position_upright);
                    break;
                case 19:// prone
                    imageButton_position
                            .setImageResource(R.drawable.position_prone);
                    break;
                case 20:// supine
                    imageButton_position
                            .setImageResource(R.drawable.position_supine);
                    break;
                case 21:// inverted
                    imageButton_position

                            .setImageResource(R.drawable.position_inverted)
                            ;
                    break;
                case 22:// side
                    imageButton_position
                            .setImageResource(R.drawable.position_side);
                    break;
                case 23:// unknown
                    //
                    imageButton_position.setImageResource(R.drawable.positi
                    on_unknown);
                    break;
                case 152:// falling
                    imageButton_position
                            .setImageResource(R.drawable.position_falling);
                    break;
                case 165:// moving (slowly)
                    imageButton_position
                            .setImageResource(R.drawable.position_moving);
                    break;
                }
```

```
            }
        }
    }

    protected void displayMisc(String readMessage) {
        // TODO Auto-generated method stub
        for (int i = 0; i < readMessage.length() - 3; i++) {
            if (readMessage.charAt(i) == 'n') {
                char msg1 = readMessage.charAt(i + 1);
                char msg2 = readMessage.charAt(i + 2);
                byte bmsg1 = (byte) (msg1 - 0x20);
                byte bmsg2 = (byte) (msg2 - 0x20);
                int sensorValue = (0x000000ff & bmsg1)
                        | ((0x000000ff & bmsg2) << 6);
                switch (sensorValue) {
                case 14:// battery low
                    panel_bottom1.setBackgroundDrawable(getResources()
                            .getDrawable(
                                    R.drawable.
                                    frame_light_black_outline_red));
                    break;
                case 15:// battery ok
                    panel_bottom1.setBackgroundDrawable(getResources()
                            .getDrawable(R.drawable.
                            frame_light_black_outline));
                    break;
                case 24:// Apnea
                    panel_bottom2.setBackgroundDrawable(getResources()
                            .getDrawable(
                                    R.drawable.
                                    frame_light_black_outline_red));
                    break;
                // case 25:// vital sign green
                // panel_bottom3.setBackgroundDrawable(getResources()
                // .getDrawable(R.drawable.frame_light_black_outline));
                // break;
                case 27:// vital sign red
                    panel_bottom3.setBackgroundDrawable(getResources()
                            .getDrawable(
                                    R.drawable.
                                    frame_light_black_outline_red));
                    break;
                case 183:// irregular rythm
                    panel_bottom4.setBackgroundDrawable(getResources()
                            .getDrawable(
                                    R.drawable.
                                    frame_light_black_outline_red));
                    break;
                }
            }
        }
    }

    private String positionTransfer(int sensorValue) {
        // TODO Auto-generated method stub
        position = String.valueOf(sensorValue);
        return position;

    }

    private void displayHeartRate(String Message) {

        for (int i = 0; i < Message.length() - 3; i++) {
            if (Message.charAt(i) == 'h') {
                char msg1 = Message.charAt(i + 1);
                char msg2 = Message.charAt(i + 2);

                byte bmsg1 = (byte) (msg1 - 0x20);
```

```
                byte bmsg2 = (byte) (msg2 - 0x20);

                double sensorValue = (0x000000ff & bmsg1)
                        | ((0x000000ff & bmsg2) << 6);
                double formattedValue = (sensorValue * 0.1);
                int endValue = (int) Math.round(formattedValue);

                heartRateTransfer(endValue);

                panel_heart_rate.setText("HR(bpm):" + endValue);

                if (endValue > maximumHR) {
                    panel_heart_rate.setBackgroundDrawable(getResources()
                            .getDrawable(
                                    R.drawable.
                                    frame_light_black_outline_red));
                    button_ind_1.setBackgroundDrawable(getResources()
                            .getDrawable(R.drawable.button_green));
                    button_ind_2.setBackgroundDrawable(getResources()
                            .getDrawable(R.drawable.button_green));
                    button_ind_3.setBackgroundDrawable(getResources()
                            .getDrawable(R.drawable.button_yellow));
                    button_ind_4.setBackgroundDrawable(getResources()
                            .getDrawable(R.drawable.button_red));
                } else {
                    if (endValue < minimumHR) {
                        panel_heart_rate
                                .setBackgroundDrawable(getResources()
                                        .getDrawable(
                                                R.drawable.
                                                frame_light_black_outline_r
                                                ed));
                        button_ind_1.setBackgroundDrawable(getResources()
                                .getDrawable(R.drawable.button_red));
                    } else {
                        if (endValue > 120) {
                            button_ind_1.setBackgroundDrawable(getResources
                            ()
                                    .getDrawable(R.drawable.button_green));
                            button_ind_2.setBackgroundDrawable(getResources
                            ()
                                    .getDrawable(R.drawable.button_green));
                            button_ind_3.setBackgroundDrawable(getResources
                            ()

                                    .getDrawable(R.drawable.button_yellow))
                                    ;
                        } else {
                            panel_heart_rate
                                    .setBackgroundDrawable(getResources()
                                            .getDrawable(
                                                    R.drawable.
                                                    frame_light_black_outli
                                                    ne));
                            button_ind_1.setBackgroundDrawable(getResources
                            ()
                                    .getDrawable(R.drawable.button_green));
                            button_ind_2.setBackgroundDrawable(getResources
                            ()
                                    .getDrawable(R.drawable.button_green));
                        }
                    }
                }
            }
        }
    }

    private String heartRateTransfer(int endValue) {
```

```
        // TODO Auto-generated method stub
        heartRate = String.valueOf(endValue);
        return heartRate;
    }

    private void displayRespirationRate(String Message) {

        for (int i = 0; i < Message.length() - 3; i++) {
            if (Message.charAt(i) == 'u') {
                char msg1 = Message.charAt(i + 1);
                char msg2 = Message.charAt(i + 2);

                byte bmsg1 = (byte) (msg1 - 0x20);
                byte bmsg2 = (byte) (msg2 - 0x20);

                double sensorValue = (0x000000ff & bmsg1)
                        | ((0x000000ff & bmsg2) << 6);
                double formattedValue = (sensorValue * 0.1);
                int endValue = (int) Math.round(formattedValue);

                respirationRateTransfer(endValue);

                panel_respiration_rate.setText("BR(bpm):" + endValue);

                if (endValue > maximumBR) {
                    panel_respiration_rate.setBackgroundDrawable(
                    getResources()
                            .getDrawable(
                                    R.drawable.
                                    frame_light_black_outline_red));
                    button_ind_5.setBackgroundDrawable(getResources()
                            .getDrawable(R.drawable.button_green));
                    button_ind_6.setBackgroundDrawable(getResources()
                            .getDrawable(R.drawable.button_green));
                    button_ind_7.setBackgroundDrawable(getResources()
                            .getDrawable(R.drawable.button_yellow));
                    button_ind_8.setBackgroundDrawable(getResources()
                            .getDrawable(R.drawable.button_red));
                } else {
                    if (endValue < minimumBR) {
                        panel_respiration_rate
                                .setBackgroundDrawable(getResources()
                                        .getDrawable(
                                                R.drawable.
                                                frame_light_black_outline_r
                                                ed));
                        button_ind_5.setBackgroundDrawable(getResources()
                                .getDrawable(R.drawable.button_green));
                    } else {
                        if (endValue > 30) {
                            button_ind_5.setBackgroundDrawable(getResources
                            ()
                                    .getDrawable(R.drawable.button_green));
                            button_ind_6.setBackgroundDrawable(getResources
                            ()
                                    .getDrawable(R.drawable.button_green));
                            button_ind_7.setBackgroundDrawable(getResources
                            ()

                                    .getDrawable(R.drawable.button_yellow))
                                    ;
                        } else {
                            panel_respiration_rate
                                    .setBackgroundDrawable(getResources()
                                            .getDrawable(
                                                    R.drawable.
                                                    frame_light_black_outli
                                                    ne));
```

```
                            button_ind_5.setBackgroundDrawable(getResources
                            ()
                                    .getDrawable(R.drawable.button_green));
                            button_ind_6.setBackgroundDrawable(getResources
                            ()
                                    .getDrawable(R.drawable.button_green));
                        }
                    }
                }
            }
        }

    private String respirationRateTransfer(int endValue) {
        // TODO Auto-generated method stub
        respirationRate = String.valueOf(endValue);
        return respirationRate;

    }

    private void displayTemperature(String Message) {
        for (int i = 0; i < Message.length() - 3; i++) {
            if (Message.charAt(i) == 't') {
                char msg1 = Message.charAt(i + 1);
                char msg2 = Message.charAt(i + 2);

                byte bmsg1 = (byte) (msg1 - 0x20);
                byte bmsg2 = (byte) (msg2 - 0x20);

                double sensorValue = (0x000000ff & bmsg1)
                        | ((0x000000ff & bmsg2) << 6);
                double formattedValue = (sensorValue * 0.1);
                int endValue = (int) Math.round(formattedValue);

                temperatureTransfer(endValue);

                panel_temperature_c.setText("Temp(C°):" + endValue);

                if (endValue > maximumT) {
                    panel_temperature_c.setBackgroundDrawable(getResources
                    ()
                            .getDrawable(
                                    R.drawable.
                                    frame_light_black_outline_red));
                    button_ind_9.setBackgroundDrawable(getResources()
                            .getDrawable(R.drawable.button_green));
                    button_ind_10.setBackgroundDrawable(getResources()
                            .getDrawable(R.drawable.button_green));
                    button_ind_11.setBackgroundDrawable(getResources()
                            .getDrawable(R.drawable.button_yellow));
                    button_ind_12.setBackgroundDrawable(getResources()
                            .getDrawable(R.drawable.button_red));
                } else {
                    if (endValue < minimumT) {
                        panel_respiration_rate
                                .setBackgroundDrawable(getResources()
                                        .getDrawable(
                                                R.drawable.
                                                frame_light_black_outline_r
                                                ed));
                        button_ind_9.setBackgroundDrawable(getResources()
                                .getDrawable(R.drawable.button_green));
                    } else {
                        if (endValue > 30) {
                            button_ind_9.setBackgroundDrawable(getResources
                            ()
                                    .getDrawable(R.drawable.button_green));
                            button_ind_10.setBackgroundDrawable(
```

```
                            getResources()
                                    .getDrawable(R.drawable.button_green));
                            button_ind_11.setBackgroundDrawable(
                            getResources()

                                    .getDrawable(R.drawable.button_yellow))
                                    ;
                        } else {
                            panel_respiration_rate
                                    .setBackgroundDrawable(getResources()
                                            .getDrawable(
                                                    R.drawable.
                                                    frame_light_black_outli
                                                    ne));
                            button_ind_9.setBackgroundDrawable(getResources
                            ()
                                    .getDrawable(R.drawable.button_green));
                            button_ind_10.setBackgroundDrawable(
                            getResources()
                                    .getDrawable(R.drawable.button_green));
                        }
                    }
                }

            }
        }
    }

    private String temperatureTransfer(int endValue) {
        // TODO Auto-generated method stub
        temperature = String.valueOf(endValue);
        return temperature;

    }

    public void onActivityResult(int requestCode, int resultCode, Intent
    data) {
        if (D)
            Log.d(TAG, "onActivityResult " + resultCode);
        switch (requestCode) {
        case REQUEST_CONNECT_DEVICE:
            // When DeviceListActivity returns with a device to connect
            if (resultCode == Activity.RESULT_OK) {
                // Get the device MAC address
                String address = data.getExtras().getString(
                        DeviceListActivity.EXTRA_DEVICE_ADDRESS);
                // Get the BLuetoothDevice object
                BluetoothDevice device = mBluetoothAdapter
                        .getRemoteDevice(address);
                // Attempt to connect to the device
                mRfcommClient.connect(device);
            }
            break;
        case REQUEST_ENABLE_BT:
            // When the request to enable Bluetooth returns
            if (resultCode == Activity.RESULT_OK) {
                // Bluetooth is now enabled, so set up a chat session
                setupMonitoring();
            } else {
                // User did not enable Bluetooth or an error occured
                Log.d(TAG, "BT not enabled");
                Toast.makeText(this, R.string.bt_not_enabled_leaving,
                        Toast.LENGTH_SHORT).show();
                finish();
            }
        }
    }
```

```
    public void onSharedPreferenceChanged(SharedPreferences prefs, String
    key) {
        if (key.equals("syncEnabled")) {
            if (!prefs.getBoolean("syncEnabled", false) && mBound) {
                Log.d(TAG,
                        "Preferences changed, BluetoothRfcommService should
                        now unbind and stop");

                killBluetoothChatService();
            }

            if (prefs.getBoolean("syncEnabled", false) && !mBound) {
                startBluetoothChatService();
            }
        }
    }

    private void startBluetoothChatService() {
        if (preferences.getBoolean("syncEnabled", false)) {

            Intent intent = new Intent(this, BluetoothRfcommService.class);
            startService(intent);
            setupMonitoring();
        }
    }

    private void killBluetoothChatService() {
        // Unbind from the BluetoothRfcommService
        // mChatService.stop();
        unbindService(mConnection);
        mBound = false;

        // Stop the BluetoothRfcommService
        Intent intent = new Intent(this, BluetoothRfcommService.class);
        stopService(intent);

    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.option_menu, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        case R.id.scan:
            if (mBound) {
                // Launch the DeviceListActivity to see devices and do scan
                Intent serverIntent = new Intent(this, DeviceListActivity.
                class);
                startActivityForResult(serverIntent,
                REQUEST_CONNECT_DEVICE);
            } else
                Toast.makeText(getBaseContext(),
                        "BluetoothRfcommService is not enabled!",
                        Toast.LENGTH_LONG).show();
            return true;
        case R.id.store:
            // Configurations menu
            // Intent storage = new Intent(this, Storage.class);
            // startActivity(storage);
            boolean didItWork = true;
            String b = "nirgends";

            try {
```

```
                for (int i = 0; i < bluedata.length() - 3; i++) {
                    DatabaseHelperClass DHC = new DatabaseHelperClass(
                            BluetoothMonitoring.this);
                    String dateTime = new SimpleDateFormat("yyyy-MM-dd
                    HH:mmZ")
                            .format(new Date());
                    // DHC.open();
                    // DHC.createEntry(position, heartRate,
                    respirationRate,
                    // temperature, dateTime);
                    // DHC.close();
                    b = BluetoothMonitoring.this.getDatabasePath(
                    "EquivitalDb")
                            .getPath();
                }
                // return true;
            } catch (Exception e) {
                didItWork = false;
                String error = e.toString();
                Dialog d = new Dialog(this);
                d.setTitle("Dang it!");
                TextView tv = new TextView(this);
                tv.setText(error);
                d.setContentView(tv);
                d.show();
            } finally {
                if (didItWork) {
                    Dialog d = new Dialog(this);
                    d.setTitle("Storage!");
                    TextView tv = new TextView(this);
                    tv.setText("Beginn to save, DB stored in" + b);
                    d.setContentView(tv);
                    d.show();
                }

            }
            return true;

        }
        return false;
    }

    /**
     * Defines callbacks for service binding, passed to bindService()
     */
    private ServiceConnection mConnection = new ServiceConnection() {
        public void onServiceConnected(ComponentName className, IBinder
        service) {
            // We've bound to BluetoothRfcommService, cast the IBinder and
            get
            // BluetoothRfcommService instance6
            LocalBinder binder = (LocalBinder) service;
            mRfcommClient = binder.getService(mHandler);
            mBound = true;
            // Toast.makeText(getBaseContext(), "BluetoothRfcommService
            Bound",
            // Toast.LENGTH_LONG).show();
            if ((preferences.getString("autoconnect_device", null) != null)
                    && (preferences.getBoolean("autoconnect_enabled",
                    false))
                    && (mRfcommClient.getState() != BluetoothRfcommService.
                    STATE_CONNECTED)) {
                // autoconnect to the device specified in preferences
                String macAddress = preferences.getString(
                "autoconnect_device",
                        null);
                // Get the BLuetoothDevice object
                BluetoothDevice device = mBluetoothAdapter
```

```
                        .getRemoteDevice(macAddress);
                // Attempt to connect to the device
                mRfcommClient.connect(device);
            }
        }

        public void onServiceDisconnected(ComponentName argo) {
            mBound = false;
        }
    };

    private class MyReceiver extends BroadcastReceiver {

        @Override
        public void onReceive(Context arg0, Intent arg1) {
            // TODO Auto-generated method stub
            // String datapassed = arg1.getStringExtra("DATAPASSED");
            // als alternative zum massenger

        }

    }

}
```

```
/*
 * Copyright (C) 2009 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package com.project.bluetooth.monitoring;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.util.UUID;

import android.app.Service;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothServerSocket;
import android.bluetooth.BluetoothSocket;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Binder;
import android.os.Bundle;
import android.os.Environment;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.preference.PreferenceManager;
import android.util.Log;

/**
 * This class does all the work for setting up and managing Bluetooth
 * connections with other devices. It has a thread that listens for
 incoming
 * connections, a thread for connecting with a device, and a thread for
 * performing data transmissions when connected.
 *
 */
public class BluetoothRfcommService extends Service {
    // Debugging
    private static final String TAG = "BluetoothRfcommService";
    private static final boolean D = true;
    File archiv = new File(Environment.getExternalStorageDirectory()
            + "/project/data.txt");

    // private NotificationManager mNM;
    // Unique Identification Number for the Notification.
    // We use it on Notification start, and to cancel it.
    // private int NOTIFICATION = R.string.local_service_started;

    // This is the object that receives interactions from clients.
    private final IBinder mBinder = new LocalBinder();

    // Name for the SDP record when creating server socket
    private static final String NAME = "BluetoothMonitoring";
```

```
    // Unique UUID for this application
    private static final UUID SPP_UUID = UUID
            .fromString("00001101-0000-1000-8000-00805f9b34fb");

    // Member fields
    private BluetoothAdapter mAdapter;
    private Handler mHandler;
    private AcceptThread mAcceptThread;
    private ConnectThread mConnectThread;
    private ConnectedThread mConnectedThread;
    private int mState;

    // Constants that indicate the current connection state
    public static final int STATE_NONE = 0; // we're doing nothing
    public static final int STATE_LISTEN = 1; // now listening for incoming
    // connections
    public static final int STATE_CONNECTING = 2; // now initiating an
    outgoing
                                                  // connection
    public static final int STATE_CONNECTED = 3; // now connected to a
    remote
                                                  // device
    // Stored Preferences
    private SharedPreferences preferences = null;

    final static String MY_ACTION = "MY_ACTION";

    public int OnStartCommand(Intent intent, int flags, int startId) {

        if (preferences == null)
            preferences = PreferenceManager
                    .getDefaultSharedPreferences(getBaseContext());
        // Toast.makeText(this, "Rfcomm Service started",
        Toast.LENGTH_LONG)
        // .show();
        Log.d(TAG, "onStartCommand");
        // We want this service to continue running until it is explicitly
        // stopped, so return sticky.
        // start();
        return START_STICKY;

    }

    public void OnDestroy() {

        // Toast.makeText(this, "Service Process destroyed",
        Toast.LENGTH_LONG)
        // .show();
        Log.d(TAG, "onDestroy");
        stop();
    }

    /**
     * Class for clients to access. Because we know this service always
     runs in
     * the same process as its clients, we don't need to deal with IPC.
     */
    public class LocalBinder extends Binder {
        BluetoothRfcommService getService(Handler handler) {
            if (preferences == null)
                preferences = PreferenceManager
                        .getDefaultSharedPreferences(getBaseContext());
            if (mState != STATE_CONNECTED) {
                mAdapter = BluetoothAdapter.getDefaultAdapter();
                mState = STATE_NONE;
                mHandler = handler;
                if (preferences.getBoolean("syncEnabled", false))
                    BluetoothRfcommService.this.start();
```

```
            }
            // Return this instance of TexTabService so clients can call
            public
            // methods
            return BluetoothRfcommService.this;
        }
    }

    @Override
    public IBinder onBind(Intent intent) {
        // TODO Auto-generated method stub
        return mBinder;
        // return null;
    }

    // public void OnCreate() {
    //
    // // mNM =
    (NotificationManager)getSystemService(NOTIFICATION_SERVICE);
    // // showNotification();
    //
    // Toast.makeText(this, "Rfcomm Service Created", Toast.LENGTH_LONG)
    // .show();
    // Log.d(TAG, "onCreate");
    // // Display a notification about us starting. We put an icon in the
    // // status bar.
    //
    // }

    /**
     * Show a notification while this service is running.
     */
    // private void showNotification() {
    // // In this sample, we'll use the same text for the ticker and the
    // // expanded notification
    // CharSequence text = getText(R.string.local_service_started);
    //
    // // Set the icon, scrolling text and timestamp
    // Notification notification = new Notification(R.drawable.help, text,
    // System.currentTimeMillis());
    //
    // // The PendingIntent to launch our activity if the user selects this
    // // notification
    // PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
    // new Intent(this, LocalServiceActivities.Controller.class), 0);
    //
    // // Set the info for the views that show in the notification panel.
    // notification.setLatestEventInfo(this,
    // getText(R.string.local_service_label), text, contentIntent);
    //
    // // Send the notification.
    // mNM.notify(NOTIFICATION, notification);
    // }

    /**
     * Constructor. Prepares a new BluetoothChat session.
     *
     * @param context
     *            The UI Activity Context
     * @param handler
     *            A Handler to send messages back to the UI Activity
     */
    // public BluetoothRfcommService(Context context, Handler handler) {
    // mAdapter = BluetoothAdapter.getDefaultAdapter();
    // mState = STATE_NONE;
    // mHandler = handler;
    // }
```

```
    /**
     * Set the current state of the chat connection
     *
     * @param state
     *            An integer defining the current connection state
     */
    private synchronized void setState(int state) {
        if (D)
            Log.d(TAG, "setState() " + mState + " -> " + state);
        mState = state;

        // Give the new state to the Handler so the UI Activity can update
        mHandler.obtainMessage(BluetoothMonitoring.MESSAGE_STATE_CHANGE,
        state,
                -1).sendToTarget();
    }

    /**
     * Return the current connection state.
     */
    public synchronized int getState() {
        return mState;
    }

    /**
     * Start the chat service. Specifically start AcceptThread to begin a
     * session in listening (server) mode. Called by the Activity
     onResume()
     */
    public synchronized void start() {
        if (D)
            Log.d(TAG, "start");

        // Cancel any thread attempting to make a connection
        if (mConnectThread != null) {
            mConnectThread.cancel();
            mConnectThread = null;
        }

        // Cancel any thread currently running a connection
        if (mConnectedThread != null) {
            mConnectedThread.cancel();
            mConnectedThread = null;
        }

        BluetoothAdapter mBluetoothAdapter = BluetoothAdapter
                .getDefaultAdapter();

        if (mBluetoothAdapter.isEnabled()) {
            // Start the thread to listen on a BluetoothServerSocket
            if (mAcceptThread == null) {
                mAcceptThread = new AcceptThread();
                mAcceptThread.start();
            }
            setState(STATE_LISTEN);
        } else {
            // Send a failure message back to the Activity
            Message msg = mHandler
                    .obtainMessage(BluetoothMonitoring.MESSAGE_TOAST);
            Bundle bundle = new Bundle();
            bundle.putString(BluetoothMonitoring.TOAST,
                    "Bluetooth is not enabled. Please re-enabled bluetooth
                    to continue.");
            msg.setData(bundle);
            mHandler.sendMessage(msg);

        }
    }
```

```
    /**
     * Start the ConnectThread to initiate a connection to a remote device.
     *
     * @param device
     *            The BluetoothDevice to connect
     */
    public synchronized void connect(BluetoothDevice device) {
        if (D)
            Log.d(TAG, "connect to: " + device);

        // Cancel any thread attempting to make a connection
        if (mState == STATE_CONNECTING) {
            if (mConnectThread != null) {
                mConnectThread.cancel();
                mConnectThread = null;
            }
        }

        // Cancel any thread currently running a connection
        if (mConnectedThread != null) {
            mConnectedThread.cancel();
            mConnectedThread = null;
        }

        // Start the thread to connect with the given device
        mConnectThread = new ConnectThread(device);
        mConnectThread.start();
        setState(STATE_CONNECTING);
    }

    /**
     * Start the ConnectedThread to begin managing a Bluetooth connection
     *
     * @param socket
     *            The BluetoothSocket on which the connection was made
     * @param device
     *            The BluetoothDevice that has been connected
     */
    public synchronized void connected(BluetoothSocket socket,
            BluetoothDevice device) {
        if (D)
            Log.d(TAG, "connected");

        // Cancel the thread that completed the connection
        if (mConnectThread != null) {
            mConnectThread.cancel();
            mConnectThread = null;
        }

        // Cancel any thread currently running a connection
        if (mConnectedThread != null) {
            mConnectedThread.cancel();
            mConnectedThread = null;
        }

        // Cancel the accept thread because we only want to connect to one
        // device
        if (mAcceptThread != null) {
            mAcceptThread.cancel();
            mAcceptThread = null;
        }

        // Start the thread to manage the connection and perform
        transmissions
        mConnectedThread = new ConnectedThread(socket);
        mConnectedThread.start();
```

```
        // Send the name of the connected device back to the UI Activity
        Message msg = mHandler
                .obtainMessage(BluetoothMonitoring.MESSAGE_DEVICE_NAME);
        Bundle bundle = new Bundle();
        bundle.putString(BluetoothMonitoring.DEVICE_NAME,
        device.getName());
        msg.setData(bundle);
        mHandler.sendMessage(msg);

        setState(STATE_CONNECTED);
    }

    /**
     * Stop all threads
     */
    public synchronized void stop() {
        if (D)
            Log.d(TAG, "stop");
        if (mConnectThread != null) {
            mConnectThread.cancel();
            mConnectThread = null;
        }
        if (mConnectedThread != null) {
            mConnectedThread.cancel();
            mConnectedThread = null;
        }
        if (mAcceptThread != null) {
            mAcceptThread.cancel();
            mAcceptThread = null;
        }
        setState(STATE_NONE);
    }

    /**
     * Write to the ConnectedThread in an unsynchronized manner
     *
     * @param out
     *              The bytes to write
     * @see ConnectedThread#write(byte[])
     */
    public void write(byte[] out) {
        // Create temporary object
        ConnectedThread r;
        // Synchronize a copy of the ConnectedThread
        synchronized (this) {
            if (mState != STATE_CONNECTED)
                return;
            r = mConnectedThread;
        }
        // Perform the write unsynchronized
        r.write(out);
    }

    /**
     * Indicate that the connection attempt failed and notify the UI
     Activity.
     */
    private void connectionFailed() {
        setState(STATE_LISTEN);

        // Send a failure message back to the Activity
        Message msg = mHandler.obtainMessage(BluetoothMonitoring.
        MESSAGE_TOAST);
        Bundle bundle = new Bundle();
        bundle.putString(BluetoothMonitoring.TOAST, "Unable to connect
        device");
        msg.setData(bundle);
        mHandler.sendMessage(msg);
```

```
    }

    /**
     * Indicate that the connection was lost and notify the UI Activity.
     */
    private void connectionLost() {
        setState(STATE_LISTEN);

        // Send a failure message back to the Activity
        Message msg = mHandler.obtainMessage(BluetoothMonitoring.
        MESSAGE_TOAST);
        Bundle bundle = new Bundle();
        bundle.putString(BluetoothMonitoring.TOAST,
                "Device connection was lost");
        msg.setData(bundle);
        mHandler.sendMessage(msg);

    }

    /**
     * This thread runs while listening for incoming connections. It
     behaves
     * like a server-side client. It runs until a connection is accepted
     (or
     * until cancelled).
     *
     * not sure this part can completely be removed
     */
    private class AcceptThread extends Thread {
        // The local server socket
        private final BluetoothServerSocket mmServerSocket;

        //
        public AcceptThread() {

            BluetoothServerSocket tmp = null;
            //
            // Create a new listening server socket
            try {
                //
                tmp = mAdapter.listenUsingRfcommWithServiceRecord(NAME,
                        SPP_UUID);
            } catch (IOException e) {
                Log.e(TAG, "listen() failed", e);
            }
            mmServerSocket = tmp;
        }

        //
        public void run() {
            if (D)
                Log.d(TAG, "BEGIN mAcceptThread" + this);
            setName("AcceptThread");
            BluetoothSocket socket = null;

            // Listen to the server socket if we're not connected
            while (mState != STATE_CONNECTED) {
                try {
                    // This is a blocking call and will only return on a
                    // successful connection or an exception
                    socket = mmServerSocket.accept();
                } catch (IOException e) {
                    Log.e(TAG, "accept() failed", e);
                    break;
                }

                // If a connection was accepted
```

```
                if (socket != null) {
                    synchronized (BluetoothRfcommService.this) {
                        switch (mState) {
                        case STATE_LISTEN:
                        case STATE_CONNECTING:
                            // Situation normal. Start the connected
                            thread.
                            connected(socket, socket.getRemoteDevice());
                            break;
                        case STATE_NONE:
                        case STATE_CONNECTED:
                            // Either not ready or already connected.
                            // Terminate
                            // new socket.
                            try {
                                socket.close();
                            } catch (IOException e) {
                                Log.e(TAG, "Could not close unwanted
                                socket", e);
                            }
                            break;
                        }
                    }
                }
            }
            if (D)
                Log.i(TAG, "END mAcceptThread");
        }

        public void cancel() {
            if (D)
                Log.d(TAG, "cancel " + this);
            try {
                mmServerSocket.close();
            } catch (IOException e) {
                Log.e(TAG, "close() of server failed", e);
            }
        }
    }

    /**
     * This thread runs while attempting to make an outgoing connection
     with a
     * device. It runs straight through; the connection either succeeds or
     * fails.
     */
    private class ConnectThread extends Thread {
        private final BluetoothSocket mmSocket;
        private final BluetoothDevice mmDevice;

        public ConnectThread(BluetoothDevice device) {
            mmDevice = device;
            BluetoothSocket tmp = null;

            // Get a BluetoothSocket for a connection with the
            // given BluetoothDevice
            try {

                tmp = device.createRfcommSocketToServiceRecord(SPP_UUID);

            } catch (IOException e) {
                Log.e(TAG, "create() failed", e);
            }
            mmSocket = tmp;
        }

        public void run() {
            Log.i(TAG, "BEGIN mConnectThread");
```

```
            setName("ConnectThread");

            // Always cancel discovery because it will slow down a
            connection
            mAdapter.cancelDiscovery();

            // Make a connection to the BluetoothSocket
            try {

                mmSocket.connect();
            } catch (IOException e) {
                connectionFailed();
                // Close the socket
                try {
                    mmSocket.close();
                } catch (IOException e2) {
                    Log.e(TAG,
                            "unable to close() socket during connection
                            failure",
                            e2);
                }

                // Start the service over to restart listening mode
                BluetoothRfcommService.this.start();
                return;
            }

            // Reset the ConnectThread because we're done
            synchronized (BluetoothRfcommService.this) {
                mConnectThread = null;
            }

            // Start the connected thread
            connected(mmSocket, mmDevice);
        }

        public void cancel() {
            try {
                mmSocket.close();
            } catch (IOException e) {
                Log.e(TAG, "close() of connect socket failed", e);
            }
        }
    }

    /**
     * This thread runs during a connection with a remote device. It
     handles all
     * incoming and outgoing transmissions.
     */
    private class ConnectedThread extends Thread {
        private final BluetoothSocket mmSocket;
        private final InputStream mmInStream;
        private final OutputStream mmOutStream;

        // private IncomingBuffer ringBuffer;
        // private IncomingBuffer RingBuffer;

        public ConnectedThread(BluetoothSocket socket) {
            Log.d(TAG, "create ConnectedThread");
            mmSocket = socket;
            InputStream tmpIn = null;
            OutputStream tmpOut = null;

            // Get the BluetoothSocket input and output streams
            try {
                tmpIn = socket.getInputStream();
                tmpOut = socket.getOutputStream();
```

```
            } catch (IOException e) {
                Log.e(TAG, "temp sockets not created", e);
            }

            mmInStream = tmpIn;
            mmOutStream = tmpOut;
        }

        public void run() {
            Log.i(TAG, "BEGIN mConnectedThread");
            byte[] buffer = new byte[1024];
            int bytes = 0;
            int bytestest = 0;
            String testtest = "0";

            // Keep listening to the InputStream while connected
            while (true) {
                try {
                    // Read from the InputStream
                    bytes = mmInStream.read(buffer);
                    bytestest = bytestest + bytes;

//                  String readMessage = new String(buffer, 0, bytes);
//                  Intent intent = new Intent();
//                     intent.setAction(MY_ACTION);
//                     intent.putExtra("DATAPASSED", readMessage);
//                     sendBroadcast(intent);
                    // Send the obtained bytes to the UI Activity
                    //

                    mHandler.obtainMessage(BluetoothMonitoring.MESSAGE_READ
                    ,
                            bytes, -1, buffer).sendToTarget();
                    mHandler.obtainMessage(
                            BluetoothMonitoring.MESSAGE_GET_DATA,
                            bytestest,
                            -1, null).sendToTarget();
                    String testrest = new String(buffer, 0, bytes);
                    testtest = testtest + testrest;
                    if (testtest.length() >= 80) {

//                      mHandler.obtainMessage(
//                              BluetoothMonitoring.MESSAGE_GET_STRING,
bytes,
//                              -1, testtest).sendToTarget();
                        // put testtest into some buffer or cache
                        // fifo queue linkedlist linkedhashmap etc and
                        access it
                        // from
                        // another activity
                         save(testtest);
                        bytestest = 0;
                        testtest = "";

                    }

                } catch (IOException e) {
                    Log.e(TAG, "disconnected", e);
                    connectionLost();
                    // Start the service over to restart listening mode
                    // BluetoothRfcommService.this.start();
                    break;
                }
            }
        }

        private void save(String testtest) {
            // TODO Auto-generated method stub
```

```
            try {

                FileOutputStream fOut = new FileOutputStream(archiv, true);
                OutputStreamWriter osw = new OutputStreamWriter(fOut);
                osw.write(testtest);
                osw.flush();

            } catch (IOException ioe2) {
                Log.i("error en el 2", "" + ioe2);
                connectionLost();
            } catch (Exception e) {
                Log.i("error in archiv", "" + e);
                e.printStackTrace();
                connectionLost();
            }

        }

        /**
         * Write to the connected OutStream.
         *
         * @param buffer
         *            The bytes to write
         */
        public void write(byte[] buffer) {
            try {
                mmOutStream.write(buffer);

                // Share the sent message back to the UI Activity
                mHandler.obtainMessage(BluetoothMonitoring.MESSAGE_WRITE, -
                1,
                        -1, buffer).sendToTarget();
            } catch (IOException e) {
                Log.e(TAG, "Exception during write", e);
            }
        }

        public void cancel() {
            try {
                mmSocket.close();
            } catch (IOException e) {
                Log.e(TAG, "close() of connect socket failed", e);
            }
        }
    }

}
```


Figure 2: Monitoring classes

References

[1] WHO. *Cardiovascular diseases (CVDs)*. September 2012. URL http://www.who.int/mediacentre/factsheets/fs317/en/index.html. iii, iv

[2] wirehead, 2012. URL http://www.wireheadtec.com/images/healthcare.jpg. vii, 7

[3] weltel, 2012. URL http://www.weltel.org/wp-content/gallery/weltel-gallery/WelTel?20logo2.jpg. vii, 9

[4] ATNF, 2012. URL https://twimg0-a.akamaihd.net/profile_images/2024412672/ATNF_Logo.JPG. vii, 10

[5] skinvision, 2012. URL https://skinvision.com/assets/about-01-ccfaee12868e1e86044ddc1a2a0a1d17.jpg. vii, 27

[6] vimeo, 2012. URL http://b.vimeocdn.com/ps/115/727/1157274_300.jpg. vii, 28

[7] zdnet, 2012. URL http://cdn-static.zdnet.com/i/story/61/18/011660/mobile_os_array_jan2010.jpg. vii, 29

[8] os market, 2012. URL http://www.multilingual-search.com/wp-content/uploads/2012/07/os-market-share-global1.jpg. vii, 29

[9] handy news, 2012. URL http://www.handy-news.at/wordpress/wp-content/uploads/2012/05/nokia-symbian.jpg. vii, 30

[10] developer nokia, 2012. URL http://library.developer.nokia.com/topic/GUID-E35887BB-7E58-438C-AA27-97B2CDE7E069/

GUID-1742B2A1-6EC1-5368-919B-362516A5D452_d0e425624_href.png. vii, 31

[11] mobility, 2012. URL http://mobilitymagazin.de/wp-content/uploads//2011/02/windows-phone-7-logo.jpg. vii, 32

[12] windowsphone, 2012. URL http://windowsphone.interoperabilitybridges.com/media/134/Windows-Live-Writer_Chapter-1-Windows-Phone-7-Platform-intro_7E5E_clip_image002_2.jpg. vii, 32

[13] rim, 2012. URL http://cdn.ne8.de/wp-content/uploads/2012/06/rim.jpg. vii, 33

[14] apple ios, 2012. URL http://techhimcdn.techhim.netdna-cdn.com/wp-content/uploads/2012/05/apple-ios_thumb.jpg. vii, 33

[15] apple, 2012. URL http://images.apple.com/business/accelerator/develop/images/gallery_architecture.jpg. vii, 34

[16] htc, 2012. URL http://www.htcinside.de/wp-content/uploads/2012/03/20120319_android_logo_01.jpg. vii, 35

[17] linux, 2012. URL http://cdn.linuxforu.com/wp-content/uploads/temp-uploads/2009/05/fig01-android-arch.jpg. vii, 35

[18] mobilemarketingwelt, 2012. URL http://www.mobilemarketingwelt.com/wp-content/uploads/2010/11/Mobile-Health.jpg. vii, 36

[19] visi mobile, 2012. URL http://www.visimobile.com/wp-content/uploads/2011/11/product-header-770x288.jpg. vii, 36

[20] edp 24, 2012. URL http://www.edp24.co.uk/polopoly_fs/hidalgo_1_1220847!image/2364666984.jpg_gen/derivatives/landscape_630/2364666984.jpg. vii, 37

[21] businesswire, 2012. URL http://mms.businesswire.com/bwapps/mediaserver/ViewMedia?3Fmgid?3D330816?26vid?3D4. vii, 39

[22] dynatec, 2012. URL http://cdn.medgadget.com/img/dynatec.jpg. vii, 40

[23] shl telemedicine, 2012. URL http://www.shl-telemedicine.com/uploads/Products/product_12/big_thumb_smh.gif. vii, 42

[24] wifinotes, 2012. URL http://www.wifinotes.com/computer-networks/body-area-network.jpg2. viii, 44

[25] ban, 2012. URL http://www.cse.wustl.edu/~jain/cse574-08/ftp/ban/fig1.png. viii, 46

[26] Formoso de Pinho, Coimbra, Fernandes, and Oliveira. Droid jacket: a mobile monitoring system for a team, 2010. viii, 47

[27] uni graz, 2012. URL http://webpsy.uni-graz.at/gesundheitspsychologie/files/2011/12/GerteEquivital_a.jpg. viii, 51

[28] Hidalgo. Sem full disclosure interface specification, 2010. viii, 51, 52, 53, 54, 55

[29] bluetooth, 2012. URL http://www.netbookhelden.de/wp-content/uploads/bluetooth.jpg. viii, 56

[30] piconet, 2012. URL http://images.techtree.com/ttimages/story/75869_piconet.jpg. viii, 57

[31] chemnitz, 2012. URL http://www.tu-chemnitz.de/informatik/RA/news/stack/kompendium/vortr_2000/irdafunk/pics/b_prot.png. viii, 59

[32] spp, 2012. URL http://newsletter.atxx.de/html/img/pool/SerialoPortoProfile.JPG. viii, 61

[33] sensation, 2012. URL http://www.htc-sensation.de/wp-content/uploads/2011/04/htc_sensation_4g.jpg. viii, 62

[34] specification, 2012. URL http://smartphonegears.com/wp-content/uploads/2011/04/HTC-Products-HTC-Sensation-Specification.png. viii, 64

[35] activity lifecycle, 2012. URL http://3.bp.blogspot.com/-7x5xlYXl75o/ThFUE08R6uI/AAAAAAAAACQ/LYFkSNtSHQY/s1600/activity_lifecycle.png. viii, 69

[36] lifecycle, 2012. URL http://www.anddev.org/images/android/activity_lifecycle.png. viii, 70

[37] service, 2012. URL http://3.bp.blogspot.com/-KmxGX2mYUGQ/TngZ54rD-oI/AAAAAAAACJg/3dR42OCWCKE/s1600/service_lifecycle.png. viii, 72

[38] emulator, 2012. URL http://cdn3.blogsdna.com/wp-content/uploads/2010/05/Android-2.2-Froyo-Emulator-for-PC-and-Mac-OS-X6.jpg. viii, 75

[39] eclipse, 2012. URL http://www.guru-20.info/wp-content/uploads/2009/12/eclipse.png. viii, 77

[40] architecture, 2012. URL http://www.ibm.com/developerworks/aix/library/au-fiteclipse2/EclipseArchitecture.gif. viii, 78

[41] Hidalgo. Equivitaltm livelink standard edition, 2010. viii, 79, 80, 81, 82, 83

[42] Hdalgo. Equivital sem customise, 2010. viii, 83, 84, 86

[43] Mohammad Chowdhury, Sivarama Krishnan, and Siddharth Vishwanath. Touching lives through mobile health assessment of the global market opportunity. pages 1–36, 2012. 1, 8, 22

[44] NPR, 2009. URL http://media.npr.org/assets/img/2012/03/19/11430481_h8474024_wide-50425c785741f75f9517079fec4f5772b42e639c.jpg?s=4. 4

[45] gradnet, 2012. URL http://www.gradnet-db.wits.ac.za/include/images/MSF.Sudan-clinic-queue.jpg. 4

[46] bio lynx, 2012. URL http://www.bio-lynx.com/equivital/EQ01-1000.jpg. 5, 49

[47] iem, 2012. URL http://www.iem.de/ckfinder/userfiles/images/bluetooth.png. 5, 49

[48] go android, 2012. URL http://www.go-android.de/sites/default/files/images/u1/htc-sensation.jpg. 5, 49

[49] netbooknews, 2012. URL http://www.netbooknews.de/wp-content/uploads/android-logo-running2.jpg. 5, 49

[50] Bundesministerium der Finanzen. Ausgabenstruktur im bundeshaushalt 2012, 2012. URL http://www.bundesfinanzministerium.de/Content/DE/Bilder/Bildstrecken/Mediathek/Infografiken/Bundeshaushalt/20111206Ausgabenstruktur-BHH-2012.html. 6

[51] rzte Zeitung. Spanien dnnt gesundheitssystem aus, 2012. URL http://www.aerztezeitung.de/politik_gesellschaft/gesundheitspolitik_international/article/810216/spanien-duennt-gesundheitssystem.html. 6

[52] 1871 bis 2060 Anteile der Altersgruppen unter 20, ab 65 und ab 80 Jahre in Deutschland. Bundesinstitut fr bevlkerungsforschung, 2012. URL http://www.bib-demografie.de/DE/DatenundBefunde/02/Abbildungen/a_02_12_ag_20_65_80_d_1871_2060_saeulen.html?nn=3074114. 7

[53] Robert Litan. Vital signs via broadband: Remote monitoring technologies transmit savings. pages 1–10, 2008. 7

[54] WelTel. mhealth, 2012. URL http://www.weltel.org/. 8

[55] Telemedicine India. Apollo, 2012. URL http://www.telemedicineindia.com/. 8

[56] SkinVision. Skinvision, 2012. URL https://skinvision.com/. 8

[57] Vitality. Glowcaps, 2012. URL http://www.vitality.net/. 8

[58] Paul Cerrato. Mobile health revolution: Doctors and patients disagree, 2012. URL http://www.informationweek.com/healthcare/mobile-wireless/mobile-health-revolution-doctors-and-pat/240001916. 9

[59] GSM Arena. Os (operating system), 2012. URL http://www.gsmarena.com/glossary.php3?term=os. 10

[60] Andreas Constantinou. Mobile operating systems: the new generation, 2006. 11

[61] Gartner. Gartner says worldwide sales of mobile phones declined 2 percent in first quarter of 2012; previous year-over-year decline occurred in second quarter of 2009, 2012. URL http://www.gartner.com/it/page.jsp?id=2017015. 11, 19

[62] Andrew S. Tanenbaum. Moderne betriebssysteme, 2009. 12

[63] ARM. Symbian on arm, 2012. URL http://www.arm.com/community/software-enablement/symbian-on-arm.php. 12

[64] Eldar Murtazin. Review of nokia e7, 2012. URL http://mobile-review.com/review/nokia-e7-en.shtml. 12

[65] Nokia. Symbian platform, 2012. URL http://www.developer.nokia.com/Devices/Symbian/. 12, 14

[66] Andrew S. Tanenbaum. Moderne betriebssysteme, 2009. 13

[67] Windows. Windows phone, 2012. URL http://dev.windowsphone.com/en-us. 14

[68] Markus Weidner. Windows phone 7 von anfang an als zwischenschritt geplant, 2012. URL http://www.teltarif.de/microsoft-windows-phone-8-kernel/news/47343.html. 14

[69] Jane Sales with Martin Tasker. Introducing eka2, 2012. 14

[70] Istvan Cseri. Windows phone architecture: Deep dive, 2011. URL http://channel9.msdn.com/events/MIX/MIX11/DVC19. 16

[71] blackberry. Blackberry developer, 2012. URL https://developer.blackberry.com/. 16, 17

[72] Qusay H. Mahmoud. Programming the blackberry with j2me, 2005. URL http://www.oracle.com/technetwork/java/index-139239.html#0. 16

[73] Zach Epstein. More bad news: Developers are giving up on blackberry, too, 2012. URL http://bgr.com/2012/07/13/blackberry-developer-loyalty-2012/. 17

[74] Dieter Bohn. ios: A visual history, 2011. URL http://www.theverge.com/2011/12/13/2612736/ios-history-iphone-ipad. 17, 18

[75] Apple. ios technology overview, 2012. 18, 19

[76] Android. Developers android, 2012. URL http://developer.android.com/index.html. 19, 20, 66, 67, 68, 72, 76

[77] Open Handset Alliance. Open handset alliance, 2012. URL http://www.openhandsetalliance.com/. 19

[78] Reto Meier. Professional android 4 application development, 2012. 20, 22, 66, 71

[79] Brian Dolan. By 2012, 81? of physicians use smartphones, 2009. URL http://mobihealthnews.com/4744/report-81-of-physicians-to-use-smartphones-by-2012/. 23

[80] Frank Krschner-Pelkmann. rztemangel in afrika, 2010. 23

[81] Z. A. Khan M. Jaffar U. Rafiq A. Bibi N. A. Khan, N. Javaid. Ubiquitous healthcare in wireless body area networks, 2012. 23

[82] GSMA. Device listing, 2012. 23, 24

[83] ABI Research, 2012. URL http://www.abiresearch.com/research/service/mhealth/. 24

[84] US Food and Drug Administration, 2012. URL http://www.fda.gov/. 24

[85] Padma Nagappan. Sotera?s visi mobile vital-sign tracker ok?d, 2012. URL http://www.utsandiego.com/news/2012/aug/21/soteras-visi-mobile-vital-sign-tracker-okd/. 25, 26

[86] Gene Ostrovsky. Visi mobile wireless vitals monitoring system, 2010. URL http://medgadget.com/2010/06/visi_mobile_wireless_vitals_monitoring_system.html. 25

[87] Cambridge Wireless, 2012. URL http://www.cambridgewireless.co.uk/. 37

[88] Hidalgo. The eq02 lifemonitor, 2012. URL http://www.equivital.co.uk/products-new/tnr-products-new/sense-and-transmit. 37, 38

[89] Hidalgo. Eq02 lifemonitor sensor electronics module (sem), 2012. 38

[90] Zephyr. Bioharness, 2012. URL http://www.zephyr-technology.com/products/bioharness_bt. 38, 39, 40

[91] RS TechMedic. Dyna-vision, 2012. URL http://www.dyna-vision.com/index.php?option=com_content&view=category&layout=blog&id=6&Itemid=20&lang=de. 41

[92] SHL. smartheart, 2012. URL http://www.shl-telemedicine.com/products/12/48/smartheart/. 42, 43

[93] Brian Dolan. Fda clears smartheart mobile ecg device, 2012. URL http://mobihealthnews.com/17031/fda-clears-smartheart-mobile-ecg-device/. 42, 43

[94] Erik Karulf. Body area networks (ban), 2008. 44

[95] Bo Yu. Wireless body area networks for healthcare: A feasibility study, 2009. 44

[96] Upkar Varshney. Pervasive healthcare computing, 2009. 45, 46

[97] Yu-Wei Su Jin-Shyan Lee and Chung-Chou Shen. A comparative study of wireless protocols: Bluetooth, uwb, zigbee, and wi-fi, 2007. 47

[98] Linton Instrumentation. Hidalgo equivital lifemonitor, 2012. URL http://www.lintoninst.co.uk/Products/tabid/63/ProdID/599/Language/en-US/Equivital_LifeMonitor.aspx. 50

[99] Hidalgo Limited. Equivital eq-01 physiological monitoring, 2012. 55

[100] bluetooth organisation. Bluetooth basics, 2012. URL http://www.bluetooth.com/Pages/Basics.aspx. 55, 58

[101] Elektronik-Kompendium. Bluetooth, 2012. URL http://www.elektronik-kompendium.de/sites/kom/0803301.htm. 56, 57, 61

[102] bluetooth.org. Bliuetooth basics year =. 56, 58

[103] Mealling Leach and Salz. A universally unique identifier (uuid) urn namespace, 2005. URL http://www.ietf.org/rfc/rfc4122.txt. 56

[104] palowireless. Serial port profile, 2012. URL http://www.palowireless.com/infotooth/tutorial/k5_spp.asp#Scope. 61

[105] Jennifer Bray and Charles Thurman. Rfcomm, 2001. 61

[106] HTC. Specifications, 2012. URL http://www.htc.com/de/smartphones/htc-sensation/#specs. 62, 65

[107] Markku D. Lammerz. Android architektur, 2010. 67

[108] nVidia. Android lifecycle for application developers: Guidelines and tips, 2011. 68

[109] Victor Matos. Application?s life cycle, 2012. 73

[110] Ivan Ruchkin and Vladimir Prus. Single window integrated development invorenmnet, 2010. 76

[111] Eclipse. About, 2012. URL http://www.eclipse.org/org/#about. 77, 79

[112] Maximilian Kgel Wayne Beaton, Jonas Helming. Was ist eclipse?, 2011. URL http://it-republik.de/jaxenter/artikel/Was-ist-Eclipse-4007.html. 79