FH Technikum Wien

MASTER'S THESIS

Thesis submitted in partial fulfilment of the requirements
for the degree of Master of Science in Engineering

at the University of Applied Sciences Technikum Wien
Master Biomedical Engineering

Development and Implementation of Standardized Interfaces for an Android based Telemonitoring Server

by
Javier Urricelqui, BSc

Supervisor 1: Dipl. Ing. Ferenc Gerbovics
Supervisor 2: Dr. Ing. Luis Serrano

Vienna, 06.05.2011

FACHHOCHSCHULE
TECHNIKUM WIEN

Declaration

„I confirm that this thesis is entirely my own work. All sources and quotations have been fully acknowledged in the appropriate places with adequate footnotes and citations. Quotations have been properly acknowledged and marked with appropriate punctuation. The works consulted are listed in the bibliography. This paper has not been submitted to another examination panel in the same or a similar form, and has not been published. “

Place, Date

Signature

Resumen

El tema de este Proyecto Fin de Carrera es el “Desarrollo e Implementación de Interfaces Estandarizadas para un Servidor de Telemonitorización basado en Android” complementario al dispositivo HDH.

El Sistema de Telemonitorización conocido como Health Data Hub (HDH) ha sido desarrollado por un grupo de estudiantes de Máster en la University of Applied Sciences Technikum Wien obteniendo como resultado, un dispositivo tipo muñequera capaz de medir datos médicos mediante sensores, recibir datos de parámetros vitales por parte de dispositivos para el cuidado de la salud, además del reenvío de datos como mensajes de alarma y diversas actualizaciones a distintos servidores mediante interfaces estandarizadas.

Por lo que, el principal objetivo que trata esta proyecto es el desarrollo e implementación de interfaces estandarizadas para un servidor de telemonitorización basado en Android, capaz de gestionar los datos enviados por los dispositivos enmarcados en este proyecto según el estándar HL7 v2.6.

El dispositivo HDH es capaz de medir parámetros vitales, generar actualizaciones automáticas y mensajes de alarma para enviarlos posteriormente al Android Server mientras el paciente realiza su vida cotidiana. De esta manera, el usuario del Android Server puede disponer de información relativa al usuario del HDH.

El conjunto de dispositivos que se encuentran en el marco del proyecto HDH son el Servidor HDH, el cliente HDH, el Servidor Android y el dispositivo HDH. El servidor Android desarrollado durante este proyecto, consiste en una solución esencial de este entorno.

El Android Server está realizado sobre la plataforma Android, lo cual permite ofrecer nuevas funcionalidades y servicios que dotan de gran potencial al proyecto permitiendo llevar a cabo los objetivos propuestos.

Además, Android se define como código abierto, lo cual permite al desarrollador una fácil integración a otros entornos y provee herramientas que permiten que el desarrollo de la aplicación sea más atractivo para el programador.

Este proyecto ha sido desarrollado en colaboración con A. Tellechea [29], estudiante de la Universidad Pública de Navarra y estudiante de intercambio Erasmus en la University of Applied Sciences Technikum Wien.

Las tareas asignadas para el desarrollo de la aplicación Android Server fueron proporcionadas a ambos estudiantes, por lo que se trabajó conjuntamente en el desarrollo de la aplicación desde sus inicios hasta las últimas soluciones adoptadas para su desarrollo.

En lo concerniente al desarrollo de este proyecto ha sido implementado el cliente con tecnología Bluetooth del Android Server, además del desarrollo del servidor con tecnología Wi-Fi. En cuestión al procesado de datos, se ha colaborado conjuntamente debido a que es independiente de la tecnología que se use en la comunicación. Por otro lado, en este proyecto también se ha desarrollado la conexión entre el Android Server y el servidor principal.

Las otras partes del proyecto, como el desarrollo de la gestión del paciente, el graficado de sus datos clínicos, el diseño e implementación de la interfaz gráfica, simuladores usados etc fueron creados por ambos estudiantes, haciendo posible que los objetivos del proyecto fuesen concluidos.

Además, debido a que el funcionamiento del Android Server tenía que ser testeado en un entorno real, fue necesario trabajar con los responsables del desarrollo del HDH M. Bitterman [14] y el servidor principal P. Khumrin [47], ambos estudiantes de la University of Applied Sciences Technikum Wien

Abstract

The topic of this Master Thesis is the “Development and Implementation of Standardized Interfaces for an Android based Telemonitoring Server”.

A Telemonitoring-System called Health Data Hub (HDH) was developed in the University of Applied Sciences Technikum Wien by teamwork of Master Students. The outcome of this project was a wrist wearable health device which is capable of measuring health data via built-in sensors, to receive data from personal health devices over standardized interfaces and to forward data with alarm messages and status updates to multiple servers.

Thus, the main goal of this thesis is the development and implementation of standardized interfaces for a telemonitoring server on the Android platform, in order to deal with the health data sent by the HDH. This health data is based on HL7 v2.6 standard.

In this way, HDH is able to measure the general well-being and health status of the person wearing it and to generate automatic status updates and alarm messages with high sensitivity and high specificity to be received by the Android server and reviewed by a monitoring person.

Basically, the HDH infrastructure consists of 4 main components, the HDH wrist device, the HDH Server, HDH Client and HDH Android Server. Thus, the Android Telemonitoring-Server solution created is intended to act as an essential part of this telemonitoring ecosystem.

The Android Server is done on Android programming language, which offers great potential for implementing new functionality and services. Being open source allows fast integration with other platforms and offers tools that facilitate the development of applications. All these features make Android powerful tool for the development of this project.

This Master Thesis has been developed in collaboration with A. Tellechea [29], also student from Public University of Navarra and incoming exchange student in University of Applied Sciences Technikum Wien by the Erasmus program.

Due to the fact that the development of this Android Server application was proposed to both Master Thesis students, they worked hand in hand in the achievement of all the targets: from the first steps learning the Android programming language until the performance analysis, design of the application and the adopted solutions.

Regarding the connections, I dealt with the implementation of the Android Server (acting as the Client) via Bluetooth and also with the development of the part in where the system works with WiFi technology as a Server. The processing of the data and the way to work with

them, since it is independent of the used technology, was developed by both students. On the other hand, I developed the Connection between the Android Server and HDH Main Server running nowadays in this application.

The other parts of the Thesis, for instance the performance of the patient information management, everything regarding the medical graphics, as well as the design and implementation of the GUI and the simulators etc. were created by both students. Thanks to that, the targets given in the beginning of the Master Thesis were successfully achieved.

Furthermore, since the correct implementation of all functionalities created in the Android Server application had to be tested in real scenario, it was needed to work with the responsible of the HDH device M. Bitterman [14] and the responsible of the Main Server P. Khumrin [47]

Acknowledgements

To the Public University of Navarra, for giving me the necessary knowledge for face the professional challenges. To Dr. Ing. Luis Serrano, the responsible of this exchange program, for making possible the relationship between both universities.

To the University of Applied Sciences Technikum Wien, to the Healthy Interoperability group and especially want to cordially thank my project leader Dipl. Ing. Ferenc Gerbovics for his support and proofreading of this master's thesis.

To all the people related with the students exchange program, who work to do possible this experience.

To all the friends that I have met during this year.

At last but not least, from the bottom of my heart, thanks to my family for always being there when I need you.

A la Universidad Pública de Navarra, por el conocimiento que me ha proporcionado para afrontar este reto profesional. A Dr. Ing. Luis Serrano, como responsable del programa de intercambio que hace posible que la relación entre ambas universidades sea llevada a cabo.

A la University of Applied Sciences Technikum Wien, al grupo de Healthy Interoperability y especialmente a mi tutor Dipl. Ing. Ferenc Gerbovics por su apoyo y corrección de este proyecto fin de carrera.

A todas las personas que hacen posible que este intercambio de estudios se haya podido realizar.

A todos los amigos que he conocido durante este año.

Y aunque en última instancia pero no por ello menos importante, a mi familia, por estar siempre cuando más os he necesitado.

Table of Contents

1 Introduction

In this point, I deal firstly with the benefits happened in health-care because of the Telemonitoring practices. In this context, I explain the eHealth concept, the importance of this emerging field not only as a technical development but also as an attitude, and a commitment for networked to improve health by using information and communication technology.

I also explain why a standard for communication is needed, talking about ISO/IEEE 11073 and HL7,the most widely used standard messaging in the healthcare industry.

It is also introduced the Health Data Hub wrist device developed by Health Interoperability team in the UAS Technikum Wien.

As I worked with Android, I briefly introduce it and Eclipse too, the Android development environment.

It is also important to describe the HDH project, inside the Healthy Interoperability research Project developing in UAS Technikum Wien. Finally I talk about motivation, objectives and the structure of the memory.

1.1 Telemonitoring Advances Home Care Practices

Depending on the severity of illness of patients, there are cases in which they should be monitored in real time. That is, if a patient is suffering a chronic disease, the ideal would be to examine him continuously 24 hours a day. However, this fact clearly implies the need for space in hospitals for each of these patients as well as qualified professionals, which is a huge expenditure in the health system. Unfortunately, in many cases patient can´t have the best treatment.

The telemonitoring is presented as the solution of this problem. From a generic point of view, a home telemonitoring system basically consists of medical devices needed to take measures and a central computer called a gateway or that acts as a gateway integrated engine connection to the telemonitoring server.

The Gateway, which is usually a PC, PDA, phone or any device that can perform this function, will be the responsible of controlling the communication between the media devices to the server. Thus, monitoring of the patient happens.

In this communication system, it is imperative that all devices are equipped with a proper communication capability. Of course, wireless technologies provide greater comfort for

the user but their communication protocols must also ensure the safety of information. One of the targets of the personal Telemonitoring system is to be a plug and play communication between devices forming the network, so that these devices connect, configure and transmit their data without intervention by the patient (who, besides, wouldn´t need to have any technical knowledge).

In next point called *1.7 Background of the Project: HDH project at the UAS Technikum Wien* it is detailed the telemonitoring system with which we are working on now.
A very brief summary of it would be this: We have a device called HDH (Health Data Hub-prototype) which communicates with the Android mobile phone (it performs the function of a Gateway). The protocols used for the communication is Bluetooth, GSM and WiFi. At the same time, the Android phone communicates with a central server. Currently, it works via Wifi. The Android periodically sends the accumulated data, therefore, the central server has all the information about the patient.

1.2 eHealth, transforming the healthcare landscape

eHealth is an emerging field in the intersection of medical informatics, public health and business, referring to health services and information delivered or enhanced through the Internet and related technologies. In a broader sense, the term characterizes not only a technical development, but also a state-of-mind, a way of thinking, an attitude, and a commitment for networked, global thinking, to improve health care locally, regionally, and worldwide by using information and communication technology.

Because the Internet created new opportunities and challenges to the traditional health care information technology industry, the use of a new term to address these issues seemed appropriate. These "new" challenges for the health care information technology industry were mainly the capability of consumers to interact with their systems online, improved possibilities for institution-to-institution transmissions of data; new possibilities for peer-to-peer communication of consumers.

Examples include health information networks, electronic health records, telemedicine services, wearable and portable systems which communicate with each other, health portals, and many other ICT-based tools assisting disease prevention, diagnosis, treatment, health monitoring and lifestyle management.

As said, eHealth means also Information and Communication Technology tools and services for health. Whether eHealth tools are used behind the scenes by healthcare professionals, or directly by patients, they play a significant role in improving the health of citizens.

Information and Communication Technologies (ICTs) have an ever-growing impact on our working and private lives, and the healthcare sector is not an exception. Used appropriately, the tools and services which contribute to eHealth provide better, more efficient healthcare services for all.

This explanation hopefully is broad enough to apply to a dynamic environment such as the Internet and at the same time e-health encompasses more than just "Internet and Medicine".

As such, the "e" in e-health does not only stand for "electronic," but implies a number of other "e's," which together perhaps best characterize what e-health is all about.

For example:

- Efficiency: increase efficiency in health care and also decrease costs.
- Enhancing quality of care: for example by allowing comparisons between different providers.
- Evidence based: effectiveness and efficiency should not be assumed but proven by rigorous scientific evaluation.
- Encouragement of a new relationship between the patient and health professional.
- Enabling information exchange and communication in a standardized way between health care establishments.
- Extending the scope of health care beyond its conventional boundaries. eHealth enables consumers to easily obtain health services online from global providers.
- Ethics: it involves new forms of patient-physician interaction and poses new challenges and threats to ethical issues such as online professional practice, informed consent, privacy and equity issues.
- Equity: to make health care more equitable is one of the promises of e-health, but at the same time it is a handicap to avoid. People, who do not have the money, skills, and access to computers and networks, cannot use computers effectively. As a result, these patient populations are those who are the least likely to benefit from advances in information technology, unless political measures ensure equitable access for all.

In addition e-health should be easy-to-use.

eHealth systems provide patients with better information, for instance, on treatments, on their condition, and on improved standards of living. The use of electronic patient records allows doctors to see much more of a person’s medical history than do paper files, which typically only include information on treatment in a single surgery or hospital. A patient’s

condition can be monitored remotely, either freeing up a hospital bed which would have been required with previous monitoring equipment, or providing a better standard of care for the patient.

1.3 Why a standard for communication is needed: Standard ISO/IEEE 11073

In general, information systems require standards for the storage, retrieval and manipulation of information. In all voice, data or video signal systems, communication can´t happen without standards for sending and receiving messages. The same happens in the environment of home telemonitoring systems.

However, usually, medical devices from different manufacturers exist, and they work with their own formats. That means they have different ways of recording and transmitting data using different protocols and semantics. It shows clearly a great need for standardization in this area.

With the standardization of communications in the home telemonitoring system area decrease of costs can be achieved, by being able to choose from a wide range of products from different companies. At the same time, it would facilitate the replacement and addition of devices and provide a unique platform to develop new equipment for the systems, progressing in this area faster. Interoperability allows plug and play systems in which the system automatically detects devices when they connect, configure them and communicate with them without user intervention.

For all these reasons, they created a standard for medical device communication.

The current standard for medical device communication is the ISO / IEEE 11073 (X73) [1]. This family of standards covers the seven levels of the OSI protocol stack and provides the versatility to convert information into an interoperable format so that it can be exchanged between personal health devices and a server infrastructure. Data can then be sent to a remote place for control or storage of an Electronic Health Record, and by this, to build more complete systems.

In other words, CEN ISO/IEEE 11073 Health informatics - Medical / health device communication standards enable communication between medical, health care and wellness devices and with external computer systems. They provide automatic and detailed electronic data capture of client-related and vital signs information, and of device operational data.

The X73 standard, which was developed focusing on patient Point of Care (Point of Care, X73-PoC) and on the implementation in Intensive Care Units (ICUs) has experienced in recent years several developments not covered initially (new use cases, new networking technologies, new profiles, etc.).

1.4 HL7, Health Level Seven

As mentioned before, the environment in which this project was created was the healthcare field, as the data working with was medical data. The data transmitted from the wrist device (HDH) to the Android server is the pulse and other parameters like activity and altitude. Then the data is processed in the Android Server and sent to the main server. It is mandatory to use this standard for sending and receiving medical data.

As cited, we are working with the HL7 standard [2]. The name references to the top layer (Level 7) of the Open Systems Interconnection (OSI) layer protocol for the health environment. The HL7 standard is the most widely used standard messaging in the healthcare industry around the world. Later, in the section *2.2 HL7* it´s possible to find more information on this subject and there is a deeper explanation of the messages that we used (fields, construction, analysis, treatment etc.) in the section *3.3 Parser: from HL7 message to HL7*.

1.5 Android

Android [3] is a mobile operating system initially developed by Android Inc. Android was bought by Google in 2005. Android is based upon a modified version of the Linux kernel.

Google and other members of the Open Handset Alliance collaborated on Android's development and release. The Android Open Source Project (AOSP) is tasked with the maintenance and further development of Android.

As said, Android is the product of primarily Google but more appropriately of the Open Handset Alliance. The Open Handset Alliance is an alliance of approximately 30 organizations committed to bringing a “better” and “open” mobile phone to the market. A quote taken from its website says it best: “Android was built from the ground up with the explicit goal to be the first open, complete, and free platform created specifically for mobile devices” [4].

The expert in analysis for the hight-tech industry, Canalys [5], reported that in 2010 the Android O.S. was the world's best-selling smartphone platform, dethroning Nokia's Symbian from the 10-year top position.

1.6 Eclipse

Eclipse [6] is an open source community, whose projects are focused on building an open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle. The Eclipse Foundation is a not-for-profit, member supported corporation that hosts the Eclipse projects and helps cultivate both an open source community and an ecosystem of complementary products and services.

Eclipse, from the viewpoint of software, is a multi-language software development environment comprising an integrated development environment (IDE) and an extensible plug-in system. It is written mostly in Java and can be used to develop applications in Java and, by means of various plug-ins, other programming languages including Ada, C, C++, COBOL, Perl, PHP, Python, Ruby (including Ruby on Rails framework), Scala, Clojure, and Scheme. The IDE is often called Eclipse ADT for Ada, Eclipse CDT for C/C++, Eclipse JDT for Java, and Eclipse PDT for PHP.

The initial codebase originated from VisualAge. In its default form it is meant for Java developers, consisting of the Java Development Tools (JDT). Users can extend its abilities by installing plug-ins written for the Eclipse software framework, such as development toolkits for other programming languages, and can write and contribute their own plug-in modules.

Released under the terms of the Eclipse Public License, Eclipse is free and open source software [6].

1.7 Background of the Project: HDH project at the UAS Technikum Wien

The Healthy Interoperability [7] research Project, developing at the University of Applied Sciences Technikum Wien, has the goal to implement interoperable telemonitoring solutions based on international standards together with business partners in pilot projects. One of the latest developments within this project is the Heath Data Hub (HDH), a wrist wearable medical device which measures health data with built-in sensors [8] .

The HDH is also capable of receiving medical data from personal heath devices (PHDs) which utilize the Continua Health Alliance guideline [9] (ISO/IEEE 11073) or the ANT [10] protocol. Furthermore, the received and measured values are sent by the HDH to a server. Hereafter the server redirects alarm messages and status updates about the HDH users to a monitoring person [8].

Methods

The CPU (Microchip Technology inc., Arizona, USA; PIC32) of the HDH is receiving raw data from buit-in sensors and standardized medical data from PHDs over the ANT+ and Bluetooth-module. The pulse rate sensor is based on the principles of pulseoximetry. Safeguarding real-time measurements and peak detection algorithms are operated on a separate processing unit. Sensors for altitude and acceleration measurement are utilizing the I2C bus to communicate with the CPU. A capacity sensor is used to verify the presence of the HDH user. Measured data is transferred to the server using the Bluetooth-module, in case the server is not available the data is buffered on the SD-Card [8].

Results

In general the HDH is capable of measuring pulse rate values with the built-in sensor, to receive standardized data from external PHDs and to put the available data into context to each other using intelligent algorithms.

To work in the intended way, the HDH also needs an infrastructure to exchange data. This is provided by two types of communication servers: Android Server and HDH Server (see Figure 1.1). Data is sent to the servers by means of HL7 V2.6 messages which are acknowledged by the server. User feedback is achieved via the touch-screen [8].

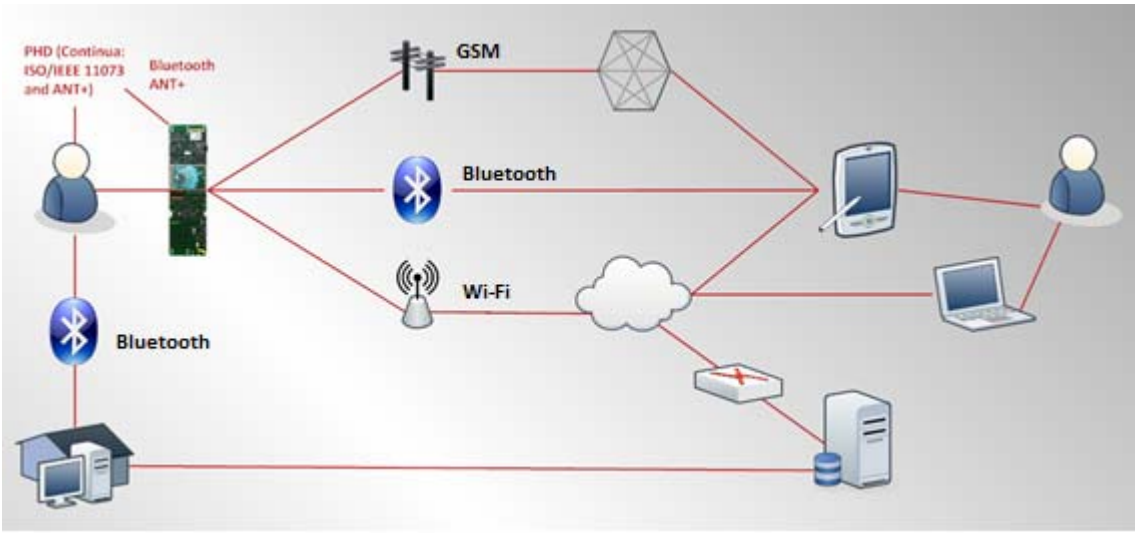


Figure 1.1 HDH Infrastructure

Discussion

The HDH implements the basic functions as described. Minor development steps are still open to achieve conformity to industry standards. Another major issue is the social distance between the HDH user and their relatives and care givers. This technology can complement but never replace personal care [8].

Acknowledgements

This project is supported by the Austrian Research Promotion Agency (FFG) within the "FHPlus in coin" program and by the City of Vienna Municipal Department 27 "EU-Strategy and Economic Development" following the "Fachhochschulförderrichtlinie" 2005 [8].

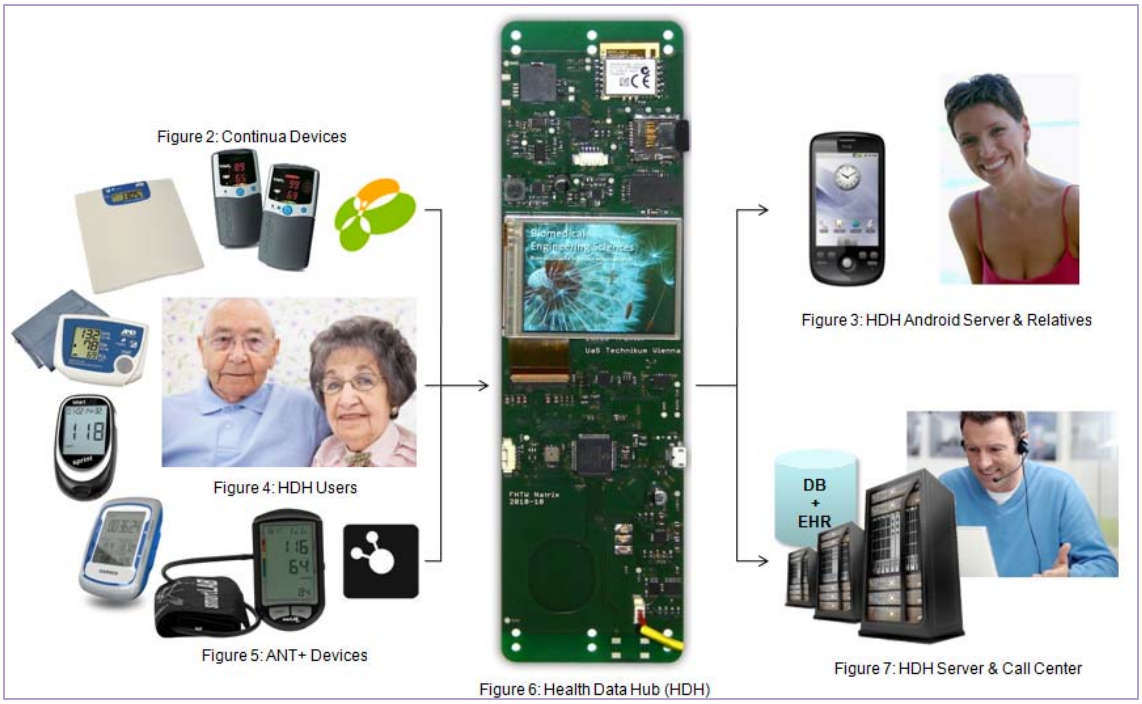


Figure 1.2 Environment of the HDH

1.8 Project motivation

As explained in the previous section, this project is a part of a larger project developed at the UAS Technikum Wien: Implementation of a wrist wearable Health Data Hub-prototype for context based telemonitoring of elderly people using standardized data transfer methods.

In a few words, this system in development is aimed at the capture of certain vital signs of patients, mostly elderly patients and achieves telemonitoring. Thus making viable the continuous monitoring of them over long-distances and in case of emergency to provide assistance immediately.

The idea to help a greater number of people is very appealing. Through this system, it is not only possible to manage medical monitoring, but also it is possible to monitor many patients.

Regarding to the health-care of elderly people, the role of the family is indisputable. Caring for elders takes a long time for their relatives. The care of the elderly requires

much effort, and for the most of the society caring for them 24 hours per day is impossible. The fact of having to leave alone their elders makes the family members feel ill at ease. Instead, through the introduction of this system, this wouldn´t be a problem.

By the incorporation of the developed project in the real life family members would have the possibility to monitor over a large distance only by using an Android mobile (with Android server). Thanks to this system, family members may be in a different place, as for example in their work place, or even doing their routine tasks outside, having assurance that their elders feel well. And if ever there is any complication, through the system of alert messages family members would be informed instantly, and can reach their home immediately or even call health services if necessary.

As this would be an almost automatic process, because the users don´t have to worry about the operation of the HDH (i.e. communication processes as sending or receiving of messages...) This results in an easy and comfortable system, to be used by a large number of patients.

The fact of thinking that this project could be set in practice is very important. It is a good feeling that working in a project that in future can be used in real life. It is fantastic to help patients at home, making them and their relative's life more comfortable, and improving the quality of health care.

The implementation of this system allows reducing the number of times that the patient must go to the hospital for routine measures taken, and medical services would not have the need to move to the patient's house that often (saving in this way both time and money). It also ensures that their families can realize their tasks with the full confidence that their elders are getting the best care.

The truth is that nowadays, the implementation of new devices for telecommunications in the medical field is a very exciting option. This project, in the context of a bigger project, was very gratifying.

Besides, speaking in technical terms, using a platform of open source as Android, gives important knowledge, useful for the introduction in the professional world.

Android is a multitasking operating system, it offers an advanced development environment and by this it guarantees high level benefits. And it is free software; it allows easy integration with other implementations and platforms, something that does not happen with proprietary systems.

In addition, the majority of mobile phone manufacturers are moving to it. It seems like the election of Android for the implementation of the project is correct.

If we add to all these amenities the use of a wireless technology, such as Bluetooth, the facilities that the system offers are very good. Today, user demands governing the market are the wireless technologies. This is very natural because it is a very handy option. In the developed system the use of cables is not necessary to measure the elder´s parameters, which is more comfortable. It is inconceivable to establish a system which uses a lot of cables because it would considerably limit the movements of the patients, decreasing their mobility, thus decreasing their quality of life. All this is superseded by Bluetooth technology.

The knowledge that this project offers, like knowledge about medical messages (HL7) and how the medical protocols work is also very interesting.

1.9 Objectives of the Thesis

The main target of this project is the Development and Implementation of Standardized Interfaces for an Android based Telemonitoring Server, complementary to the HDH.

The HDH Android Server designed shall be capable of:

- Receive, acknowledge and permanently store HL7 V2.6 (optionally ISO/IEEE 11073-20601) messages (alarm messages or status updates) from the HDH via Bluetooth, WiFi and GSM.
- Manage data internally by different sender (for monitoring multiple persons with a single android device).
- Communicate alarm messages to the user of the Android device via acoustic and haptic feedback.
- Provide servicing mode for exporting data to HDH Server.
- Display graphs of trends of received values.

In the later section, *3. Development and Implementation*, is specified how the analysis and development has been performed of each of the objectives listed above. What the characteristics of each of the targets are, regarding the design, programming and implementation. What difficulties were encountered while implementing them and how they were solved. In conclusion, overcoming all objectives given at the beginning of this project.

1.10 Structure of the Thesis

Chapter 1

The first chapter introduces the context in which this Master Thesis has been developed and it addresses the general and specific objectives, detailing the contents of the various sections and subsections of the thesis.

Chapter 2

The second chapter develops the state of the art of this master thesis, dealing with some of the most important concepts of it. For instance, the health care organization and standards are mentioned, more specifically the ISO/IEEE 11073 and HL7. This chapter also refers to the HDH (Health Device Hub) wrist device. On the other hand, it is shown the current landscape of existing operating systems and due the fact that Android is the main character of this master thesis, deeper information about it is given. Finally, Eclipse, a development environment for Android, is introduced.

Chapter 3

This chapter presents the developed Android Server application. It deals mainly with the development and implementation of this application. This chapter begins talking about the different connections between the Android phone and both the HDH wrist device and the Main Server. The related technologies used for the connections are explained: Bluetooth, Wifi and GSM. All the message types used in the communication are explained and the alarm messages are described also. The messages created in order to achieve the correct performance of the system are deeply analyzed and the used HL7 parsed too. Besides, the patient information management in the system is explained. Furthermore, the graphics developed with the purpose of illustrate the received health data information are analyzed. At the end of the chapter, the design of both the graphical user interface and the simulators used while trying the connections are described.

Chapter 4

The chapter 4 deals with the evaluation and results obtained after finishing this master thesis. In the results, the different connections in the system are analyzed. Firstly, there are mentioned the connections between the HDH wrist device and the Android Server application via Bluetooth, Wifi and GSM. Moreover, it is analyzed the connection between the Android Server application and the Main Server (via Wifi). In the end, the graphical user interface is tested also.

Chapter 5

To conclude, the conclusions obtained after the development of this thesis are given. After acquiring the appropriate knowledge, there are revealed some guidelines to consider in the future.

2 State of Art

In this section, it will be explained deeply the organizations and standards for health, ISO/IEEE 11073 and also the standard for medical messages, HL7 (the organization, the standard and HAPI, a very useful tool).

It is introduced the Health Data Hub wrist device developed at the UAS Technikum Wien by the Healthy Interoperability team.

Later, several mobile OS are mentioned, as Symbian, Blackberry, Windows, iPhone, and for sure, Android, showing the evolution of sales both in Europe and around the world.

Since the tool worked with is Android, more about this issue will be explained; the Platform, Developers, the Market and its Virtual Machine Dalvik and about other respects too.

Later, different integrated development environments for Android are introduced with focus on Eclipse, talking about the Android SDK and ADT plugin.

2.1 Health care Organization and Standard

The main organizations involved in medical informatics and ICT for health and among other important activities, collaborating on the development of standards are:

CEN, *the* ***European Committee for Standardization*** *or* ***Comité Européen de Normalization*** [11] is a non-profit organization whose mission is to foster the European economy in global trading, the welfare of European citizens and the environment by providing an efficient infrastructure to interested parties for the development, maintenance and distribution of coherent sets of standards and specifications.

ISO, International Standards Organization and ***IEEE, Institute of Electrical and Electronics Engineering*** [12]***,*** organizations with more responsibilities, even the international committees depend on them.

Following there is an explanation about the most important standard for medical interoperability information system, ISO/IEEE 11073(X73) [1].

2.1.1 ISO/IEEE 11073

The primary goals of ISO/IEEE 11073 [1] standards are to provide real-time plug-and-play interoperability for personal-related medical, healthcare and wellness devices and

facilitate efficient exchange of device data, acquired at the point-of-care, in all care environments.

The standards are targeted at personal health and fitness devices and at continuing and acute care devices. There are four main partitions to the standards:

- Device data. It includes a nomenclature, optimized for vital signs information representation based on an object-oriented data model and device specializations.
- General application services.
- Internetworking and gateway standards (e.g., an observation reporting interface from CEN ISO/IEEE 11073-based messaging and data representation to HL7 [2] or DICOM).
- Transports (e.g., cable connected or wireless). In a non-standardized device connectivity environment it would be some unavoidable problems.

For example, in the absence of standards for these devices, data is captured either manually or it is not captured at all, which is most often the case. Manually captured data is intensive labor, recorded infrequently and prone to human error.

With no standardized in this area, even when similar devices do provide communications, there is no consistency in the information and services that are provided, thus inhibiting the development of advanced care delivery systems or even consistent health records.

ISO/IEEE 11703-20601:2010 defines a common framework for making an abstract model of personal health data available in transport-independent transfer syntax required to establish logical connections between systems and to provide presentation capabilities and services needed to perform communication tasks. The protocol is optimized to personal health usage requirements and leverages commonly used methods and tools wherever possible [1].

ISO/IEEE 11703-20601:2010 addresses a need for an openly defined, independent standard for converting the information profile into an interoperable transmission format so the information can be exchanged to and from personal telehealth devices and computer engines [1].

2.2 HL7

2.2.1 The organization

Health Level Seven International (HL7) is the global authority on standards for interoperability of health information technology with members in over 55 countries.

Health Level Seven International (HL7) is a not-for-profit, ANSI-accredited standards developing organization dedicated to providing a comprehensive framework and related standards. The main targets are the exchange, integration, sharing, and retrieval of electronic health information that supports clinical practice. It also deals with the management, delivery and evaluation of health services. HL7's 2,300+ members include approximately 500 corporate members who represent more than 90% of the information systems vendors serving healthcare.

HL7 deals mainly with the creation of the best standards in healthcare [2].

2.2.2 The Standard

Health Level 7 (HL7) [2] is also used to refer to some of the specific standards created by this organization (for example HL7 v2.x, v3.0, HL7 RIM). It is an ANSI standard for healthcare specific data exchange between computer applications.

The name itself refers to the top layer (Level 7) of the Open Systems Interconnection (OSI) layer protocol for the health environment. The HL7 standard is the most widely used messaging standard in the healthcare industry around the world.

Many different computer systems are used in the hospitals and other healthcare provider organizations. All of these systems should communicate with each other when they receive new information but not all do so. Thanks to the HL7, which specifies a number of flexible standards, guidelines and methodologies, various healthcare systems can communicate with each other. Such guidelines or data standards are a set of rules that allow information to be shared and processed in a uniform and consistent manner. These data standards are meant to allow healthcare organizations to easily share clinical information. Theoretically, this ability to exchange information should help also to minimize the tendency for medical care to be geographically isolated.

There are four types of standards developed by HL7:

- Conceptual standards (e.g., HL7 RIM).
- Document standards (e.g., HL7 CDA).
- Application standards (e.g., HL7 CCOW).
- Messaging standards (e.g., HL7 v2.x and v3.0). These standards are especially important because thanks to them it is explained how information is packaged and communicated from one party to another.

HL7 encompasses the complete life cycle of a standards specification including the development, adoption, market recognition, utilization, and adherence.

HL7 Members can access standards for free and non members can buy the standards from HL7 or ANSI.

The development of HL7 is linked, among others, the following standards:

- ANSI HISPP (Healthcare Information Standards Planning Panel) for the coordination of the various standards in the health field.
- ANSI HISB (Healthcare Information Standards Board).
- ASC X12N Group, for the development of EDI standards.
- ASTM E31.11 Group, dedicated to the exchange of clinical data.
- ACR / NEMA DICOM Group, dedicated to medical imaging standards and other aspects of Radiology Information Systems.
- IEEE P1157 G.

[2]

2.2.3 Very useful HL7 tool: HAPI

HAPI (HL7 application programming interface) is an open-source, object-oriented HL7 2.x parser for Java [13].

HAPI is a powerful tool to parse HL7 messages into an HL7 object. It is a tool where by simply introducing a HL7 message you can obtain the result of all the separate fields of the message, the name of each field or to do error correction. It is a great tool which makes the programmers job much easier. The parsing of the message is an important part of the project as it does the interpretation of the received medical message.

Unfortunately, the use of HAPI for this project was not applicable. As mentioned at the beginning, HAPI is a Java tool and in this project Android was used.

Note that Android and Java are two different platforms which share the same programming language, as well as basic programming libraries. But in this case, it isn't possible to use the HAPI tool in Android.

In order to understand why the Google API is not 100% compatible with the JavaSE you can read *Annex VI. Java vs. Android*, where this issue is explained in detail. Briefly, Android uses a virtual machine called Dalvik which is not the same as the Java virtual machine, causing that not all Java applications are executable in Android.

2.3 Health Data Hub

The Health Device Hub is a medical device developed by the Healthy Interoperability (HIO) [7] team from the Biomedical Engineering & Information Engineering Department of the University of Applied Sciences Technikum Wien. The HDH was created to be wearable on the wrist of the lower arm to measure the pulse. Following information was provided by *M. Bitterman, a Master Thesis student also from the University of Applied Sciences Technikum Wien responsible for the "Development of the Data Model for the wrist wearable Health Device Hub"* [14].

The HDH should be capable of acquiring the pulse of the patient as well as monitoring it. The observation of the pulse is done in relation to physical activity. Moreover, it ensures the reliable transfer of generated standard and alert messages to different receiver devices like smart phones or server applications on stationary PCs. In order to achieve the mentioned tasks, following modules are integrated:

- *Sensor modules*
  - *Pulse meter for acquisition of the heart rate*
  - *Accelerometer for activity measurement*
  - *Altimeter for additional information of the activity*
  - *Temperature sensor*
- *Communication modules for wireless data transmission*
  - *ANT+*
  - *Bluetooth*
- *OLED display with touch-screen function*
- *Micro USB for data communication and power supply for recharging the battery.*
- *Micro SD card reader for local storage management*
- *Real time clock unit*
- *Power supply (battery)*

Due the fact that the HDH has to be a wristband device, it has to be very comfortable, small and very lightweight.

The figure **2.1** *2.1 represents the first prototype of the HDH. Obviously, it is based on the mentioned design requirements. The alignment of the electrical parts and the signal routing are developed to create further prototypes on*

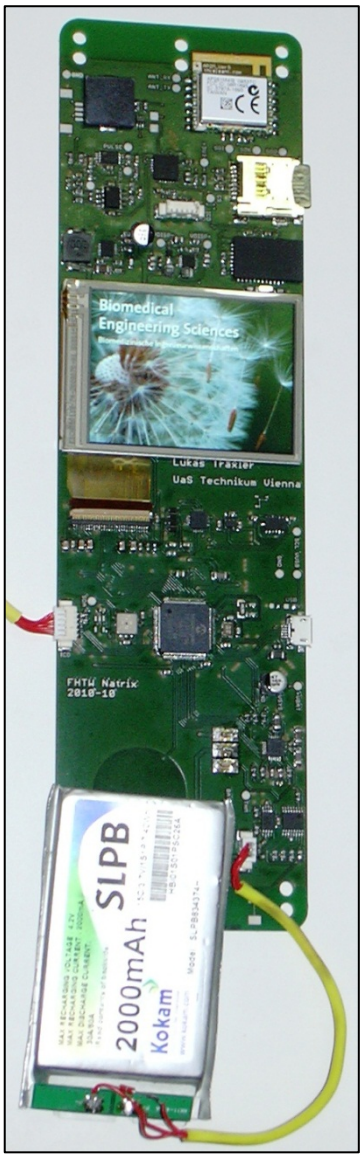


Figure 2.1 HDH-first prototype

a flex printed circuit board. This will make it possible to wear it like a watch with the display on the upper side and the pulse sensors on the lower side of the forearm [14].

Hardware

Figure 2.2 illustrates the position of the main components of the first prototype design [14].

Accelerometer

The ADXL345 module from Analog Devices Inc. USA. It is a thin, low power, 3-axis accelerometer with a fixed 10-bit resolution and a full resolution up to 13 bit at up to ± 16g. This enables measurement of inclination changes less than 1.0°.

ANT+

An AP2 RF transceiver module from Dynastream Innovations Inc. provides ANT+ data communication in the 2.4GHz worldwide ISM band with 78 selectable RF channels (2403 to 2480MHz) and a RF data rate of 1 Mbps. Further it supports parallel connection up to 8 ANT channels and is optimized for ultra-low power operation

Altimeter

The digital pressure sensor BMP085 from Bosch Sensortec GmbH Germany, is able to measure the pressure from 300hPa to 1100hPa (-500m to +9000, above sea level) with a resolution of 16bit to 19bit and an accuracy of 0.01hPa. Further it provides the temperature measurement from 0°C to 65°C in a 16 bit range and a resolution of 0.1°C. The pressure and temperature data is to be compensated by the calibration data in the E^2PROM of the sensor, used to compensate offset, temperature dependence and further parameters of the sensor.

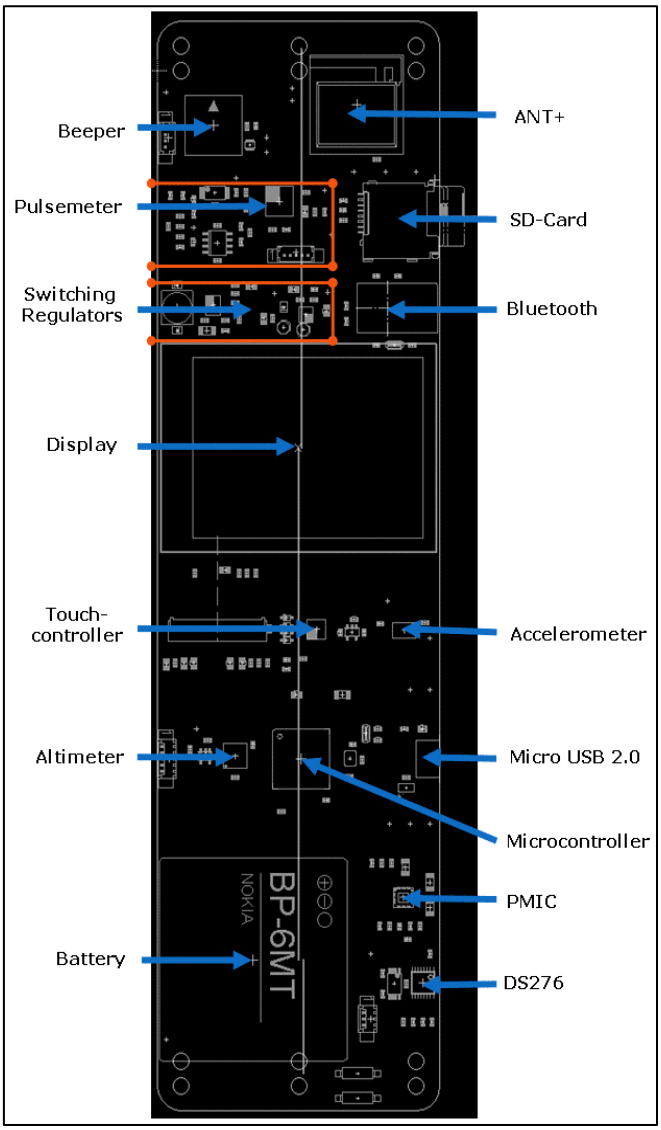


Figure 2.2 HDH hardware design

Battery Protector

A high precision lithium battery monitor DS2764 from Maxim Semiconductor Dallas is used for battery protection provides current charging level, voltage and charge/discharge current of the battery and protects against overvoltage and over current.

Bluetooth

The LMX9838 Bluetooth serial port class 2 operating module from National Semiconductor is a fully integrated Bluetooth 2.0 baseband controller including all hardware and firmware to provide a complete solution from antenna through the complete lower and upper layers of the Bluetooth stack. The stack includes: Baseband and link manager, Protocols: L2CAP, RFCOMM, SDP, Profiles: GAP, SDAP and SPP.

The module supports data rates up to the theoretical maximum over RFCOMM of 704kbps and a UART command/data port speed up to 921.6kbits. The firmware features point-to-point and point-to-multipoint link management for up to 7 active Bluetooth data links and one active SCO link.

Display

An S6E63D6 260k RGB colour AMOLED panel from Samsung Electronics Inc. is a single chip solution with a maximum resolution of 240x320 pixels. Therefore it supports pictures with 18-/16-/6-bit RGB colours.

Touch Controller

For user operation of the HDH, an mTouch AR1020 Resistive Touch Screen controller from Microchip Technology, Inc. is implemented. It supports different touch modes like, stream, down, up, off and more with a touch resolution of 10bit resolution maximum and a touch coordinate report rate of 140 reports per second. Further it provides touch algorithms for good calibration – corrected coordinates with no additional development or code.

Microcontroller

The core of the HDH is the high performance PIC32MX795F512L microcontroller from Microchip Technology Inc. This is 32 bit module with a maximum frequency of 80 MHz, 512k flash memory plus an additional 12kB of boot flash and 128kB SRAM memory.

Micro USB 2.0

The micro USB 2.0 plug in provides data exchange from the SD-card of the HDH with external devices. Additionally the battery can be recharged by means of the power supply from the USB connector.

SD-Card slot

It is used as a local mass storage for micro SD-cards. It also grants access to the micro SD-card when plugged to a PC via USB [14].

Software

The source code for the microcontroller is written in C- programming language with the MPLAB IDE v8.56 and the MPLAB X IDE beta version 5 programming environments from Microchip Technology Inc. The Integrated Circuit Debugger 3 (ICD3) device is used to run the source code in the debug mode and to flash the compiled and linked code onto the microcontroller. Therefore the C32 (v1.11) compiler, provided by Microchip Technology Inc. for the 32bit microcontroller, is used.

A FatFS file system library from ELM– FatFS Generic FAT File System Module is implemented for the file system of the SD-card. It supports complete access of SD cards and handles FAT12, FAT16 and FAT32 file systems. Therefore it provides several functions necessary, to manage a complete file system [14].

2.4 Mobile Operating System

The following graphs [15] show the evolution of sales of the major mobile OS, both in Europe and worldwide. The data is from March 2010 to March 2011.

EUROPE

It seems that in Europe the undisputed sales leader is iOS, with almost 50% of sales. It is notable also the increase in sales that Android has achieved, reaching even to the BlackBerry´s sales. In the last nine months, Android has grown in sales from 1% up to 20%. It seems that Android is becoming more and more powerful. In Europe, Symbian is suffering a notable decline.

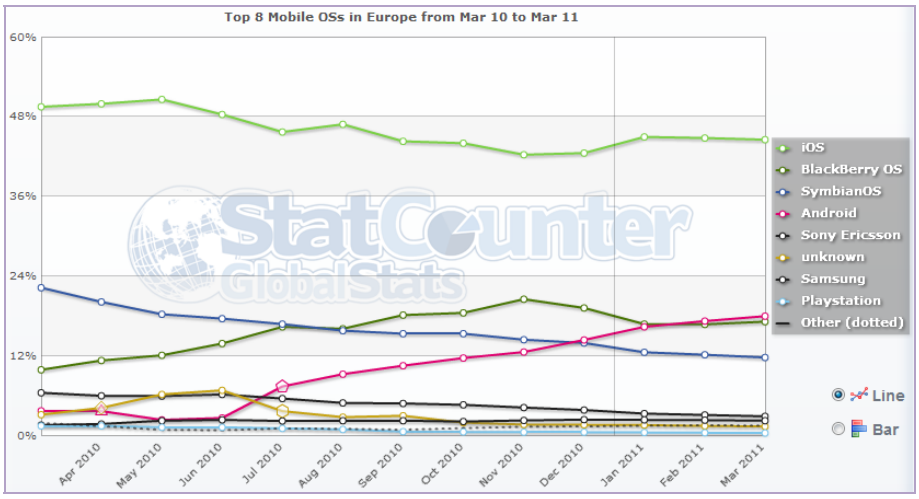


Figure 2.3 Top 8 Mobile OSs in Europe (03/10-03/11) **[15]**

WORLDWIDE

Moreover, worldwide, Symbian is the owner of the top market share around 32% last year. iOS has suffered a 10% drop in sales and Android, which is on the road to success, has reached 16% in just nine months.

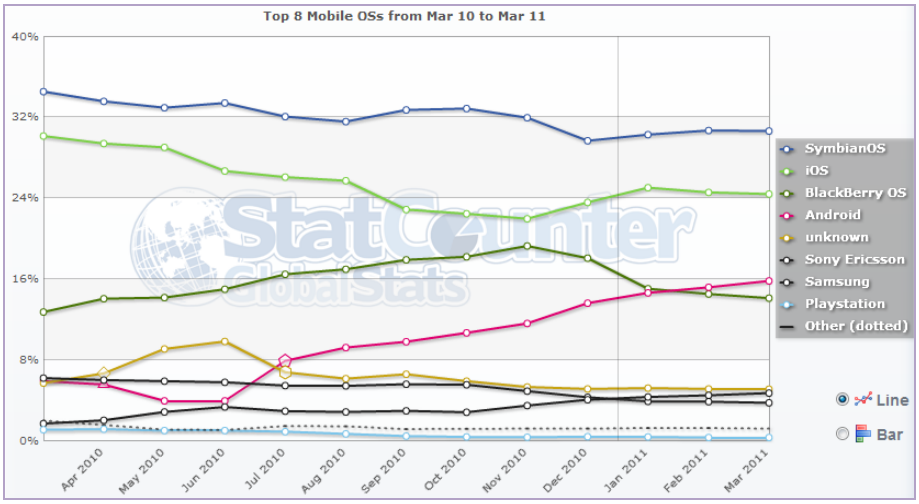


Figure 2.4 Top 8 Mobile OSs worldwide (03/10-03/11) **[15]**

In the following lines a brief explanation is given of each of the more important mobile Operating Systems that exist nowadays:

Symbian and Meego:

Figure 2.5 Symbian

The most common and well-known operating systems are those responsible for the operation of the nokia phones. The main reason is because they have a really high market share [16].

Developed by Symbian, nowadays mostly used are the S40 and S60.

S40 is implemented in 'feature phones' and 'entry level', differing from the S60 (smartphones) in the access to Internet, limited or nonexistent, as they are not multi-tasking and that, in general, have no broadband connection (S40 are mobile generation 2 and 2.5, GPRS) .

In S60 it should be noted a version developed for touch screen mobile, called S60 R5.

Now, halfway between mobile phones and small laptops (notebooks, netbooks, smartbooks, MIDs, etc.) there is a new operating system led by Nokia and Intel: Meego [17]. This new operating system will combine Intel developed Moblin and Nokia developed Maemo. Its main characteristic is that it is an open system.

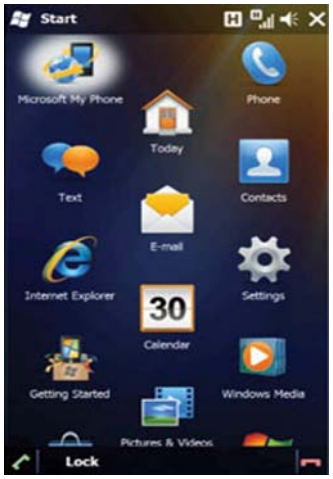
Blackberry:

The owner of this operating system is Research In Motion (RIM). This OS is adapted to their different devices.

Its secret, as is known, is the optimization of the network and having an open line continuously, making its e-mail management, instant messaging or network, exceptional [18].

Figure 2.6 Blackberry

Windows:



Before 2010, Microsoft offered Windows Mobile, focused especially on customers with professional profile. From that year, a revolution happened because Microsoft was diversified and began marketing two new platforms for Smartphone's. Far from the professional environment, they are intended to be used by general public, basically uploaders (young people who like to be continuously connected with friends and upload content). So Windows Mobile is now called Windows Phone and currently it is available for the users Windows Phone 7 [19].

Figure 2.7 Windows

iPhone:

Figure 2.8 iPhone

iOS (known as iPhone OS prior to June 2010) is Apple's mobile operating system. Originally developed for the iPhone, it has been extended to support other Apple devices such as the iPod touch, iPad and Apple TV.

The user interface of iOS is based on the concept of direct manipulation, using multi-touch gestures. Interface control elements consist of sliders, switches, and buttons. The response to user input is immediate and provides a fluid interface. Interaction with the OS includes gestures such as swipe, tap, pinch, and reverse pinch. All of them have specific definitions within the context of the iOS operating system and its multitouch interface. Internal accelerometers are used by some applications to respond to shaking the device or rotating it in three dimensions.

Four versions that have been implemented: iPhone, iPhone3, 3G and 4. After launching last summer its revolutionary iPhone model 4, which, despite being a bestseller, has led a number of technical issues that have led to protests among users. Despite the coverage problems resulting from its antenna, Apple considers the iPhone 4 has been a milestone in the interaction of mobile telephony and network access.

Apple is working on the new version of the iPhone, iPhone5. At the present moment, the launched of iPhone5 seems as the best kept secret ever by Apple [20].

Android*:*

Figure 2.9 Android

Some companies related in the sector (operators, manufacturers, semiconductor companies etc.) joined to formed the Open Handset Alliance (OHA). Google was the promoter. In this context Android was born, an open operating system based on Linux, designed to be free for the users to decide how and why they would like to use their phones, making feasible to design the applications that they want to have.

Android is multiscreen, it 'feeds' of its own software market and is designed for tactile displays… At this time, the OS is considered to be more flexible and easy to use than all other existing OSs.

Currently, there are eight versions. Last six are known for their 'sweet' names: Cupcake 1.5, Donut 1.6, Eclair 2.0/2.1, Froyo 2.2, Gingerbread 2.3 and Honeycomb 3.0.

It should be noted that there are three types of Android phones: Google Phones (available from Google), the Google Experience (it has integrated the basic operating system, with the most typical services of the famous searcher) and custom Android, which are those to which manufacturers incorporate a user interface developed by themselves and identifying differences with the product and seeking loyalty to the brand. It is the case of HTC Sense, MotoBlur Motorola, Acer UI 30, etc. [3].

2.5 Android

2.5.1 The Android Platform

Android is a software environment built for mobile devices. It has to be mentioned that it is not a hardware platform. Android includes a Linux kernel-based OS, a rich UI, end-user applications, code libraries, application frameworks, multimedia support, and much more. While components of the underlying OS are written in C or C++, user applications are built for Android in Java [3].

2.5.2 Android Developers

The application programs (“apps”) are written by a large community of developers which are responsible of extending the functionality of the devices. Currently there are approximately over 200,000 apps available for Android [3].

2.5.3 Android Market

Figure 2.10 Android Market

The online app store run by Google is Android Market is. Anyway, apps can also be downloaded from third-party sites.

In 2007 the Open Handset Alliance (OHA) foundation announced the distribution of Android. OHA is a consortium of 79 hardware, software and telecom companies devoted to advancing open standards for mobile devices. Google released most of the Android code under the Apache License, a free software and open source license.

The Android operating system software stack consists of Java applications running on a Java-based, object-oriented application framework on top of Java core libraries running on a Dalvik virtual machine featuring JIT compilation. Besides, libraries (written in C) include the surface manager, OpenCore media framework, SQLite relational database management system, OpenGL ES 2.0 3D graphics API, WebKit layout engine, SGL graphics engine, SSL, and Bionic libc [3].

2.5.4 Dalvik Virtual Machine

The virtual machine of Android is Dalvik. It is an integral part of this OS, typically used on mobile devices such as mobile phones, tablet computers and netbooks. As the rest of Android, it is also open-source software [21].

Before execution, Android applications are converted into the compact Dalvik Executable (.dex) format, which is designed to be suitable for systems that are constrained in terms of memory and processor speed.

There are some important facts about the Dalvik VM that are important to understand some procedures in the development of this Master Thesis. They will be explained below.
As mentioned, Dalvik VM executes files in the Dalvik Executable format (.dex) which is optimized for minimal memory footprint. The VM is register-based, and it is able to run classes compiled by a Java language compiler that have been transformed into the .dex format by the included "dx" tool.

In order to conserve space, duplicate strings and other constants used in multiple class files are included only once in the .dex output. Java bytecode is also converted into an alternate instruction set used by the Dalvik VM. An uncompressed .dex file is typically a

few percent smaller in size than a compressed .jar (Java Archive) derived from the same .class files.

Even as said before the tool called “dx” is used to convert Java .class files into the .dex format, it can´t do the conversion of all of them. That has been a handicap in this project because it was difficult to use some Java libraries in the Android project. As far as these Java libraries can´t be converted to .dex format, using these libraries was impossible. It was necessary to solve a critical need of the libraries for Parsing and Encoding the HL7 messages. A new Parsing and Encoding system for this Android Project was needed to be realized. More information about this issue is illustrated in *Annex VI. Java vs. Android*.

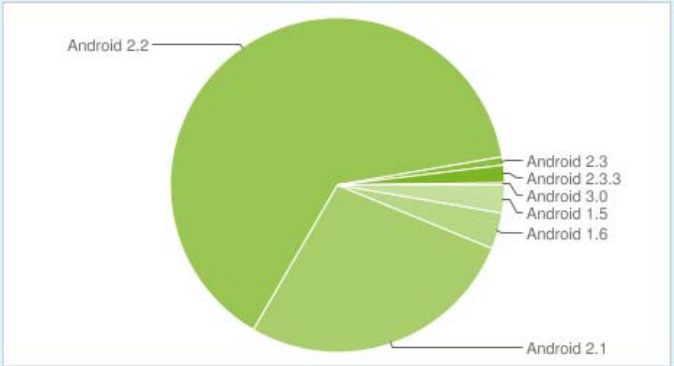
Nowadays, there aren´t open source libraries available for doing the Parsing and Encoding of HL7 messages in Android. This was the first step to be taken to be able to work with HL7 messages.

2.5.5 Platform Versions

This section shows data about the relative number of active devices running a version of the Android platform. It illustrates the landscape of device distribution and so, it is possible to deduce how to prioritize the development of the application features for the devices currently in the hands of users [3].

Current Distribution

The following pie chart and table is based on the number of Android devices that have accessed Android Market [3].



Platform	API Level	Distribution
Android 1.5	3	2.7%
Android 1.6	4	3.5%
Android 2.1	7	27.2%
Android 2.2	8	63.9%
Android 2.3	9	0.8%
Android 2.3.3	10	1.7%
Android 3.0	11	0.2%

Data collected during two weeks ending on April 1, 2011

Figure 2.11 Current distribution of Android Platform versions **[3]**

Historical Distribution

In the next Figure 2.12, it is shown a history of the relative number of active Android devices running different versions of the Android platform. It also provides perspective of how many devices an application is compatible with, based on the platform version.

It indicates the total percent of active devices that are compatible with a given version of Android. For example, if an application is developed for a version that is at the very top of the chart, then this application is compatible with 100% of active devices (and all future versions), because all Android APIs are forward compatible. In the other hand, if an application is developed for a version lower on the chart, then it is currently compatible with the percentage of devices indicated on the y-axis [3].

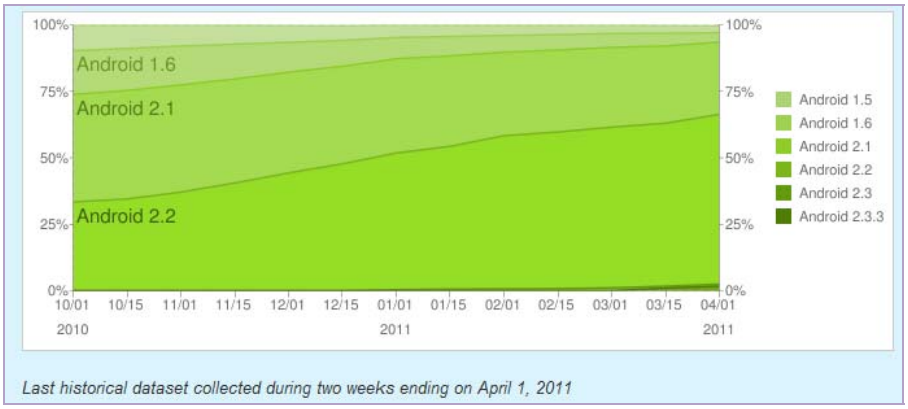


Figure 2.12 Historical distribution of Android Platform versions **[3]**

2.6 IDE Integrated Development Environment

The three most popular integrated development environments currently in Java are: Intellij Idea [22], Netbeans [23] and Eclipse [6].

Intellij Idea

It was created by JetBrains and supports these languages: Java, JavaScript, HTML/XHTML/CSS, XML/XSL, ActionScript/MXML, Python, Ruby/HRuby, Groovy, SQL, and PHP. But it is not open source [22].

Netbeans and Eclipse are both open source. Comparing these IDEs, we can explain some advantages or disadvantages.

Netbeans

- Advantages:
  - Netbeans has a GUI builder which is extremely helpful for rapidly creating java applications.

- Netbeans has support for languages like PHP, C++, RoR and Grails.

- Disadvantages:
  - It lacks the easy integration of SDKs like the Blackberry SDK and Android SDK.

Eclipse

On the other hand, Eclipse IDE:

- Advantages:
  - There is easy integration for things like flex builder, the blackberry SDK, and the Android SDK in the form of plugins.
  - Eclipse also has a support for languages other than Java, like C++, and PHP.
  - Eclipse has tons of plugins which are easy to install.
- Disadvantages:
  - Eclipse tended to have a slower startup in comparison to Netbeans.

Both of them have their own advantages comparing with the other. The use of Netbeans seems to be better for Java applications, because of the GUI builder.

It is said that the use of Netbeans for Android is possible but not recommended in all respects. It is possible to use *nbandroid*, a project in Netbeans ("plugin-netbeans"), enabling to work with Android.

Nevertheless, of course there are remarkable advantages if you work with Eclipse. In it, all the needed things are integrated. For example, the DDMS [24] (for debugging errors and much more) is part of the Eclipse´s views [25]. In Netbeans DDMS needs to be executed on its own (this application is in the folder /tools of Android SDK).

In Eclipse a lot of useful tools can be found and facilities for development, such as the using of XML for the graphical interfaces. It has a view where it can be seen how the screen is going to look like, without the need to use the phone or the emulator. Another important thing is that Eclipse is capable of detecting the errors in XML because it knows which attributes the interface has and the values of each of them. So, it can notify if there is an error.

When you connect the mobile with the cables both IDEs are capable to detect it properly.

Besides all this, it is good to know that in the official Android Developers webpage they recommend to install the ADT Plugin for Eclipse [26]. In the next paragraph it will be explained what has to be done to start working with Eclipse as a development environment for Android.

2.7 Android development environment working with Eclipse

To get the Android development environment working with Eclipse, two components are needed to install on the development system:

- The Eclipse Android Developer Tools (ADT).
- The Android SDK.

The Eclipse Android Developer Tools are Android development tools specifically created for the Eclipse IDE. The Android SDK is the standard Android SDK that comes from Google. Both are needed to develop Android applications with Eclipse [27].

2.7.1 The Android SDK

The Android SDK (Software Development Kit) includes sample projects with source code, development tools, an emulator, and required libraries to build Android applications.

All the information about installing the Android SDK is in the Android Developers webpage [27] and [28]. There, it is possible to learn how to prepare the development computer and how to download the SDK Starter Package.

2.7.2 Developing in Eclipse with ADT Plugin

The Android Development Tools (ADT) plugin for Eclipse adds powerful extensions to the Eclipse integrated development environment. It enables creating and debugging Android applications easier and faster [26].

If Eclipse is used, the ADT plugin gives an incredible boost in developing Android applications. For example:

- The developer can execute other Android development tools from inside the Eclipse IDE. For instance, ADT allows accessing several capabilities of the DDMS tool: take screenshots, set breakpoints, view thread and process information directly from Eclipse etc.
- Thanks to the Project Wizard, the creation and setting up of all of the basic files needed for a new Android application are created.
- The process of building the Android application is simplified.
- The Android code editor helps writing valid XML for the Android manifest and resource files.
- It will even export the created project into a signed APK, which can be distributed to users.

[26]

3 Development and Implementation

Figure 3.1 Connections of the Android Server

3.1 Connections between HDH and Android Server

In the application a connection between the HDH wrist device and the mobile phone was developed by three types of wireless technologies: Bluetooth, Wi-Fi and GSM.

3.1.1 Bluetooth Technology

Figure 3.2 Bluetooth connection HDH-Android Server

The first is using Bluetooth technology. On one hand, the Android phone can act as a server, for example waiting for incoming connections from various HDH; and on the other hand can act as a client, because the program connects to the selected HDH (which will start transferring HL7 messages).

The package responsible for the Bluetooth connection is **com.upna.AndroidServer.Bluetooth**. In this package are stored eight Classes described below.

When you have started the application, and you have chosen the HDH Server button in the **AndroidServer_Home** activity, you go inside a new activity called **ServerW3GorBT** where two buttons are located related with the two servers: Bluetooth Server and Wi-Fi Server (see Figures 3.3 and 3.4).

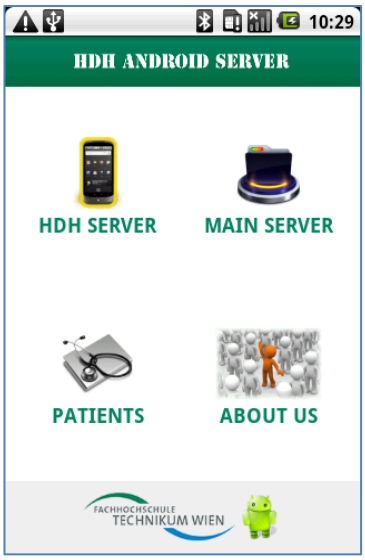


Figure 3.3 Bluetooth Manager

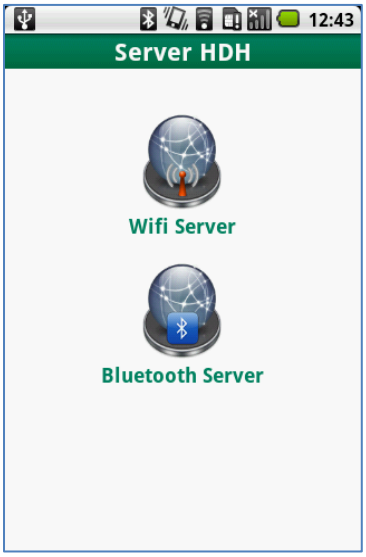


Figure 3.4 ServerW3GorBT

Figure 3.5 Bluetooth Manager

In addition to the two server functions of the activity shown above, in the Bluetooth Manager activity (Figure 3.5) we can also see the mobile function as a client. For this, we only have to push up the hidden menu clicking the menu button in the mobile phone (See Figure 3.11).

Currently, the developed Android Server application connects to the HDH as a client (not as a server). The reason for this is explained in greater detail in chapter *Android mobile working as a client with regard to the HDH device.*

In the lines below, it is described how the Bluetooth connection (where the Android mobile is the Server) works.

Android Mobile as a Server

The Server [29] is executing as a Service [30] in the application. It provides many advantages in comparison with executing as an Activity [31] or executing as a Thread [32].

Opposite of the Activity, the Service allows to the user to write a SMS, receiving phone calls etc. while receiving HL7 messages. If the application had been developed as an Activity, the reception of the HL7 messages should have been in the life cycle of the

Activity (please see Annex VII. Activity) preventing the user to use the phone for other purposes during the execution of the program.

Otherwise, if the application had been developed as a Thread, the mobile could not run Notifications thereby preventing to show notifications when Alert messages were received. Notifications [33] can only be launched from Activities or Services.

For all these reasons, the application has been developed as a Service.

Following concept map illustrates the Android Server Service developed in this Master Thesis by Bluetooth technology. It is a clarifying schema about the performance of the Service.

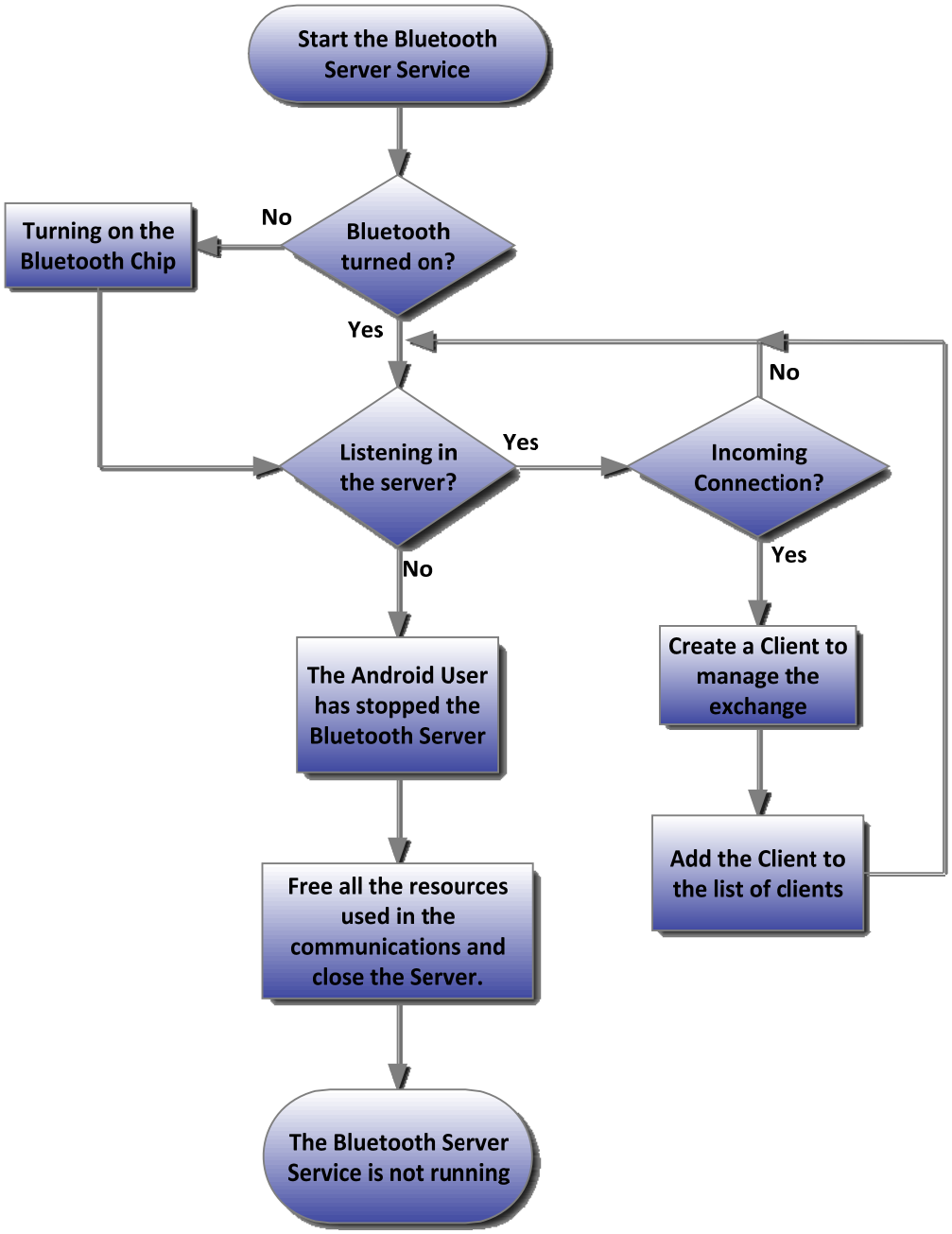


Figure 3.6 *Android Server Service performance schema. Bluetooth technology.*

*When you first press the button to start the Service, the server starts waiting for incoming connections. Meanwhile in the Activity related with this Service (**BluetoothManager**) a progress bar (more about the performance of the progress in the chapter Progress Bar) switches on and also a notification to advise the user that the service is running (see Figure 3.7).*

Figure 3.7 Service On button pressed first time. Service starts running

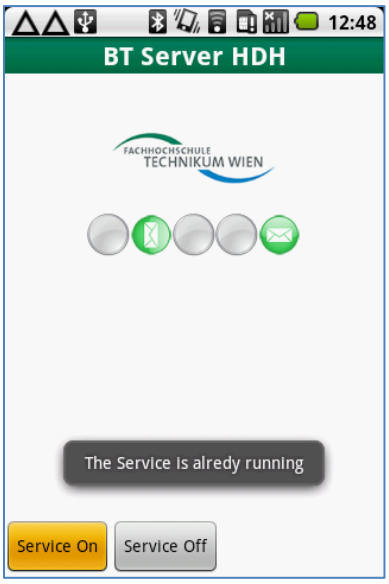


Figure 3.8 Service On button pressed for second time. Service is already running

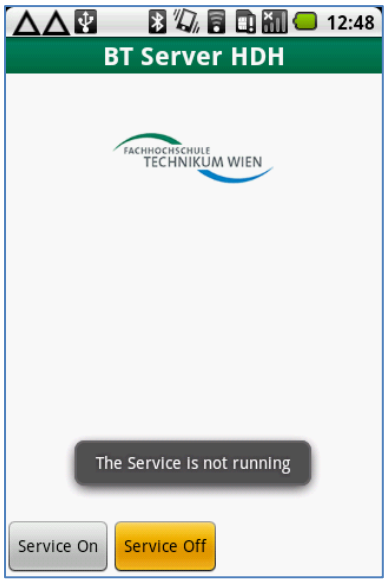


Figure 3.9 Service Off button pressed. Service finished

*Every time the user goes again to the **BlueoothManager** Activity to see the status of the Service in that moment, the following method is called: **stateService**. It checks if the service that the user is looking for is the same as one of the services which already are working. If the service is already working the progress bar will continue on. In addition, every time you try to turn on or off the server, the application displays a message warning the user. This is the code that lists the services available on Android:*

```
private boolean stateService(){
    // Check if the service is running
    ActivityManager activityManager = (ActivityManager) getSystemService(ACTIVITY_SERVICE);
    List<ActivityManager.RunningServiceInfo> runningServices = activityManager.getRunningServices(50);

    for (int i = 0; i < runningServices.size(); i++) {
        ActivityManager.RunningServiceInfo runningServiceInfo = runningServices.get(i);
        if (runningServiceInfo.service.getClassName().equals(getString(R.string.hdhmanagerservice))){
            isRunning=true;
        }
        Log.i("BluetoothManager", "Process " + runningServiceInfo.process + " with component " +
                runningServiceInfo.service.getClassName());
    }
    runningServices.clear();
    return isRunning;
}
```

*When you press the button "Service ON ", it makes a reference to **HDHManagerService** class where the service is located, and the life cycle of the service starts.*

*During the life cycle, firstly is checked if the device where the software is installed supports the Bluetooth technology [34]. If so, a notification showing that the server is listening is created as well as an object of the **BluetoothHDHService** class (where a Thread starts where the connections between the HDH and mobile are in "accept"*

*mode). Added to this object there are: the **Context** [35] of the Service and a **Handler** [36] type object. (This handler will enable the communication between the **ListenThread** and the Bluetooth Service. All about this handler is explained later).*

*After we have created the **BluetoothHDHService** object, the **ListenThread** starts running waiting for the incoming connections.*

```
public void run()  //Android as Server
{
    setName("ListenThread");
    try
    {
        while(!this.finish)
        {
            BluetoothSocket socket = mmServerSocket.accept();
            if(socket != null)
            {   // Pass the socket to the "Talking" thread and add to the queue
                HDHClient client = new HDHClient(socket, mHandlerService);
                clients.addClient(client);
            }
        }
    // Close the server
    mmServerSocket.close();
    }catch(IOException e){
        Log.e(TAG, "IO error: " + e.getMessage());
    }catch (Exception e) {
        Log.e(TAG, "Unexpected error: " + e.getMessage());
    }
}
```

*As shown, there is a while loop which allows the connections with a lot of HDH clients until the Server is switched off. The **accept** of the **BluetoothServerSocket** [37] class is a "blocker" instruction (it means that the accept will wait in this code line until a connection from outside is done) and then it will return a Bluetooth Socket which identifies the connection. This instruction, **accept**, avoids the CPU of the Android Server to break, because being a blocker instruction prevents that the CPU achieves 100% of constant load, which can lead to damage the CPU and shorten the battery life.*

*When the Bluetooth Socket [38] is obtained, it is checked if it has been created correctly and an **HDHClient** object is created. This object will be responsible for the communication with the wrist device by a Thread, and it is added to a List where all other currently connected clients are listed. When this is done, the code will come back to the **accept**, which will wait for other connections.*

*The List of all the patients is controlled by the **HDHManagedClients** class. This class is responsible of adding new clients, or delete and free some of the resources used in the communication.*

In this way, the developed application becomes in a multi-user application, achieving successfully one of the targets given for this project.

After this explanation of the Listen Thread and the clients acceptance, the issues related with the accepted client and the mobile phone will be introduced. As mentioned before (shown in the code above), this communication is done within the ***HDHClient*** *class.*

Below is shown ***ReceiverThread*** *included in the* ***HDHClient:***

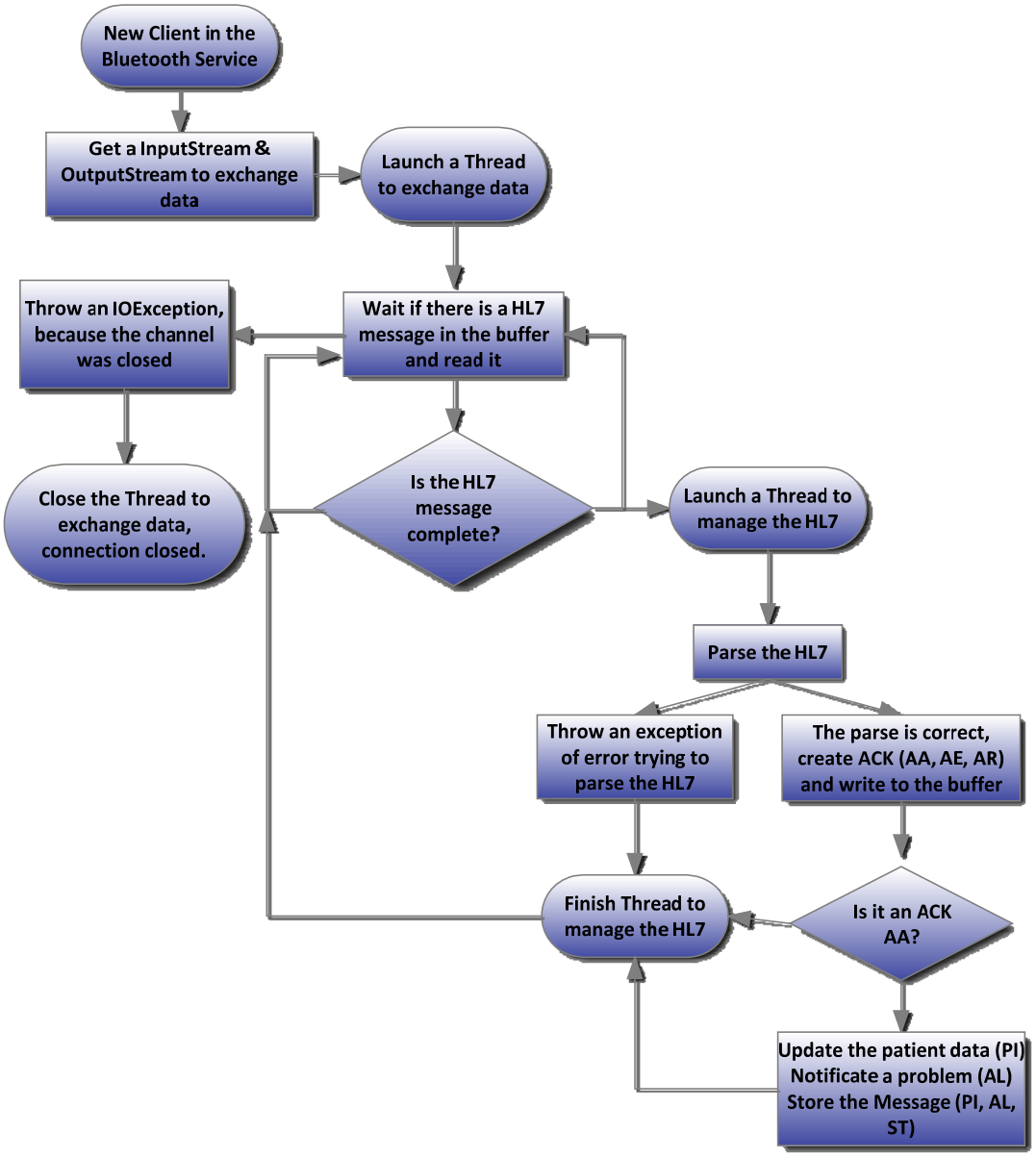


Figure 3.10 *Schema of HL7 message reception management*

*The code related with the **ReceiverThread** is:*

```
public void run() {
    setName("ReceiverThread");
    while(!this.finish){
        try {
            int bytesAvailable=1;
            String received="";
            while(bytesAvailable>0){
                byte[] buffer = new byte[1024];
                int readed = mSocket.getInputStream().read(buffer, 0, buffer.length);
                Log.d(TAG, "bytes readed "+String.valueOf(readed));
                Log.d(TAG, "Take it from the buffer: "+new String(buffer));
                Log.d(TAG, "ZZzzZZZZ");
                Thread.sleep(1000);
                Log.d(TAG, "Wake Up");
                bytesAvailable=mSocket.getInputStream().available();
                Log.d(TAG, "bytesAvailable without reading "+String.valueOf(bytesAvailable));

                String received1=new String(buffer).substring(0, readed);
                String received2=ParcheSubstring(received1);
                received=received+received2;
            }
            Log.d(TAG, "Received final: "+received);
            // Remove the null pointer
            received= received.substring(0, received.length()-1);
            HL7Manager hl7=new HL7Manager(received);
            hl7.start();
            Log.d(TAG, "Launching HL7Manager Thread");

        }catch (NullPointerException e) {
        }catch (IOException e) {
            // Disconnect Event
            Log.d(TAG, "EOF readed because channel is closed");
            break;
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    Log.d(TAG, "Receiver thread exiting.");
}
```

This Thread is in charge for the reception on the HL7 messages. In the image before, in purple, is shown the part which is working with the reading of the buffer.

*The reader process performs the following actions. The **read** instruction of the **InputStream** [39] class of Android, blocks the thread until there is something new in the buffer. When new data arrives, it reads from the buffer and returns the bytes.*

*Then, the thread is forced to **sleep**. That is done because the arriving message can be separated in different parts in the reception moment. By this way, it tests if there is something more in the buffer and if so, it reads and adds to the part of message received before. (The code is able to delete all the blank spaces that could have been introduced in the buffer between the several parts of the message, this action is done by **ParcheSubstring**).*

In this way, the application is capable of receiving and rebuilding perfectly the entire HL7 message.

***Note:** the **read** can detect if the connection is closed and if so, an **IOException** is given. It enables breaking the reception loop and **closing** the Thread.*

*When the reception of the message is completed, the message is parsed. This action is performed in one Thread apart from the reception. This Thread, whose name is **HL7Manager**, is included in the class **HDHClient.***

Thus, each Thread works with its own task making the code more efficient (since a new Thread is opened every time there is a message to parse, and it will be closed when finished working with this message).

The reception of the messages happens one by one, i.e. every time a new message is received the mobile sends an ACK message for this message. If something goes wrong and the message is not received, the HDH wrist device will resend again the message seconds later.

This requires that the communication sending HL7/receiving ACK should happen in shorter time than the timer required for the retransmission. The process of the reception, parse and sending of the ACK depend on two variables: the length of the message and the size of the buffer in the reception. More or less, the average of the message length is 1024 characters, that is why the buffer selected has 1024 bytes capacity. However, if the received message is bigger than the buffer, it will read new 1024 character to complete the message.

```
public void run() {
    setName("HL7Manager");
    try{
        HL7object hl7=new HL7object(HL7Message);
        Log.d(TAG, "HL7 parse susccessful");
        ACK ack=ACK.create_ACK(hl7);

        if (ack.getAck_msa().get_acknowlegment_Code()=="AR"){
            this.writeBT(ack);
            Log.d(TAG, "Writting ACK AR");
        }
        if(ack.getAck_msa().get_acknowlegment_Code()=="AA"){
            SDCard.writeMessage(HL7Message, hl7);
            this.writeBT(ack);
            Log.d(TAG, "Writting ACK AA");
            if (hl7.getMsh().get_messageControlID().substring(0, 2).equals("PI")){
                UpdateXml upxml=new UpdateXml(hl7);
                upxml.refreshDataInXml();

            }if (hl7.getMsh().get_messageControlID().substring(0, 2).equals("AL")){
                Alert al=new Alert(hl7, mHandlerService);
                al.start();
                Log.d(TAG, "Launch Alert Thread");
            }
        }
    }catch (Exception e){
        Log.e(TAG, "Can't Parse HL7 message");
        return;
    }
}
```

When the reception of the entire message is completed, the next step is to parse it (more information about the parser in 3.3.3 Parser: from HL7 message to HL7). In case of a correct parsing of the message, the ACK type HL7 message is created and the code of the acknowledgement is analyzed: if it is "AA", the received message is stored in the SDCard (messages ST, AL and PI) and the created ACK is written in the output buffer. Then, the patient data is stored by ***UpdateXml*** *(if the message is "PI" type) or an alarm is shown (if the message type is "AL") and the Thread is closed.*

If the message type is "AL", another Thread is launched called ***Alert*** *where it is needed to launch an alert notification. As mentioned before in the report, it is very important to know that the notifications only can be launched only by Services or by Activities (not by Threads) [33]. That is why it is necessary to use an object called Handler in the Service* ***HDHManagerService****. This Handler will be a parameter of all the Threads involved in the communication. Through the Handler, it is possible to send the HL7 object created from the received AL message. Below this lines show what is done in the* ***Alert*** *Thread.*

```
public void run(){
    setName("Alert");
    // Send to the service one hl7 alarm to display
    this.mHandlerService.obtainMessage(0, -1, -1, hl7).sendToTarget();
}
```

As mentioned before, the message is received in the Handler object of the ***HDHManagerService*** *as shown:*

```
public Handler mHandlerService=new Handler(){

    @Override
    public void handleMessage(Message msg) {
        // TODO Auto-generated method stub
        super.handleMessage(msg);
           switch (msg.what) {
            case 0:
                HL7object hl7=(HL7object) msg.obj;
                launchNotification(hl7);
            break;
            }
    }
};
```

The method ***launchNotification*** *in the* ***HDHManagerService*** *class is liable launching the alert notifications. In this process, it must be kept in mind that:*

- *Each notification must have one identification number so that is clearly distinguished from other notifications launched by the system.*

- *The notifications have to be launched from a Service or Activity which are working, that is why is necessary to use a Handler in order to send the HL7 object from the reception Thread to the running Service.*
- *Each notification must have a PendingIntent object, which is created by a related Intent. In this Intent some useful data is added for the alarm activity, like the patient identifier number (Extras). By default, a distinct PendingIntent is only created when the underlying Intent is materially different from the Intent used by another outstanding PendingIntent. The used Intent only differs by the extras. If we need to have several outstanding PendingIntents, we will need to have them be on different Intents, where those Intents differ by component, action, data (Uri), or categories and not only the Extras. That is why each Intent is identified with a single category. This category is given by the moment of the creation of the Intent.*

Moreover, if the received message is "PI", what the application must do is update all the data stored in the XML file of the SDCard. This file is created to store all the patient data inside. That is why is very important to register every patient that will be with the monitoring system (this will be explained in chapter Register a patient). For the update there is a method ***refreshDataInXml*** *in the* ***UPdateXML*** *class, which uses the* ***simple-xml-2.3.6*** *library [40].*

After these paragraphs, the explanation is finished about the application procedure regarding the successful parsing of the message. In case of incorrect parsing or a thrown exception the following things occur.

The exception is caught and it will exit the Thread, waiting for the next message that will be sent when the timer expires in the sending device.

In summary, it is all about switching on the Service and sending and receiving of the messages. In the next lines, it will be illustrated in detail how it is possible to switch Off the Service.

In the lifecycle of a Service/Activity there is a method which always is executed before leaving the Context. This method is ***onDestroy*** *of the Service [30] / Activity [31] class of Android. It is called every time we want to switch Off the Service by using the instruction* ***stopService*** *(of the Service class of Android), which is executed when the button in the Figure 3.9 is pressed.*

In order to turn off the Service of the Bluetooth Server, we should firstly make sure that there is an object of ***BluetoothHDHService****, because as said before, one of the attributes of this object is the Thread "ListenThread". If* ***BluetoothHDHService*** *object is*

*available, its **stop** method is called to finalize the while loop in the ListenThread and close the ServerSocket.*

```
public synchronized void stop() {  //Android as Server
    // Stop the listening thread and free all the resources used by the communication
    if (mListenThread != null) {mListenThread.cancel(); mListenThread = null;}
    clients.freeAllResources();
}
```

After stopping of the thread, it is necessary to free all the resources used in the transmission of the all clients that we have in the queue of "clients".

*For this purpose, the method **freeAllResources** will be called in the class **HDHManagedClients**.*

```
public synchronized void freeAllResources () {
    Iterator<HDHClient> i = clients.iterator();
    while (i.hasNext()) {
        i.next().freeResources();
    }
    clients.clear();
}

public void freeResources() {
    if(receiver.isAlive())
        receiver.cancel();
}

public void cancel()
{
    this.finish = true;
    try {
        mSocket.close();
    } catch (IOException e) {
        Log.d(TAG, "close() failed");
    }
}
```

*The method **freeAllResources** will free one by one every "ReceiverThread" opened previously. For this, the method **freeResouces** of the **HDHClient** class is called, and it checks if the Thread is still running. If so, the Thread is stopped and the Socket of the communication is closed (if it is not already closed). As well, the while loop of the ReceiverThread is closed.*

In conclusion, all these issues constitute the start, transmission and the finishing of the application of the mobile Bluetooth server [29].

Android mobile working as a client with regard to the HDH device

First of all, it has to be clarified that the final version of the Android Server application that was developed should work as described in the previous section: Android mobile phone working as a Server regarding to the HDH device.

Some extra functions needed to be implemented in the code in order to have an effective application running with the current HDH device prototype.

The first step will explain why a client is needed which will connect with the HDH wrist device. The behavior of the Client will be illustrated.

At the beginning, both devices, the HDH wrist device and the Android mobile, should be paired so they know each other and can exchange data. All the communications that will use Bluetooth are going to use the Serial Port Profile.

After pairing, the user can choose one of these possible actions from the menu in the **BluetoothManager** activity: by choosing *Connect a device* the mobile can connect to the HDH and start the messages transfer, by choosing *Make discoverable* the user can make the mobile phone discoverable and at last, by pressing *Disconnect* the user can finish a connection with the HDH.

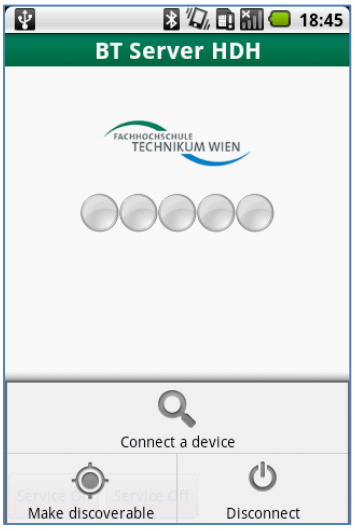


Figure 3.11 Menu button pressed

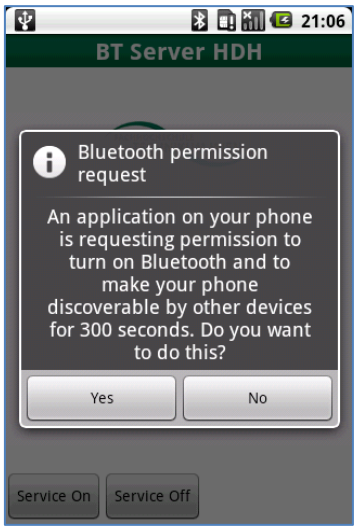


Figure 3.12 Make discoverable option clicked

The option ***Make discoverable*** allows turning On the Bluetooth (if it was switched Off) and it makes the mobile discoverable for 300 sec. Below some of this code is shown:

```
private void ensureDiscoverable() {
    if (mBluetoothAdapter.getScanMode() !=
        BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE) {
        //if the device is transparent for other devices
        Intent discoverableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
        // Make it discoverable for 300 seconds
        discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
        startActivity(discoverableIntent);
    }
}
```

The option ***Connect a device*** shows a sub activity (dialog type) which displays all paired objects so the user can scan the devices inside the Bluetooth range. In this screen (Figure 3.13) the user can select the device with he wants to be connected for transmit the HL7 messages.

Figure 3.13 Select a device to connect

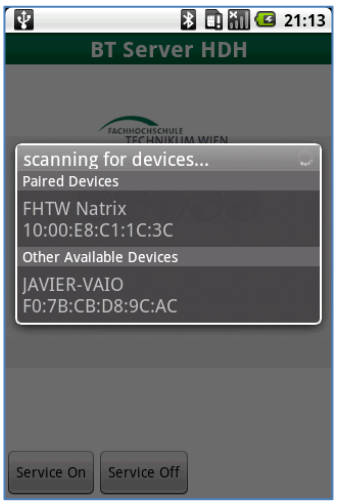


Figure 3.14 Scanning for devices

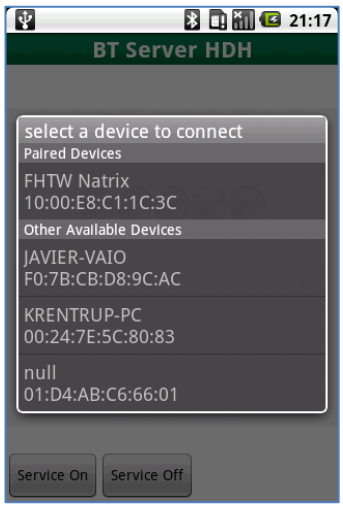


Figure 3.15 Select a device to connect

In these screenshots, it is possible to see also how the Android Server can manage the discovery of new devices.

The scan of the devices is done in the **DeviceListActivity** class, included in the **com.upna.AndroidServer.Bluetooth** package. An essential scheme about it is the following:

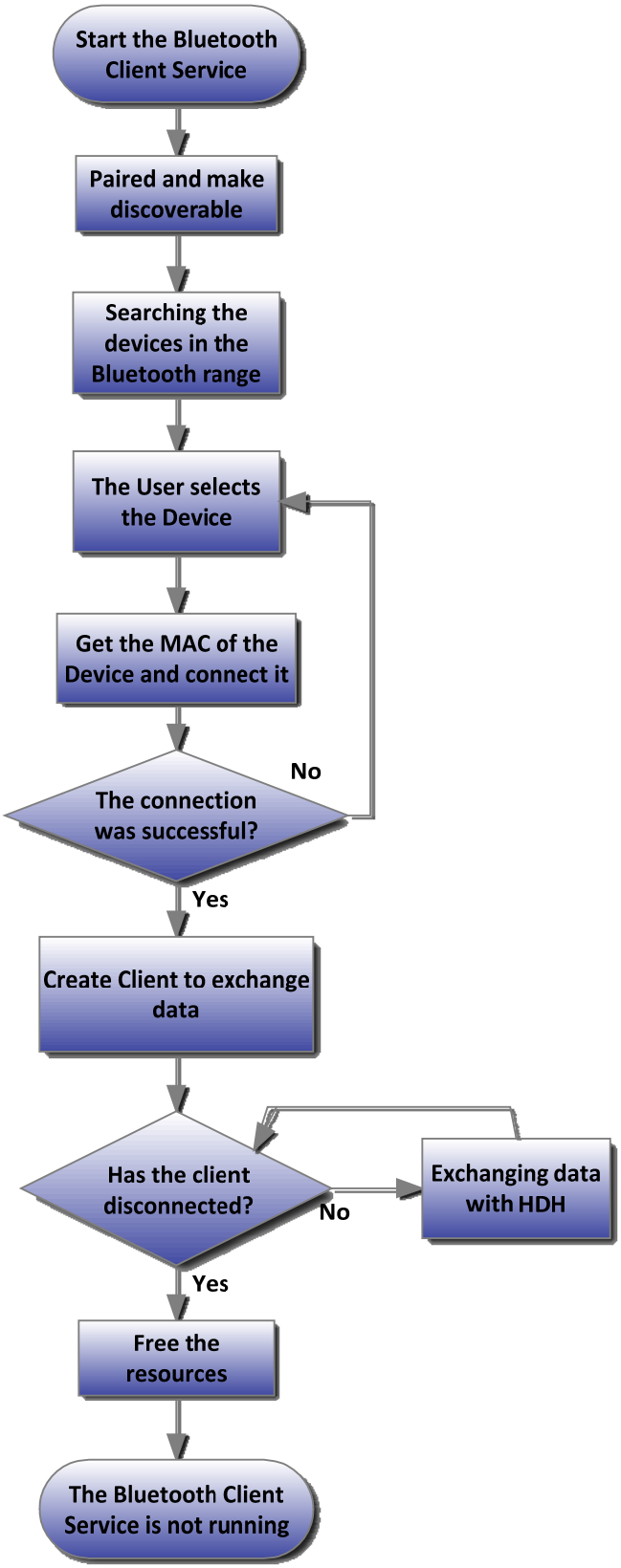


Figure 3.16 Android Server schema acting as a Client. Bluetooth technology.

When a device is selected from the **DeviceListActivity**, its MAC Address will be sent to the **BluetoothManager** class using the method **onActivityResult** (of the Acitivity class).

```
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    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);
            Intent intent=new Intent(this, HDHClientService.class);
            Bundle b= new Bundle();
            b.putString("Address", address);
            intent.putExtras(b);
            startService(intent);
        }
        break;
    }
}
```

After getting the MAC Address, the **HDHClientService** Service starts. It will deal with the communication with the selected device.

Among other tasks, this Service checks if there is a Bluetooth profile in the mobile and creates an object of **BluetoothHDHService** type (as when it acts as a server). The difference with respect to the Android Server Bluetooth is that now the mobile mustn´t wait for incoming connections but it makes the connection. For that, it is created an object of **ConnectThread** that creates a Socket for the communication and starts a thread "ConnectThread ".

```
public ConnectThread(BluetoothDevice device) {
    mmDevice = device;
    BluetoothSocket tmp = null;
    try {
        // Create a socket for the connection
        tmp = device.createRfcommSocketToServiceRecord(MY_UUID_Serial_Port);
    } catch (IOException e) {
        Log.e(TAG, "create() failed", e);
    }
    mmSocket = tmp;
}
```

As shown above, an object of BluetoothSocket is created, which is distinguished by the device and the Service type (announced from the HDH, MY_UUID_Serial_Port).

The UUID [41] is an immutable representation of a 128-bit universally unique identifier (UUID).
There are multiple, variant layouts of UUIDs, but this class is based upon variant 2 of RFC 4122, the Leach-Salz variant.

Considering that the communication is done by the Serial Port profile, the UUID selected should represent it. In our case, UUID is 00001101-0000-1000-8000-00805F9B34FB [42].

Once the Socket is created with the necessary information to do the connection, the thread “ConnectThread” is launched. The related code is this:

```
public void run() {

    Log.i(TAG, "BEGIN mConnectThread"+mmDevice.getAddress());
    setName("ConnectThread");
    // Cancel the discovery that it was launched before cause the device has been selected
    mAdapter.cancelDiscovery();
    try {
        Log.i(TAG, "Trying to connect");
        mmSocket.connect();
    } catch (IOException e) {
        try {

            Log.e(TAG, "error: connect failed");
            if (mConnectThread != null){
                Log.d(TAG, "ZZzzzZZZ");
                ConnectThread.sleep(30000);
                Log.d(TAG, "Wake Up");
                mConnectThread.cancel();
                BluetoothHDHService.this.connect(mmDevice);
            }
        }
        catch (InterruptedException e3) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            Log.e(TAG, "Thread didnt sleep", e3);
        }
        return;
    }
    // Pass the socket to the "Talking" thread and add to the queue
    Log.d(TAG, "Successful Connection ");
    HDHClientforServer client = new HDHClientforServer(mmSocket, mHandlerService);
```

As shown here, after the **connect** of the **BluetoothSocket** class is executed, if the exception isn´t caught (there wasn´t any errors while trying to connect), an object of **HDHClient** is created and added to the clients queue. It is exactly the same behavior like in the Bluetooth Android Server. Otherwise, if we catch the IOException there are several tries to reconnect, more specifically, it tries to reconnect every 10 seconds.

After this action, everything about the reception, parse, performance with all the types of messages and sending of the ACK messages, is the same as the Bluetooth Android Server (see the section *Android Mobile as a Server*).

However, in case that in the Thread that manages the HL7 messages the channel is closed, there are some differences in the behavior of this system. In the **HDHClientforServer**, it is caught an Exception because the device got out of the Bluetooth range. Then, we close the communication and try to reconnect again.

Disconnect a client: As explained in **BluetoothManager** screen with pop-up menu (Figure 3.11) we have the option to disconnect the client. This option lets you stop the Service and free all the resources used in the communication, as well as stop the Service in the Bluetooth Android Server.

3.1.2 Wi-Fi Technology

Figure 3.17 Wi-Fi connection HDH-Android Server

As mentioned in the section of the connections (*3.1 Connections between HDH and Android Server*) the Android Server application developed, can use three different technologies for the reception and transmission of the HL7 messages: Bluetooth, Wi-Fi and GSM.

In this section a thorough explanation about the IP technology is given. Wi-Fi is used to implement a Server and by this a Service is created for the users.

The package which implements this Server is **com.upna.AndroidServer.Wifi** which contains 5 classes that will be discussed later.

As described in the Bluetooth chapter, these Services are shown in the **ServerW3GorBT** activity (screen where the user can access the Wi-Fi Server after pressing the HDH Server button in the AndroidServer_Home activity).

Once the button of the Wi-Fi Server is pressed in the **ServerW3GorBT** activity, Figure 3.4, the code acts depending on the network interface available. If there is one, a new activity appears: **IPServerActivity.** If not, the Figure 3.18 is shown.

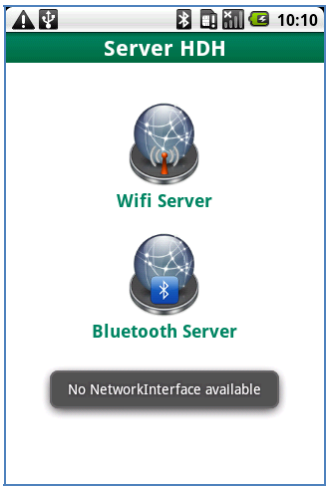


Figure 3.18 Network Interface not found

The discovering of the network is done in the first method of the lifecycle of this activity (**onCreate** of the Activity class of Android [31]) with the following code:

```
public String getLocalIpAddress() {
    try {
        for (Enumeration<NetworkInterface> en = NetworkInterface.getNetworkInterfaces(); en.hasMoreElements();) {
            NetworkInterface intf = en.nextElement();
            for (Enumeration<InetAddress> enumIpAddr = intf.getInetAddresses(); enumIpAddr.hasMoreElements();) {
                InetAddress inetAddress = enumIpAddr.nextElement();
                if (!inetAddress.isLoopbackAddress()) {
                    return inetAddress.getHostAddress().toString();
                }
            }
        }
    } catch (SocketException ex) {
        Log.e(TAG, ex.toString());
    }
    return null;
}
```

This method returns null or an available interface. If it is null, the activity is finished and the user will be notified that there is no interface available. (Figure 3.18). If there is one, the application will give an IP address (Figure 3.19) and when the user presses the Service On button the progress bar and the notification area will show that there is currently one Wi-Fi Server running (Figure 3.20).

Figure 3.19 IP address

Figure 3.20 Wi-Fi Android Server running

Turning On the Server Service

As shown in the previous figure, there are two buttons on the screen: one switches on the Server and the other switches it off.

Now the switch on is analyzed. As explained in the Bluetooth chapter, it was decided to run a Service in the Server because by doing so there are many advantages in the developed application.

Once the “Service On” button is pressed, the Service starts and these actions are performed next:

- Launch a notification to inform the user that the Service starts running.
- Creation of a Handler object which allows launching a notification from the Service.
- Creation of an object of the **HDHServer** class.

One of the attributes of the **HDHServer** object is another object which extends the Thread class “ListenThread”. It enables all the incoming connections. Its constructor is shown here:

```
private class ListenThread extends Thread {
    private final ServerSocket mServer;
    private boolean finish = false;
    private String TAG="ListenThread in HDHServer";

    public ListenThread() {

        ServerSocket tmp=null;
        try {
            tmp=new ServerSocket();
        } catch (IOException e) {
        // TODO Auto-generated catch block
            Log.d (TAG, "ServerSocket was not created");
        }
        mServer=tmp;
    }
```

As we can see every time an object is created, an instance of the **ServerSocket** class is done, so an object of the ServerSocket type is obtained [43]. After creating the object, the code of the thread is executed (**run** of the Thread class, as shown below). The mServer object must listen through the IP assigned by the network and the selected port.

```
public void run() {
    setName("ListenThread IP");
    try {
        // Set the server
        this.mServer.bind(new InetSocketAddress(HDHServer.this.ipaddress , 44444));
    }catch (IOException e) {
        // TODO Auto-generated catch block
        Log.d (TAG, "Not allowed to changing the IP and Port");
        return;
    }
    while(!this.finish){
        Socket socket;
        try {
            // Start listening
            socket = mServer.accept();
            if(socket != null) {
                // Save the client and start the receiver Thread
                HDHServerClient client = new HDHServerClient(socket, mHandler);
                clients.addClient(client);
            }
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            Log.d(TAG, "mServer.accept() failed");
        }
    }
    try {
        this.mServer.close();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
```

When an incoming connection is received in the Server, it is analyzed if the connection has been created correctly. Then, one **HDHServerClient** object is created in order to manage the connection and it is added to the clients-queue existing currently in the mobile.

Once the client is accepted, the code will wait for more incoming connections in the **accept** method of the **ServerSocket** class.

As already mentioned, the object **HDHServerClient** will manage the connection. It means it will be responsible for:

1. Reception of the HL7 messages sent by the HDH device.
2. Recognize and generate a corresponding ACK message.
3. Reply with the ACK.
4. Perform the necessary action in the received message, either storage the message, update patient information and warnings/alerts by using sound notifications.

All these tasks are maintained as follows: in the **ReceiverThread** class included in the **HDHServerClient** class (the tasks corresponding to 1, 3 and 4 point), the package with the Parser of the HL7 messages (point 2) and in the **IPServerService** by the Handler object and **Alert** class (task number 4, notifications), and the UpdateXML (task number 4, updating the patient data).

Turning Off the Server Service

Regarding to the Switch Off of the Service, it has the same features as the Service of the Blueetooth Server.

- Recognition of the state of the Service (Mode On and Mode Off), using visual effects such as the progress bar and Toast [44] type message (implemented in the class **IPServerActivity**).
- Free all the resources of the connections (implemented in classes and **HDHServerClient** and **HDHServerManagedClients**).
- The server won´t wait for more incoming connections (implemented in class **HDHServer**).

3.1.3 GSM Technology

Figure 3.21 *GSM connection HDH-Android Server*

The third mechanism implemented in the application to allow the reception of HL7 messages is GSM [29].

This scenario would be very useful and advantageous in situations where the Wi-Fi or Bluetooth systems are not able to address the situation, for instance if these services are out of range.

The functionality of the GSM is done in two packages in the developed code: ***com.upna.AndroidServer.Bluetooth*** *and* ***com.upna.AndroidServer.SMSManager****.*

In the Bluetooth package, there is the ***AndroidServerReceiver*** *class. One of the tasks of this class is to receive Broadcast messages from the mobile phone OS. The events from the Broadcast messages that the* ***AndroidServerReceiver*** *can handle are:*

- *Detection of the "Turning On" and "On Rebooting" of the Android Mobile.This action will cause that both services, the SMS reception and Bluetooth Server will start working.*

- *Reception of the GSM mobile technology's messages.This action enables the program accessing to the SMS messages and after sniffing the message, it is sent to the receiver in* ***SMSService****.*

Here part of the code of the ***AndroidManifest*** *will be illustrated [45] (see Annex III. Android Manifest) where the behavior is defined which is related with the actions that will be performed, as explained above.*

```
<receiver android:name="com.upna.AndroidServer.Bluetooth.AndroidServerReceiver"
          android:enabled="true">
    <intent-filter>
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
        <action android:name="android.content.Intent.ACTION_BOOT_COMPLETED"/>
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
        <category android:name="android.intent.category.HOME"/>
    </intent-filter>
</receiver>
```

The code written in the ***AndroidServerReceiver*** *class is responsible of executing the appropriate response to each of the events originated in Android (rebooting/turning on or reception of the messages).*

In case of the reception of a message SMS to the mobile, it will be captured to the ***AndroidServerReceiver*** *class, and it will be sent to the receiver type object of the* ***SMSService*** *class too.*

This way to manage the messages is one of the most important and essential procedures when we wanted to receive the message by GSM, and interact with each type of HL7 message (AL, PI or ST) because of the next reasons:

- *The SMS messages are captured only by the receiver of the application (****AndroidServerReceiver****), creating a communication between the Android operative system and the developed program.*

- *The classes which extend from **BroadcastReceiver** class, for instance **AndroidServerReceiver**, need a limited execution time [46], i.e. they can´t execute large code.*
- *Furthermore, in case of the HL7 alarm messages the notifications have to be launched to the user. This means that it has to be done from a service or activity as discussed in previous chapters [33].*

*Therefore, working in this context, arises the imperative need of sending the SMS message captured by the **AndroidServerReceiver**, by other message (Broadcast) to the receiver (mReceiverSMS) located in the **SMSService** class (see the code below). At the same time, this **SMSService** has to extend from Service class and in this way it is capable to launch alarm notifications.*

```
Log.d("AndroidServerReceiver", "Sending the sms to Reciver");
Intent intentSMS=new Intent(SMSService.SMS_RECEIVED);
Bundle b= new Bundle();
b.putString("SMS", str);
intentSMS.putExtras(b);
context.sendBroadcast(intentSMS);
```

Part of the receiver code (AndroidServerReceiver) responsible for sending SMS messages to the SMSService service.
The action associated with the intent is SMS_RECEIVED

```
private final BroadcastReceiver mReceiverSMS = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        Log.d("SMSService", "onReceive from SMSService");

        if (SMSService.SMS_RECEIVED.equals(action)) {
            String sms=intent.getExtras().getString("SMS");
            SMSService.this.analizeHL7(sms);
        }
    }
};
```

Part of the SMSService, where the SMS messages are received.

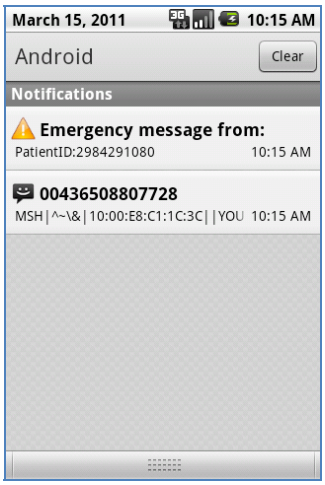


***Figure 3.22** Emergency message notification*

Obviously, the service of the reception of the SMS has to be running all the time, because the user isn´t allowed starting or finishing this service. That is why it is necessary to interpret the event in the mobile operative system that shows that the

mobile has been initiated, in order to initiate the GSM service automatically when the mobile is switched on.

In addition, it has to be kept in mind that every time this service starts it must register the receiver of the class ***SMSService****. This is done in the* ***OnCreate*** *method [29].*

```
@Override
public void onCreate() {
    // TODO Auto-generated method stub
    super.onCreate();
     IntentFilter filter = new IntentFilter(SMSService.SMS_RECEIVED);
     this.registerReceiver(mReceiverSMS, filter);
}
```

Action asociated to the Intent

```
SMS_RECEIVED="com.upna.AndoidServer.action.sms_received";
```

A brief schema about the performance of the procedure for the reception and understanding of SMSs is:

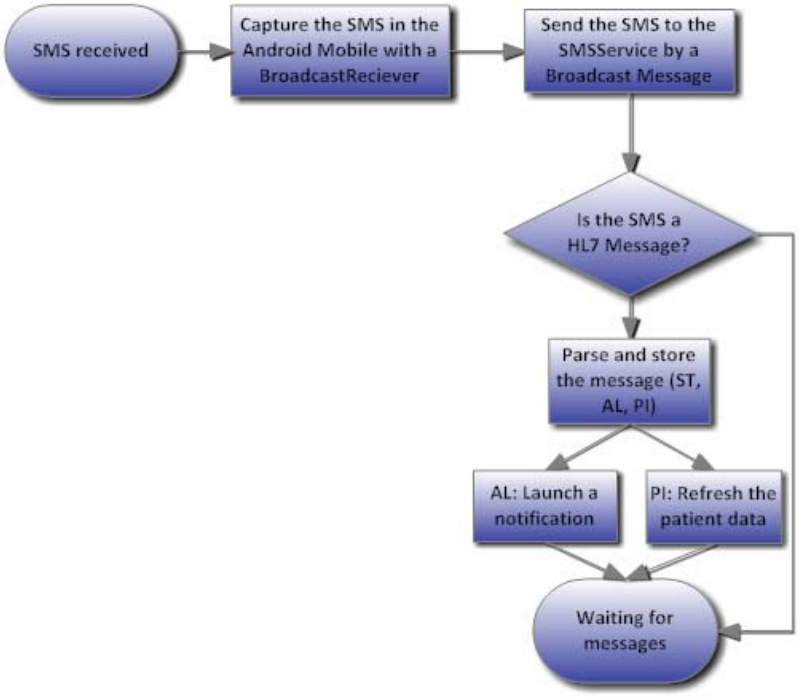


Figure 3.23 *Schema of the SMS reception and interpretation*

3.1.4 Processing Alarm Messages

As seen in the part of the connections, the Alarm message must respond to critical events that happen to the patient.

The procedure related with launching notifications is explained in the page 38 and as noticed, it is used also in the other connections. In this section it will be explained what the user can do after the notification is launched.

As explained before, the notification could be launched in each service of the communications (Bluetooth, TCP/IP or GSM).

Besides, a PendingIntent will be set. It will be executed when the user displays the area of notifications and presses on the notification (see Figure 3.24).

This action will execute an activity called **Showalarm** of the package **com.upna.AndroidServer.Alarm**. It will show these data to the user (see Figure 3.25).

This screen shows the problem from where the alarm originated and the user can react in the following ways: call the emergency services, see more information of the patient, refuse the alert or simply delete the notification.

In the first case, the dialed number is 112 (if the button “YES” is pressed). The application will maintain the notification in the area so if the user of the mobile needs more information about the patient this could be accessed again comfortably.

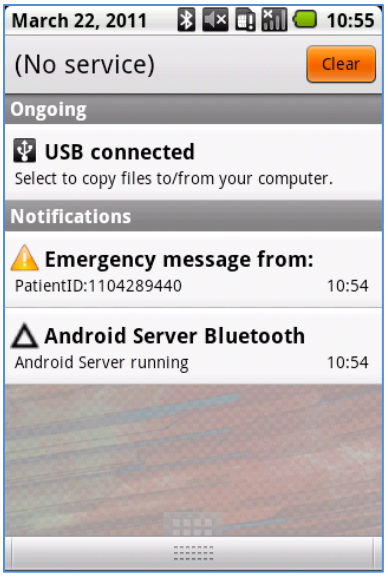


Figure 3.24 Notification area

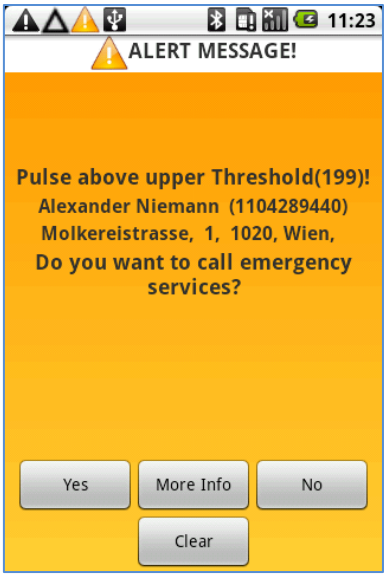


Figure 3.25 Showalarm

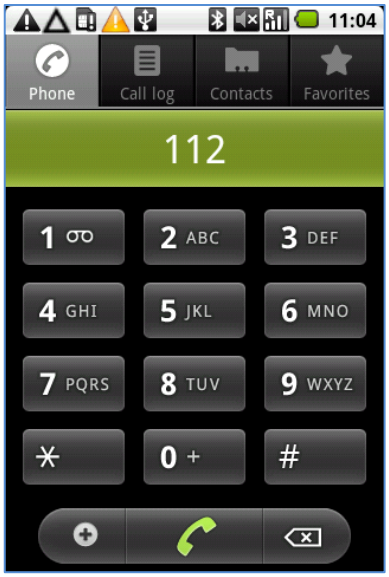


Figure 3.26 Emergency call

The second case (when the button “MORE INFO” is pressed) shows the screen of the patient information (explained in section *3.6.5. TabActivity*). Below are some pictures of the updated patient information.

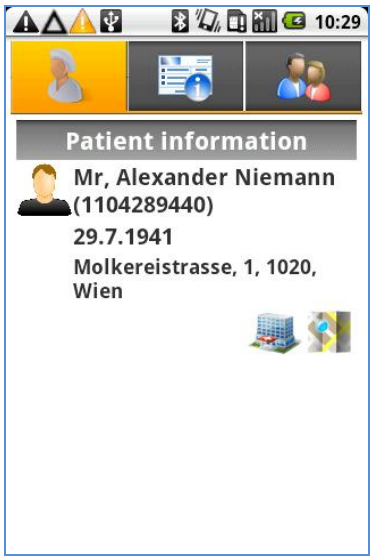


Figure 3.27 Patient Information Tab, after alert message

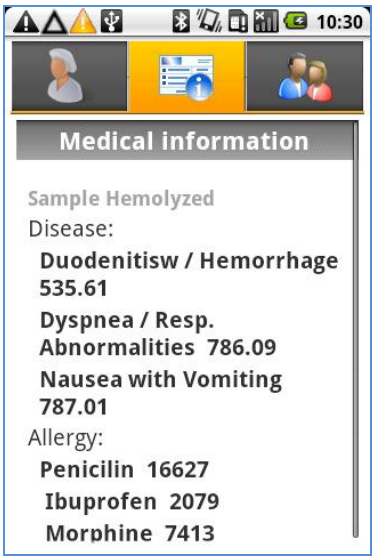


Figure 3.28 Medical Information Tab, after alert message

Figure 3.29 Relatives Information Tab, after alert message

In the **ShowAlarm** activity the needed elements of the HL7 message are shown of the alarm. The number of the notification is stored in the background for identification purposes. In this way, when the user refuses the alarm message by pressing the buttons “NO” and “CLEAR” in the activity, it is possible to make it disappear from the notification area. Furthermore, in case of having more than one alarm notifications, the application will delete the appropriate notification instead of all of them (as the button “CLEAR” of the notifications area would do).

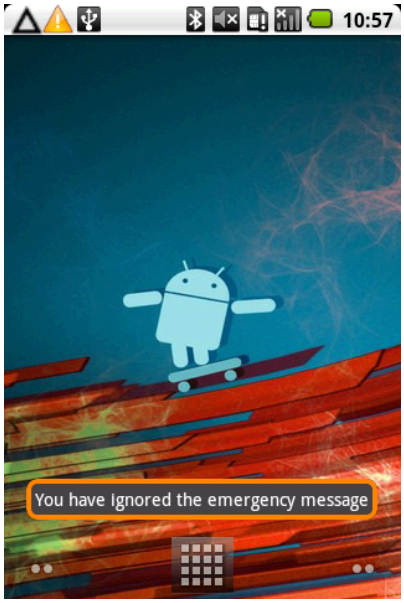


Figure 3.30 After pressed NO button in Showalarm screen

3.2 Connection between the Android Server and HDH Main Server

Figure 3.31 Uploading HL7 messages from the Android Server to the Main Server

One important property of the developed application, besides the capacity to receive the messages from the HDH wrist device, is that the system is capable of sending the stored messages to a Main Server.

Therefore special software based on the Healthy Interoperability (HIO) project of the University of Applied Sciences Technikum Wien offers an HL7 based data exchange with the Android Server. It provides tools to display received biomedical data and stores relevant data into a MySQL patient database; the main server was developed by *P. Khumrin responsible for the "Design and Implementation of Electronic Health Records for the Health Device Hub"* [47].

When the mobile user wants to delete all the messages in the phone to have free space available, these messages will be uploaded to the Main Server. In this way, a copy of all the messages will be in the Main Server. This Service is very useful, because the data storage system in the mobile has a limited space and it is possible to save all the messages in the Main Server without losing data.

The communication is done using Wi-Fi technology. To achieve the correct and complete message exchange, the Main Server should have a public IP and be in "listening" state to accept the connection. The Android will close the connection when the transfer is finished.

A schema about the transfer of the HL7 messages stored in the Android phone to the Main Server is given in the next page:

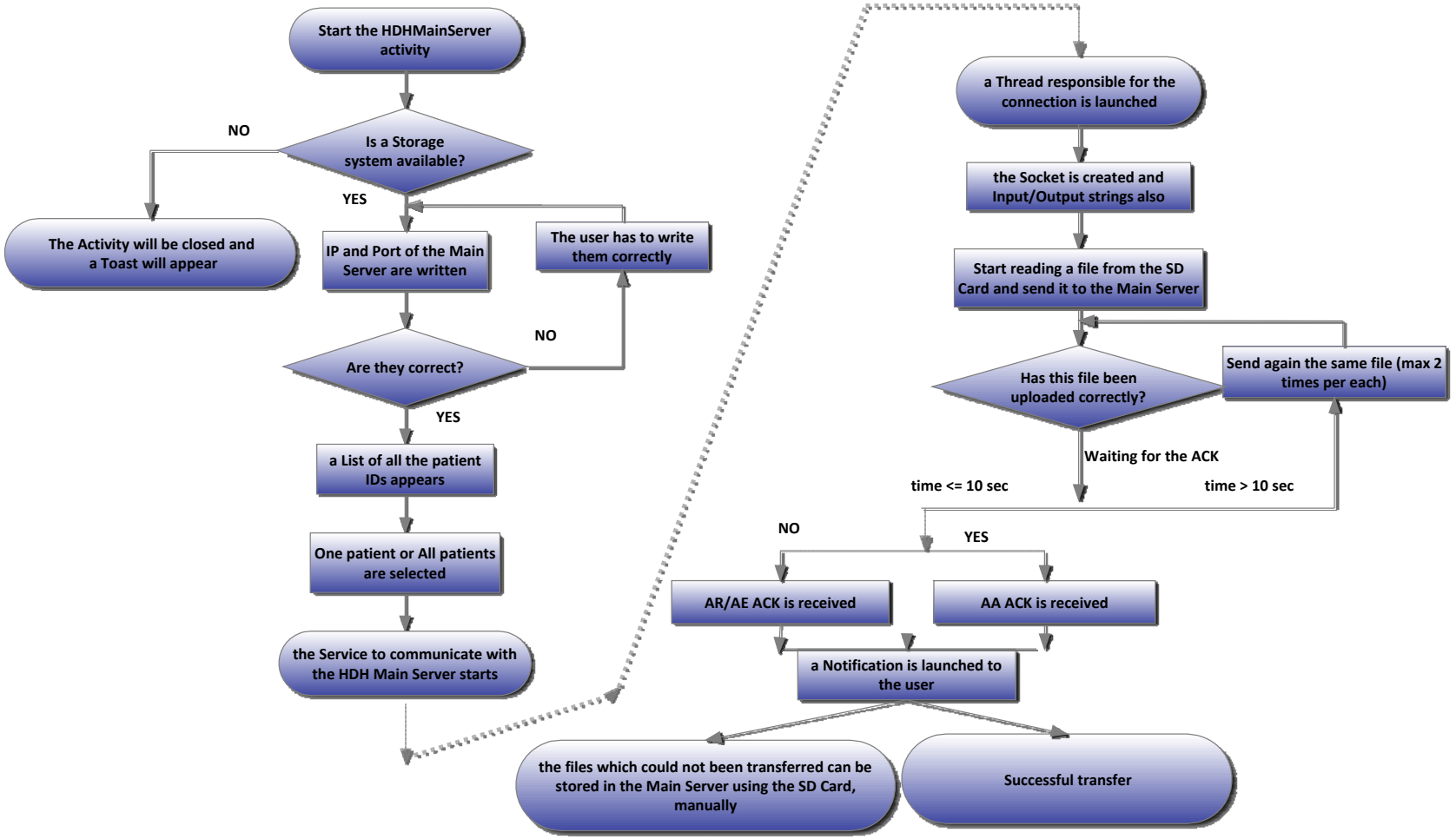


Figure 3.32 Schema of transference from Android phone to the Main Server

After explaining the context where this communication happens, and what it is about, some executed code is shown here. This code is in package **com.upna.AndroidServer.HDHMainServer** and there are four classes in there.

The first activity is **HDHMainServer**, which shows the number of patients whose information are available on the mobile. There is also a form asking for the IP and Port of the Main Server.

To go to this screen the user has to press the button “MAIN SERVER” in the home screen of the application, **AndroidServer_Home**.

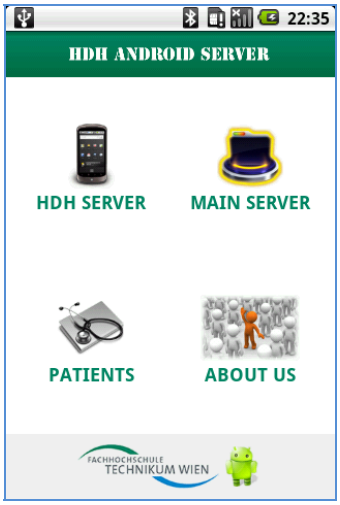


Figure 3.33
AndroidServerHome MAIN SERVER clicked

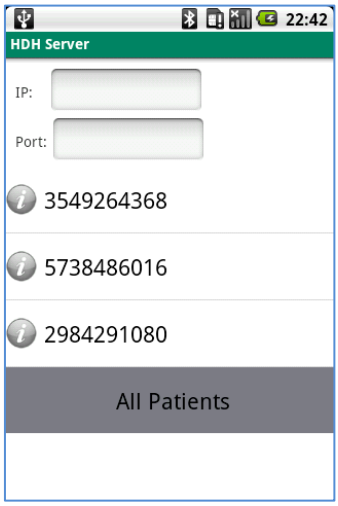


Figure 3.34
HDHMainServer

In the lifecycle of this activity, one of the first actions that should be done is to check if there is an available storage system in the mobile. Resulting from the message sending procedure, the messages are stored in the SDCard of the mobile (everything related is in the package **com.upna.AndroidServer.SDCard**). If there is no a SDcard available, the activity will be closed and the user will be notified by a Toast message that there isn´t available a storage system in the mobile. See next figures.

Figure 3.35 SDCard not available

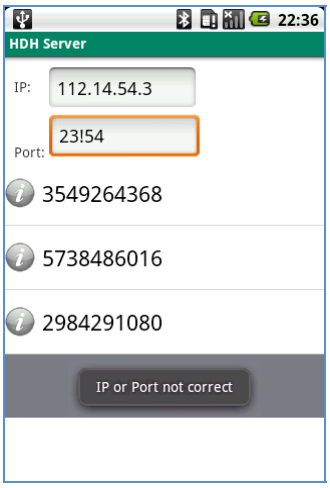


Figure 3.36 IP or Port not correct

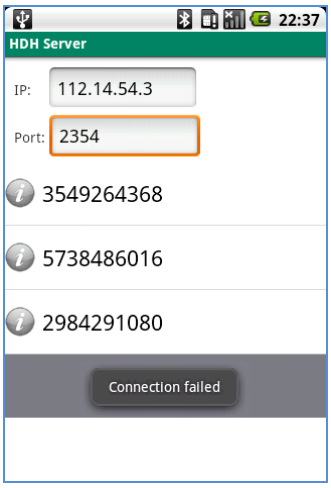


Figure 3.37 Connection failed

The HDHMainServer Activity has to be able to check if the data written in the form are correct (i.e. if the user has introduced the IP or Port properly, through REGEX, regular expression [48]) to avoid trying to connect to an incorrect IP, even if the connection is not successful. See the code that displays the operation of REGEX:

```
public boolean MatchIP(String IP){
    // Check a number with an IP
    final String REGEX = "\\d{1,3}.\\d{1,3}.\\d{1,3}.\\d{1,3}";
    boolean isValid=false;
    Pattern p = Pattern.compile(REGEX);
    Matcher m = p.matcher(IP);
    if((m.matches() == true)){
        String[] Iparray=IP.split("\\.");
        if(Integer.parseInt(Iparray[0])<=255 & Integer.parseInt(Iparray[1])<=255 &
                Integer.parseInt(Iparray[2])<=255 & Integer.parseInt(Iparray[3])<=255){
            isValid=true;
            return isValid;
        }else{
            return isValid;
        }
    }else{
        return isValid;
    }
}
```

```
public boolean MatchPort(String port){
    //Check a number with a port
    final String REGEX = "\\d{4,5}";
    boolean isValid=false;
    Pattern p = Pattern.compile(REGEX);
    Matcher m = p.matcher(port);
    if((m.matches() == true)){
        if ((Integer.parseInt(port)<=65536) & (Integer.parseInt(port)>=1024)){
            isValid=true;
            return isValid;
        }else{
            return isValid;
        }
        }else{
           return isValid;
        }
}
```

First line underlined by blue, shows us that the IP will be correct if it satisfies the IP format (numbers lower than 255.255.255.255). The same happens with the Port (second line in blue). Briefly, the regex for the Port says that it must have 4 or 5 digits (because the used server port should be upper than the Well Known ports: from 0 to 1023 and lower than 65536).

Besides checking the IP and Port, there is an icon alongside each identification number of the patient which after be pressed, will reveal the name and surname of the patient (it is done by the **PopUpWindow** and **CustomAdapter** classes, whose functionality is explained in the chapters *Progress Bar* and *Popup Menu*). Thanks to that, the mobile user can see the patients identity.

In this screen, in addition to be able to upload the information of one patient, the user could upload the information of all the patients. In this way, the SDCard will be totally empty without any patient related information, ready for more future information.

Once an option of the list displayed in this screen is selected, a new service will be opened which will be responsible for managing the connection. The name of this service is **HDHMainServerService**, where the selected data, IP and Port (of the Main Server with which the mobile wants to connect) must be sent as extras. In this case, the sending of this information is done by Intent class and Bundle, both from Android.

```
@Override
protected void onListItemClick(ListView l, View v, int position, long id) {
    // TODO Auto-generated method stub
    super.onListItemClick(l, v, position, id);
    String portS= port.getText().toString();
    String ipS= ip.getText().toString();
    // Check if the IP or port are correct
    if(MatchIP(ipS) && MatchPort(portS)){
         Intent intent=new Intent(HDHMainServer.this, HDHMainServerService.class);
         Bundle b=new Bundle();
         b.putString("Patient", l.getItemAtPosition(position).toString());
         b.putString("IP", ipS);
         b.putString("PORT", portS);
         intent.putExtras(b);
         startService(intent);
    }else{
        Toast.makeText(this, getString(R.string.ip_port_wrong), Toast.LENGTH_SHORT).show();
    }
}
```

In the **onListItemClick** method of the **ListActivity** class (**HDHMainServer**), the method realizes actions explained before (check the IP/Port, Toast for advising, send the information as extras to the service, and start by **startService** of the **Service** class).

As a remarkable aspect, the way to send extras between an activity and a service is the same way than between two activities, except that the extras are received in different methods of each lifecycle. In the case of the activity it is received in the method **onCreate** but in the service it is done in the method **onStart**.

Once the **HDHMainServerService** is initiated, the extras are collected by the method mentioned before and an object of the **HDHMainServerManager** is created, which will initiate the connection with the server. The IP and Port to connect, and the patients to upload will be sent to the object of this **HDHMainServerManager** created and to the method **connect** in this class too.

```
public synchronized void connect(String ipaddress, String port, String pUpload) {
    // Start the thread to connect to the server
    if (mConnectThread == null) {
        mConnectThread = new ConnectThreadtoMainServer(ipaddress,Integer.valueOf(port),pUpload);
        mConnectThread.start();
    }
}
```

In this **HDHMainServerManager** class, there is another implemented class, **ConnectThreadtoMainServer**, which extends Thread, and is responsible for the connection, transferring all the messages by TCP/IP to the Main Server and closing the connection after finishing.

The code related with this class is below. It starts explaining the constructor:

```
public ConnectThreadtoMainServer(String ip, int port, String pUpload){
    this.ip_address=ip;
    this.port=port;
    this.pUpload=pUpload;
    this.clientSocket=new Socket();
    if (pUpload.equals("All Patients")){
        this.idsPatientUpload=SDCard.listDirPatient();
    }else{
        String[] id= {this.pUpload};
        this.idsPatientUpload=id;
    }
}
```

As shown, we obtain the IP and Port as well as the patient or patients that must be uploaded. If all the patients have to be uploaded, the method **listDirPatient** (in the class **SDCard**) is called, which will list the patients available in the SDCard. If there is only one patient selected, it takes only the id of this patient.

In the next lines there is some code which will be executed in the thread of the class.

```
public void run() {
    // TODO Auto-generated method stub
    super.run();
    setName("ConnectThreadtoMainServer");
    try {
        // Connecting with the server
        clientSocket.connect(new InetSocketAddress(ip_address, port));
    }catch (IOException e) {
        mHandlerServiceServer.obtainMessage(1, -1, -1, 0).sendToTarget();
        return;
    }
    if (clientSocket!=null){

        InputStream tmpIn = null;
        OutputStream tmpOut = null;

        try {
            tmpIn = clientSocket.getInputStream();
            tmpOut = clientSocket.getOutputStream();
        } catch (IOException e) {
            mHandlerServiceServer.obtainMessage(1, -1, -1, 0).sendToTarget();
            return;
        }
        mmInStream = tmpIn;
        mmOutStream = tmpOut;
```

In this part of the thread the Socket of the Client is created and the connection with the Server too. If the connection is successful, the objects **InputStream** and **OutputStream** will be created in order to write and read from the buffer of the Socket. If the connection isn´t successful, the code will leave the thread and a message will be sent to the Service to stop it.

Following, the development of the communication is shown:

```
while(i<idsPatientUpload.length){
    filesUpload=dolistFile(idsPatientUpload[i]);
    for(File[] files:filesUpload){
        for (File file:files){
            boolean isSent=false;
            int j=0;
            Log.d(TAG, "Taking a new message");
            //I try to send two times the message, if something is wrong...
            while (j<2 && !isSent){
                String readed=SDCard.readFile(file.getAbsolutePath());
```

As shown in the code above, there are two “while” loops and two ”for” loops. The first one is the responsible of taking a new patient when all the information of the previous patient is already sent. After the while, in the next line are listed all the files of the current patient (idsPatientUpload[i]) in a list with elements of type File[]. In the second “for”, every File[] is taken and in the third “for” each component of the File[].

Once we have the file, the second while starts working. It will check how many times it has already tried to send the same file (two times max) and if it was correctly done. Then, the file is read from the SDCard by using the method **readFile** of the **SDCard** class. It is written in the output buffer and sent to the Main Server.

Meanwhile, the code is waiting for the ACK from the Main Server which will confirm that everything was ok in the sent message.

```
int x=0;
boolean breceived=false;
while(!breceived && x<10){
    try {
        if(mmInStream.available()==0){
            breceived=false;
            Thread.sleep(1000);
            Log.d(TAG,"Nothing in the buffer... waiting...");
        }else{
            breceived=true;
            Log.d(TAG,"We have something in the buffer...");
        }
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    x++;
}
```

By using the method **available** in the InputStream, the code will be waiting until there is something available to read in the input buffer. If there is not, the thread will sleep, max 10 times (one second each sleep time), so the system will wait 10 seconds to receive an

ACK. If in 10 seconds the ACK isn´t received, the execution of the thread will continue (remembering how many times the same message was sent).

Programming this part of the code in this way has made the code very efficient regarding the pending ACK from the Main Server. In fact, it is analyzed almost exactly at the time of arrival of the ACK and the maximum delay is less than one second.
If something is received, the algorithm proceeds to read the message and to parse it, to figure out if the message number is the expected and if it was accepted by the server. The possible options are:

- ACK with “AA” and number of the message is the same as expected. This message will be deleted from the SDCard. Everything was correct and the Main Server has received the message properly.
- ACK with “AR” or “AE” and was sent twice. This message will be stored in the list of the messages that couldn´t parse correctly in the Server. The messages of this type can be copied to the Server manually from the SDCard.
- ACK with “AA” but the message number isn´t suitable and have been sent twice. The system will act as explained above in the second point

Once the mentioned process is finished, the system will continue with the next file, either the file of the same patient or the first file of the next patient to transmit. If there aren´t more files, the connection and service will be closed by sending a message to the Handler of the current service and processing it. See next lines.

```
try {
    clientSocket.close();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
mHandlerServiceServer.obtainMessage(0, -1, -1, failedMessages).sendToTarget();
```

```
public Handler mHandlerServiceServer=new Handler(){
    @Override
    public void handleMessage(Message msg) {
        // TODO Auto-generated method stub
        super.handleMessage(msg);
        switch (msg.what) {
        case 0:
            ArrayList<String> failedMessages=(ArrayList<String>) msg.obj;
            launchNotification(failedMessages);
            stop();
            break;
        case 1:
            Toast.makeText(getApplicationContext(),
                    getString(R.string.failed_in_connection),
                    Toast.LENGTH_SHORT).show();
            stop();
            break;
        }

    }
};
```

```
public void stop(){
    // Auto to destroy the service when the upload finishs
    Intent intent=new Intent(this, HDHMainServerService.class);
    stopService(intent);
}
```

As illustrated in the Handler object, if the message is the case 0, a notification will be launched. Its **PendingIntent** will contain all the messages that couldn´t be uploaded correctly. If it is empty it means that all the uploading were successful. This is done in the **ShowTransferInfo** class.

The following figures are shown when the data transfer to the HDH Main Server concludes.

If the Figure 3.38 is shown, that means that all the files of the patient have been uploaded correctly. On the other hand, if a problem happens while uploading the files, the application informs about which file was uploaded unsuccessfully (Figure 3.39). For example, in the Figure 3.39 is shown that the file 20110411141931-101.txt of the patient 1104289440 wasn´t uploaded correctly.

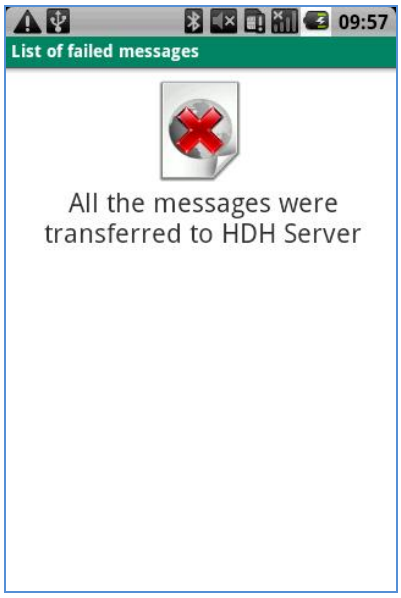


Figure 3.38 Successfully uploaded

Figure 3.39 List of failed messages

When all the files are uploaded a sound notification is launched. See Figure 3.40.

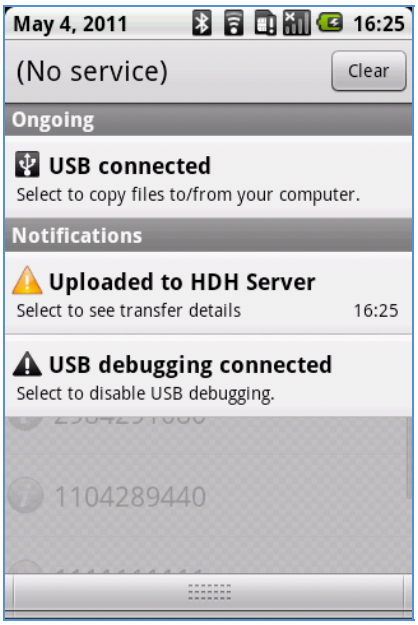


Figure 3.40 Notification area, files uploaded.

3.3 Parser: from HL7 message to HL7

3.3.1 HL7 HAPI for Parsing and Encoding

While working with HL7 messages [2], some tool to process and understand was needed. In this way the work would be faster and of course, it would be robust in case of some errors in the communication between the HDH and Android mobile device.

After searching the Internet, a really useful and strong tool called HAPI was found [13].

HAPI is a HL7 application programming interface. It is an open-source, object-oriented HL7 2.x parser for Java. This project is not affiliated with the HL7 organization; they only wrote some software that conforms to HL7 specification. The project was initiated by the University Health Network.

Basically HAPI translates between HL7-encoded Strings and Message objects. The String format is required for sending messages over the network or storing them in a file. The object format is convenient for reading and setting individual data fields. Parsing is called to convert message string to Message object and Encoding is called to converting Message object to message string.

Both of these actions are performed with a ca.uhn.hl7v2.parser.Parser object.

Although this tool was very attractive we can´t forget that HAPI is written in Java, so you must have a Java Virtual Machine (JVM) installed on your system. If you are doing development, you should get the Java 2 Standard Edition Software Development Kit (SDK), which includes a JVM.

While working with Android and Dalvik (virtual machine VM in Google's Android Operating System), there are problems to use HAPI´s libraries in Android projects. See *Annex VI. Java vs. Android*

From the webpage of HAPI [49] it is possible to download the libraries for this tool to work with Java (xercesImpl-2.4.0.jar, xmlParserAPIs-2.6.2.jar, xalan-2.7.0.jar, jaxen-1.0-FCS-2.7.0.jar, saxpath-1.0-FCS.jar, JDOM-1.1.jar). However, neither the library xerces nor saxpath works with Android.

For this reason, a Parser was built for the De/Encoding messages.

3.3.2 Structure of the HL7 message

Before explaining in detail the composition of this parser, the structure of the HL7 messages is explained more specifically.

HL7 version 2.x

The HL7 version 2 standard [2] has the aim to support hospital workflows. HL7 version 2 defines a series of electronic messages to support administrative, logistical, financial as well as clinical processes. Since 1987 the standard has been updated regularly, resulting in versions 2.1, 2.2, 2.3, 2.3.1, 2.4, 2.5, 2.5.1, 2.6 and 3. The v2.x standards are backward compatible (i.e. a message based on version 2.3 will be understood by an application that supports version 2.6). The v3 standard, as opposed to version 2, is based on a formal methodology (the HL7 Version 3 Development Framework, HDF) and object-oriented principles.

HL7 v2.x mostly uses a textual, non-XML encoding syntax based on delimiters.

HL7 v2.x has allowed for the interoperability between electronic Patient Administration Systems (PAS), Electronic Practice Management (EPM) systems, Laboratory Information Systems (LIS), Dietary, Pharmacy and Billing systems as well as Electronic Medical Record (EMR) or Electronic Health Record (EHR) systems. Currently, HL7's v2.x messaging standard is supported by every major medical information systems vendor in the United States [2].

HL7 v 2.6 Message construction

This is a brief description of the encoding rules of messages. The messages are formed by data fields of variable length separated by separator character fields. The rules describe how you can encode the different types of data in a field and when you can repeat them. The data fields are logical groupings called segments, which are separated from each other by separator characters. They can be | ^ ~ \ & <CR>. Each of them deals with one type of date [2].

For example:

- Ending of the segment <CR> (ASCII 13)
- Separator of the Field | (ASCII 124)
- Separator of the Component ^ (ASCII 94)
- Separator of the Subcomponent & (ASCII 38)
- Character of Repetition ~ (ASCII 126)
- Character of Escape \ (ASCII 92)

Each segment begins with a value of three characters length that identifies the segment in the message. The segments can be repeated and the data fields are located in a message by its relative position associated with the segments.
In the next lines, the used segments of the HL7 messages are shown, used for the communication with the HDH and Main Server.

To work in monitoring the patient, there are three types of messages, called PI, ST or AL, explained below. The HDH wrist device deals with these types of messages:

Message Type PI: Patient Information

This message type contains the main information about the patient, his relatives, diseases and allergies suffered by the patient and some medical description among other things.

The main target is to have the main important data of the patient always ready in the Android mobile device, because when the HDH is connecting to the mobile or reconnecting to an already known one, all the information will be saved in the mobile data base. In this way, the user can access to all the patient information any time, and most importantly, in emergency or alert situation.

This is an example of the PI message:

```
MSH|^~\&|10:00:E8:C1:1C:3C||00:23:D4:69:C1:F1||20100510120000||ADT^A04^ADT_A04|PI00
01|P|2.6|
EVN||20100510120000|
PID|||1104289440|| Niemann^Alexander||19410729|M|||&Molkereistrasse&1^^Wien^^1020|
NK1|1|Niemann^Gabriel^^^Dr|^Son|&Blumenweg&4^^Eitzing^^4970|^^^^0043^1234^56789|
NK1|2|Niemann^Anja^^^Ing|^Daughter|&Mariahilfer&5^^Wien^^1060|^^^^0043^9876^54321|
PV1||telepatient|
AL1|1||16627^PENICILLIN|
AL1|2||2079^ IBUPROFEN |
AL1|3||7413^ MORPHINE|
DG1|1||535.61^DUODENITISW/HEMORRHAGE|||F|
DG1|2||786.09^DYSPNEA/RESP.ABNORMALITIES|||F|
DG1|3||787.01^NAUSEA WITH VOMITING|||F|
NTE|||Sample Hemolyzed|
```

Figure 3.41 PI message

Except the MSH and PID segments, which are in all the HL7 message types, the other segments; EVN, NK1, PV1, AL1, DG1 and NTE, are segments of the PI message itself.

MSH and PID are the segments which deal with the received message´s parameters and the actually patient parameters. Without going into specific fields, the MSH owns the

information about the addresses of the HDH wrist device and mobile phone, and some important information of the messages as the receiving date-time, its type and some important ID´s, among other things. On the other hand, the PID talks about the patient basic information, his name, identification and his address.

EVN deals basically with the recorded date-time.

The information of the relatives of the patients is shown in the NK1 segments. In these HL7 messages, we only have information of the two relatives, whose name, relationship, address, phone numbers etc. are given.

PV1 classify the patient type, for example, in this case the patient is described as a telepatient.

The information of the allergies or diseases suffered by the patient is written in the AL1 and DG1 segments, which are defined by their number and description. Currently only three allergies and diseases are available within this HL7 structure.

The last segment, NTE contains all the important facts of medical diagnosis, therapies etc.

For more detailed and specific information about the meaning and contained information in each field, it is recommended to read the *Annex IV. Table of the HL7 messages components* or more in HL7 standard [2].

Message Type ST: Standard Message

This type of messages is STandard Message. This message contains all the data related with the monitoring of the patient.

Occasionally, the HDH sends the data of the measured pulse and activity of the user. The Android phone must be able to receive, store and process these messages.

ST messages have the MSH segment and PID segment, as every type of HL7 messages does. The meaning of their information is already explained above.

OBR segment is responsible to preserve the information of the starting and ending date-time of the measurements done to the patient. It also contains a description of a danger code if it is necessary.

As you can see, in our ST or AL messages, there are five OBX segments. Each of them has its own information.

For example, the first OBX has information of the channel related with the wave of the measurement. There is the information of the name of channel, amplitude and frequency. In the second OBX2 you can see information of the time of the measurement.

The information of the pulse dates are saved in the third OBX.

Due to the fact that the wrist device is not able to continuously measure the pulse on the wrist, a 4th OBX-segment is implemented. This segment contains for every pulse value a timestamp. The first two digits represent the minute and the second two digits represent the second (mmss). The hour and the date have to be parsed from the message timestamps in the MSH or second OBX-segment.

Moreover, there is implemented in the HDH wrist device an algorithm in order to know the activity of the patient in the same second that the pulse data is measured.

The activity (and the related altitude) is shown in the 5th OBX-segment. For this reason, for every second there is a pulse data and an altitude/activity data. In this way, the pulse data but also the altitude/activity range of the patient is known. This is necessary because it has to be differentiated, for instance, to have a high pulse data when the patient is calm or if he or she is moving (for example going upstairs). In addition to the pulse value it is necessary to know in which context the patient is at that moment, so the pulse can be analyzed within a specific situation (context).

For more detailed and specific information about the meaning and contained information in each field, it is recommended to read the *Annex IV. Table of the HL7 messages components* or more in HL7 standard [2].

```
MSH|^~\&|10:00:E8:C1:1C:3C||00:23:D4:69:C1:F1||20100510121500||ORU^W01^ORU_W01|AL0001|
P|2.6|
PID|||1104289440|| Niemann^Alexander||19410729|M|||&Molkereistrasse&1^^Wien^^1020|
OBR|1||0000^BLA|5^three-channelwaveformrecording^99SVL|||20100510120000|20100510121500
||||||||||||||||F|
OBX|1|CD|5&CHN^^99SVL|1|1^ONE^0.5&mv^^100^0&255||||||F|
OBX|2|DTM|5&TIM^^99SVL|1|20100510121500||||||F|
OBX|3|NA|5&WAV^^99SVL|1|110^112^115^112^110^110^109^110^108^112^112^111^112^113^115
||||||F|
OBX|4|NA|5&WAV^^99SVL|1|0007^0007^0007^0007^0007^0007^0007^0007^0007^0007^0007^0007
^0007^0007^0007||||||F|
OBX|5|NA|5&WAV^^99SVL|1|101^91^81^71^61^51^51^51^51^51^52^52^52^52^52||||||F|
```

Figure 3.42 ST message

Message Type AL: Alert Message

The third message type is an Alert Message (AL). This message is sent immediately after it´s created because it contains critical information. This message contains the personal information of the patient, a description of the alarm situation (for example pulse above a certain threshold), attributes of the recording channel (sample frequency, physical resolution, digital resolution, channel number) followed by the recorded values.

Receiving this message on the phone will cause a loud alarm and a vibrator call. Thus, a mobile user feedback is necessary to turn it off. In this way, the reaction of the mobile user for the alarm is immediately. More detail is available in the section *3.1.4 Processing Alarm Messages.*

Not conceptually but grammatically, the AL message is the same as ST. More information about the meaning and contained information in each field, is available in *Annex IV. Table of the HL7 messages components* or more in HL7 standard [2].

```
MSH|^~\&|10:00:E8:C1:1C:3C||00:23:D4:69:C1:F1||20100510121500||ORU^W01^ORU_W01|AL0001|P|2.6|
PID|||1104289440|| Niemann^Alexander||19410729|M|||&Molkereistrasse&1^^Wien^^1020|
OBR|1||0000^BLA|5^three-channel waveform recording^99SVL|||20100510120000|20100510121500||||Pulse
above upper Threshold (199)!|||||||||||||F|
OBX|1|CD|5&CHN^^99SVL|1|1^ONE^0.5&mv^^100^0&255||||||F|
OBX|2|DTM|5&TIM^^99SVL|1|20100510121500||||||F|
OBX|3|NA|5&WAV^^99SVL|1|110^130^130^170^199^180^180||||||F|
OBX|4|NA|5&WAV^^99SVL|1|0007^0007^0007^0007^0007^0007^0007||||||F|
OBX|5|NA|5&WAV^^99SVL|1|22^22^22^25^25^25^25||||||F|
```

Figure 3.43 AL message

Message Type ACK: Acknowledgement Message

However, there is a fourth type of message which is essential for the communication both with the HDH device and with the Main Server. Once HL7 messages are transmitted, (the message that the wrist device sends to the Android Server or the messages that the mobile sends to the Main Server), for each HL7 message sent/received it is necessary to send an ACK, to reply and confirm to the sender that the message receipt was correct or incorrect.

```
MSH|^~\&|00:23:D4:69:C1:F1||10:00:E8:C1:1C:3C||20100510121600||ACK^W01^ACK|1234|P|2.6|
MSA|AA|PI0001|
```

Figure 3.44 ACK message

Basically, the ACK message informs to the HL7 message transmitter that everything was well in the reception, or if there was some problem with the message.

If an ACK with "AA" is received, that means that the HL7 message was correct. Otherwise, an "AR" indicates that the message was incorrect, and thus the transmitter knows that it has to retransmit the message.

The ACK message has two segments, MSH and MSA. In both, there are fields that must be filled with specific fields from the received message.

For example, the ACK MSH-3 has to be filled by MSH-5 from the received message, the ACK MSH-5 with the MSH-3. In the same way, the ACK MSH-4 with MSH-6, and ACK MSH-6 with the MSH-4 of the received message.

Field MSH-9 ACK is also very important for the created Parser, because it is the one which reports the type of message.

Additionally, in the MSA segment, the field 2 (Message Control ID) corresponds to the MSH-10 (Message Control ID) of the received message. Thus, each ACK identifies only one HL7 message. In the example above, ACK belongs to the PI message whose control ID is 0001.

The tables written in *annex IV. Table of the HL7 messages components* show the segments used in the HL7 messages that are used currently in the system. In bold specifies each of the fields used in each of the segments.

For more detailed and specific information about the HL7 standard, please see the standard definition [2].

3.3.3 HL7 Parser

```
MSH|^~\&|10:00:E8:C1:1C:3C||00:23:D4:69:C1:F1||20100510121500||ORU^W01^ORU_W01|AL0001|
P|2.6|
PID|||123456789|| Niemann^Alexander||19410729|M|||&Molkereistrasse&1^^Wien^^1020|
OBR|1||0000^BLA|5^three-channelwaveformrecording^99SVL|||20100510120000|20100510121500
||||||||||||||||F|
OBX|1|CD|5&CHN^^99SVL|1|1^ONE^0.5&mv^^100^0&255|||||F|
OBX|2|DTM|5&TIM^^99SVL|1|20100510121500|||||F|
OBX|3|NA|5&WAV^^99SVL|1|110^112^115^112^110^110^109^110^108^112^112^111^112^113^115
||||||F|
OBX|4|NA|5&WAV^^99SVL|1|0007^0007^0007^0007^0007^0007^0007^0007^0007^0007^0007^0007
^0007^0007^0007||||||F|
OBX|5|NA|5&WAV^^99SVL|1|101^91^81^71^61^51^51^51^51^51^52^52^52^52^52||||||F|
```

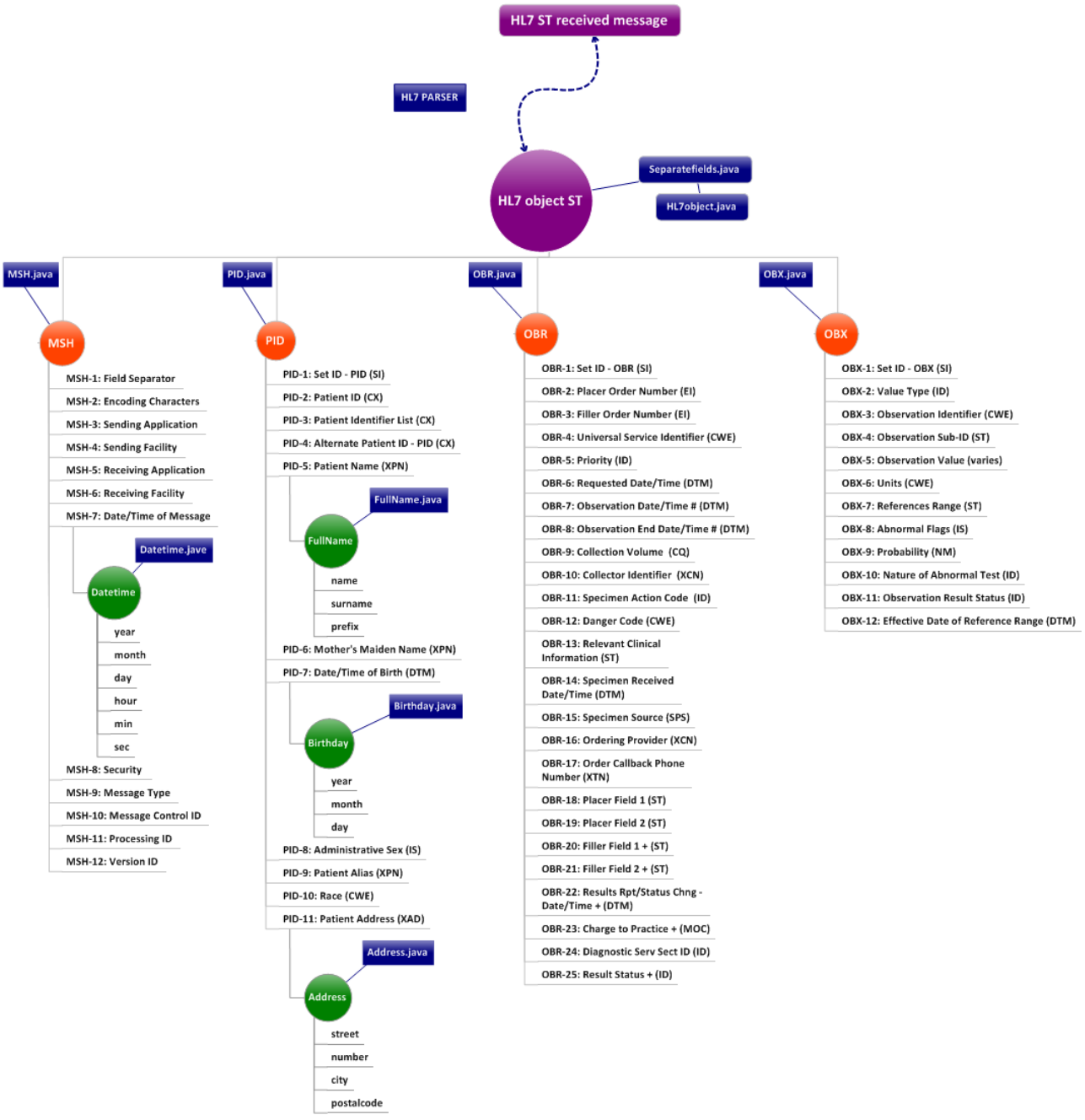


Figure 3.45 *Concept map of the current HL7 message [29]*

All classes created related to the parser, are located in the package ***com.upna.AndroidServerHL7*** *[29].*

Observe from Annex II. AndroidServer Project Directory

In the next lines the operation of the parser is explained:

Firstly, after receiving the HL7 message, the parser analyzes the type of this message. This is done in class ***Separatefields.***

The types of messages sent by the HDH wrist devices are ST, AL and PI (as explained in the chapter HL7 v 2.6 Message construction). It has to be kept in mind that the application developed for Android, won´t only work as a HDH Server, but it also has to communicate with a Main Server when the SDCard of the mobile is full, in order to transfer all the stored data. Thus, the Android mobile also receives an ACK message for each HL7 message Android uploaded to the Main Server.

The method responsible for analyzing the type of message received is ***messageType*** *(if the messages are from the HDH), or the method* ***ACKmessageType*** *(if the message is from the Main Server). In the received message, the character position in the segment, which indicates the kind of message, is not the same in both cases. As seen, for ST, AL and PI type message, it is described in the first two characters of MSH-10 (Message Control ID, the first two characters), while for the ACK the type is in MSH-9 (Message Type, the first three characters).*

Then, the message is separate in segments, using the separator of segments (in our case "\r\n", "\r" or "\n").

```
public static Separatefields Separatefields_Parse(String InputString_){

    String separator= "\r\n";
    String[] fieldsOfInputstring=InputString_.split(separator);
    if (fieldsOfInputstring.length==1){
        separator="\r";
        fieldsOfInputstring=InputString_.split(separator);
        if(fieldsOfInputstring.length==1){
            separator="\n";
            fieldsOfInputstring=InputString_.split(separator);
        }
    }else{
        Log.d("SeparatedFields", "Split worked");
    }
```

Each type of HL7 message has a certain number of segments, which are known by three characters in their first field. They can be for example, MSH, PID, OBR, OBX ... or any other of the segments explained deeply in the section of HL7 messages HL7 v2.6 Message construction.

```
if(messageType.equals("ST")| messageType.equals("AL")){
    array= new String[8];
    String setId="";
    for (int i = 0; i <= (lengthOfArray-1) ; i++){
        String type=fieldsOfInputstring[i].substring(0,3);
        if (type.equals("MSH")){array[0]=fieldsOfInputstring[i];}
        if(type.equals("PID")){array[1]=fieldsOfInputstring[i];}
        if(type.equals("OBR")){array[2]=fieldsOfInputstring[i];}
        if(type.equals("OBX")){
            setId=fieldsOfInputstring[i].substring(4,5);
            if (setId.equals("1")){array[3]=fieldsOfInputstring[i];}
            if(setId.equals("2")){array[4]=fieldsOfInputstring[i];}
            if(setId.equals("3")){array[5]=fieldsOfInputstring[i];}
            if(setId.equals("4")){array[6]=fieldsOfInputstring[i];}
            if(setId.equals("5")){array[7]=fieldsOfInputstring[i];}
        }
    }
}else if (messageType.equals("PI")){
```

(…)

After separation of the received message into segments, they are assigned as attributes of the created ***SeparateFields*** *object.*

```
public Separatefields(String[] array, String messageType){
    if (messageType.equals("ST")| messageType.equals("AL")){
        this.MSH=array[0];
        this.PID=array[1];
        this.OBR=array[2];
        this.OBX1=array[3];
        this.OBX2=array[4];
        this.OBX3=array[5];
        this.OBX4=array[6];
        this.OBX5=array[7];
    }
    else{
        if (messageType.equals("PI")){
```

(…)

The next important class is ***HL7object.***

In this class, each segment of the received message is parsed (they are the result of SeparateFields, explained above). Once the parsing of each segment is correct, then an object for that segment is created. After analyzing and parsing each of the segments and obtaining an object for each of them, they are assigned as attributes of a big object called HL7object, which covers all objects of the segments.

Following it is explained how this has been done in the code, step by step, based on the case of MSH. The same explanation can be extended to the other segments.

It is shown that there is a specific parser which analyzes the MSH. Subsequently, the ***MSHparsed*** *object of class* ***MSH*** *is created.*

```
//MSH
String MSHmessage=message.getMSH();
    try {
        MSHparsed=MSH.MSH_Parse(MSHmessage);
        this.msh=MSHparsed;
```

This is done in the same way for each of the segments, depending on the type of message, ST, PI, AL or ACK. For each of the types, it is needed to examine their own segments and parse each one of them.

MSH´s object attributes are named after their corresponding HL7 fields, which are separated by ***MSH_Parse*** *method. The separator within a segment is "|".*

```
this._fieldSeparator= _fieldSeparator;
this._encodingCharacters = _encodingCharacters;
this._sendingApplication = _sendingApplication;
this._sendingFacility = _sendingFacility;
this._receivingApplication = _receivingApplication;
this._receivingFacility = _receivingFacility;
this._dateTimeOfMessage = _dateTimeOfMessage;
this._security = _security;
this._messageType = _messageType;
this._messageControlID = _messageControlID;
this._processingID = _processingID;
this. versionID =  versionID;
```

For each segment, there is a class and a specific parser, whereby it is possible to obtain the objects for each of the segments (which are the attributes of the HL7 object).

The classes for each of the segments are: ***ACK***, ***AL1***, ***DG1***, ***EVN***, ***MSA***, ***MSH***, ***NK1***, ***NTE***, ***OBR***, ***OBX***, ***PID***, *and* ***PV1***.

And the parsers:

```
static public AL1 AL1_Parse(String InputStringAL1_)throws Exception{
static public DG1 DG1_Parse(String InputStringDG1_)throws Exception{
static public EVN EVN_Parse(String InputStringEVN_)throws Exception{
```

The other parsers are corresponding to the upper definitions.

Furthermore, with some segment fields internal objects were created, such as the case of MSH-7 (Date/Time of Message) for which an internal object Datetime was created or in the case of PID-11(Patient Address) with the internal object Address and the object Allergy for the field AL1-3(Allergen Code/Mnemonic/Description) etc.

In order to understand how the received messages are parsed into an HL7 object please see the Figure 3.44. It is important to mention that this scheme is an approximation to

what has been done, so, for more specific details it is recommended to observe the code in the package ***com.upna.AndroidServer.HL7.***

In case the parsing of the HL7 message segment is not correct following situation occurs.

Continuing the example above, the MSH segment is analyzed.

If any of the fields cannot be parsed correctly, for example, if the number of fields received does not match what was expected, the parser will throw an exception, showing that the parsing of that segment is not possible.

An exception is thrown in ***MSH*** *as the next lines show:*

```
else {
    System.out.println("FieldsOfMSH[] is not correct, number of fields not correct.");
    throw new Exception("FieldsOfMSH[] is not correct, number of fields not correct");
}
```

This exception is received in ***HL7object:***

```
//MSH
String MSHmessage=message.getMSH();
    try {
        MSHparsed=MSH.MSH_Parse(MSHmessage);
        this.msh=MSHparsed;
    }catch(Exception e){
        System.out.println("MSH can´t be parsed correctly");
    }
```

In this moment, the system discards the received HL7 message because it is invalid, and simply waits for a new message.

This behavior of exceptions in this code is a feature that makes the parser operation very efficient, because even if there is something wrong with the HL7 message, the Thread that manages the exchange of messages is not blocked. This issue is very important. It is better explained in chapter 3.1 Connections between HDH and Android Server in the section of Bluetooth.

- *Once managed to build the HL7 object from the received message, the behavior of the mobile application depends on the source of that message.*

On the one hand, the message can be an ACK message, which implies that the message is from the Main Server. In this case, it is only necessary to figure out if the ACK message is "AA" or "AR".

If it is the "AA", it is known that the message (whose Control ID is given in the MSA-2 field) which was uploaded before to the Main Server was well received, and thus, everything was successful. If the received ACK message is "AR", means that there were some problems with the message in the Main Server or while the transfer. In this case, the program has to resend again the message to the Main Server.

If there is a problem, the mobile will try to upload the same message to the Server twice. Then, if the received ACK continues showing "AR", the mobile will show to the user that something went wrong while uploading and this specific message will be stored in the mobile in order to be transferred by the SDCard.

The mobile does not save the ACK messages from the Main Server, it just processes them but doesn´t conserve.

On the other hand, if the received message is from the HDH wrist device, the message can be ST, PI or AL type. Once managed to build the HL7 object from that message, the last action is to reply to the HDH wrist device with an ACK.

To achieve this, what mainly was done after building the HL7 object is examining if the MSH, PID and OBR have the indispensable fields filled. This is necessary because it is possible to receive an HL7 message without some essential data fields such as the MSH-3 (Sending Application), or PID-2 (patient ID).

```
HL7object hl7=new HL7object(HL7Message);
Log.d(TAG, "HL7 parse susccessful");
ACK ack=ACK.create_ACK(hl7);
```

This code is executed, for example, in HDHClient of the package ***com.upna.AndroidServer.Bluetooth****. These lines are in detail in the chapter Android Mobile as a Server.*

In the ACK class, there are several functions to analyze MSH, PID and OBR.

```
static private boolean analizeARsituation_MSH(MSH MSHparsed){(…)
static private boolean analizeARsituation_PID(PID PIDparsed){(…)
static private boolean analizeARsituation_OBR(OBR OBRparsed){(…)
(…)
```

According to the analysis done in ***ACK*** *class, the MSA-1(Acknowledgment Code) can be "AA" or "AR". The "AA" will be written when the received fields are correct, and "AR" is sent if any of the important fields are null. The rest of the MSH and MSA segment fields of the ACK message will be filled properly depending on the received message; because*

the ACK always belongs only to a single received HL7 message. For example, the MSA-2 (control ID) of the ACK that is built is the same as the MSH-10 of the received message, the MSH-3 of ACK is a copy of MSH-5 of the received message. The same procedure is used for dealing with the other fields.

```
MSH msh_received=MSHparsed;// MSH OF THE RECEIVED MESSAGE
MSH _msh= new MSH();// MSH OF THE ACK MESSAGE THAT WE ARE BUILDING

_msh.set_fieldSeparator("|");
_msh.set_encodingCharacters("^~\\&");

_msh.set_sendingApplication(msh_received.get_receivingApplication()); //It´s a copy from the MSH-5 of the received messa
_msh.set_sendingFacility (msh_received.get_receivingFacility()); //It´s a copy from the MSH-6 of the received message.
_msh.set_receivingApplication(msh_received.get_sendingApplication()); //It´s a copy from the MSH-3 of the received messa
_msh.set_receivingFacility(msh_received.get_sendingFacility()); //It´s a copy from the MSH-4 of the received message.
```

(…)

Only in the case that everything was correct and the ACK which is going to be sent to the HDH is "AA" type, the mobile phone will store the received message [29].

3.4 Patients Management

This section deals with the management of the patients. Different issues are explained, for instance, the life cycle of a patient in the system, i.e., when to get introduced to the program and when to get deleted from the application. It is also mentioned which kind of structure was created to store the patient data in the external memory. Finally, it is described how all data received by the connections are stored and how they will be showed to the mobile user.

All these tasks are developed from different packages. They are listed below:

- **com.upna.AndroidServer.HistoryPatient**
- **com.upna.AndroidServer.Alarm**
- **com.upna.AndroidServer.Chart**
- **com.upna.AndroidServer.SDCard**

3.4.1 Lifecycle of the patient

As mentioned before, the lifecycle of the patient in Android Server is defined as all the states that the patient must go through in the application in order that everything works correctly. These states are:

- Register a patient
- Access to the patient data
- Delete a patient

Register a patient

As explained in the report, on the main screen of Android Server click the button "PATIENTS".

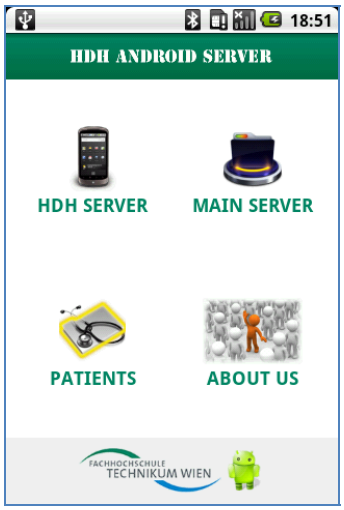


Figure 3.46 AndroidServerHome

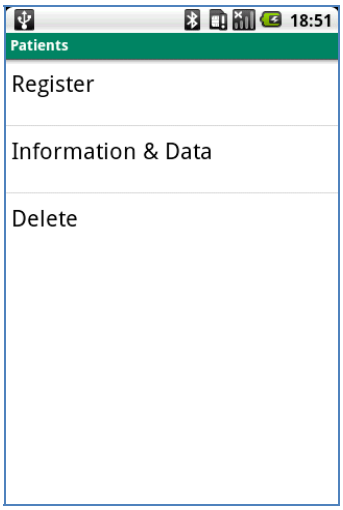


Figure 3.47 History

PATIENTS clicked

Then, the next activity is shown, **History**, which gives access to the already mentioned cases in the life cycle.

The first case is Register. It offers a form to the user which has to be filled in to complete the registration.

Note: it is necessary to comment that ***the register of the patient is mandatory*** to the reception and storage of the HL7 messages. Otherwise the proper reception and storage is not ensured. It was decided to implement this in this way to prevent that the HDH could add new patients without registry. If other functionality had been implemented, it would have generated problems; because if in one of the communications with the HDH wrist device the sending of the patient ID had been incorrect, a new patient file would have been created in the Android Server and there would have been two different files created for the same patient.

On the screen of the register form it is sufficient to include the patient ID number. The fields Name and Surname are optional and if they are not filled, default values will appear on the screen related to the patient information.

Furthermore, a REGEX [48] structure will check if the introduced data is correct. It is also examined if there is already a patient file created with the same patient identification. If so, the user will be informed that the register hasn´t been completed correctly.

Figure 3.48 Form of Register a Patient

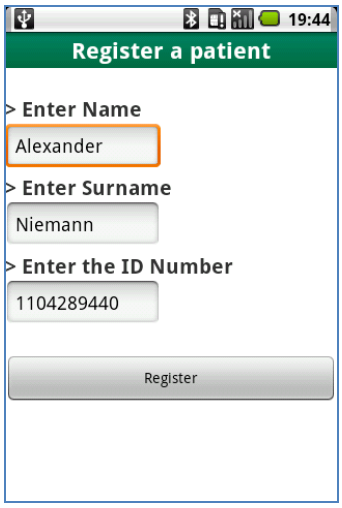


Figure 3.49 Form filled

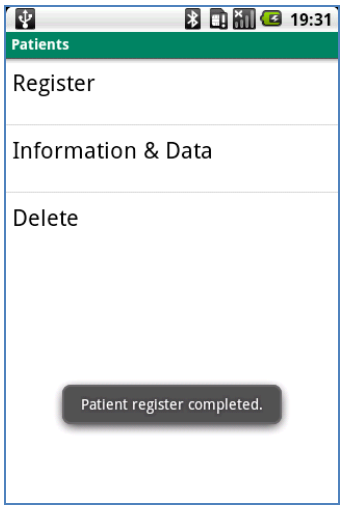


Figure 3.50 Patient was registered

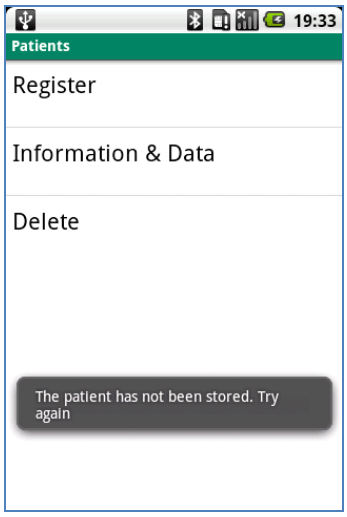


Figure 3.51 Not stored of the patient

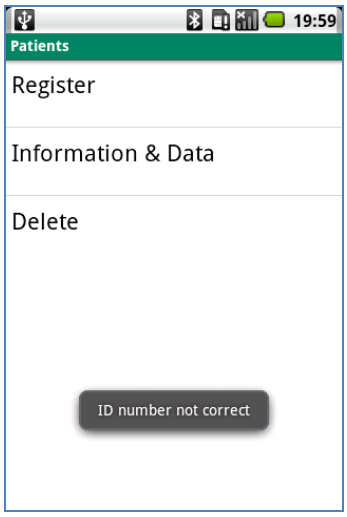


Figure 3.52 ID not correct

After explanation of the register process and possible errors that are shown to the user after this process, the internal functionality will be explained in detail.

Every time a patient is registered in the file system, a new space is given in the external memory for this patient. There, three folders are created: STAL, PI and XML. In the last mentioned folder, a XML file is created related with the structure of all data of the patient (and the data introduced in the register: identification number and name/surname if they were written).

The developed structure is the following:

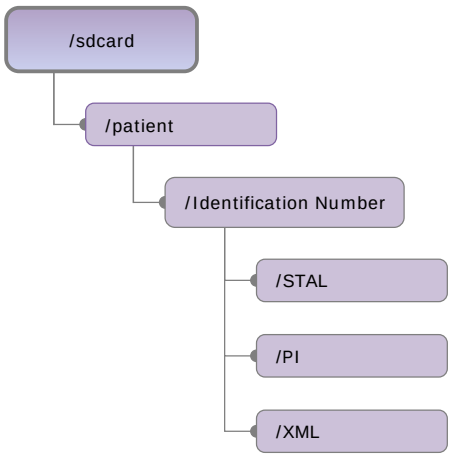


Figure 3.53 Structure of the patient in the storage system

The patient will be stored with his own Identification Number (unique and unrepeatable) and the HL7 messages will be saved there, depending on their own type. If the messages are AL or ST, they are stored together because they have inside the relevant

medical information as the pulse or activity/altitude of the patient. In the PI folder, all the update messages received from the HDH are stored. These messages, as shown in the chapter of the connections by Bluetooth Technology, are responsible of updating the information of the patient that is serialized in the XML file. This is why the XML directory exists.

The XML directory is something like “having a patient information library” in our program, which allows working with patient data any time. It is used for several things: for instance, to launch alarms, show information of the patient or even makes it easier to provide a nicer interface for the user because it allows identifying the number of the patient with his name and surname by using the pop up system (explained later in the chapter *PopupWindow*).

The procedure of serializing the patient data is done using an open source serialize type library, called Simple XML [40].

It was decided to use this library (instead of others which are implemented by Android which work in Java) because of these reasons:

- *Simple framework with powerful capabilities. The framework used to provide XML serialization is simple to use and revolves around several annotations and a single persistent object used to read and write objects to and from XML.*
- *Can handle cycles in the object graph. The persistence engine can handle cycles in the object graph, which enables complex objects with recursive references to be* serialized. This ensures that the deserialization process can recover all object references.
- *It requires absolutely no configuration. Unlike many of the XML frameworks for Java there are no mappings or configuration required to serialize objects regardless of its complexity. The XML schema is represented using field annotations.*
- *Extremely rapid development with XML. Developing XML configuration and communication systems can be done much quicker than through the use of XML frameworks such as DOM, SAX, and even other frameworks such as Digester and XStream.*
- *Converts to and from human editable XML. A primary goal for the framework is that the XML data used to deserialize a serialize objects is human readable. All XML elements and attributes take a simple structure that can be easily created with a text editor.*
- *Contains an XML templating system. As part of the deserialization process template markers within the XML elements and attributes can be replaced with variables. This allows the system to be easily adapted for use as a configuration system with dynamic value substitution* [40].

The object, that will be serialized, is created in the **Patient** class. The structure of this object is shown in the *Annex .V Patient Structure.*

Until now the discussion was about the register action. The next step is to explain how it is possible to delete a patient from the system.

Delete a patient

Deleting a patient allows to free up all the space assigned to the patient when he or she was registered. This action will destroy the stored HL7 messages of the patient if they have not been uploaded to the Main Server previously.

The delete process is generated from the **DeletePatient** activity.

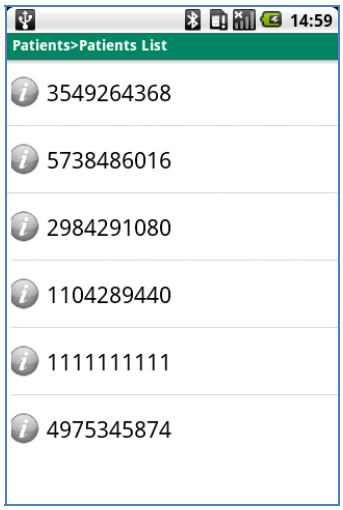


Figure 3.54 Patient List, choose the patient to delete

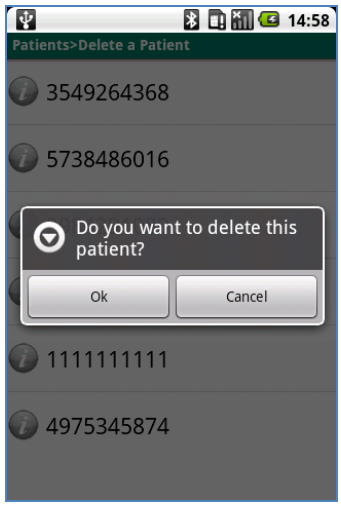


Figure 3.55 Dialog to delete

Figure 3.56 Delete successful

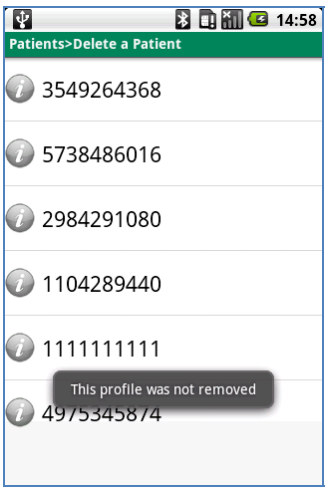


Figure 3.57 Delete failed

The methods involved in the delete process (**removeDirofPatient** and **deleteDirectory**) are in the **SDCard** class. If the patient who the user wants to delete is the last patient in the list, in addition to the patient file the patient directory is also deleted.

As you can see in the Figure 3.55 before deleting any patient, the application will ask the user if he really wants to delete the patient by using an alert dialog. If so, next code will be executed.

```
.setPositiveButton(R.string.dialog_ok, new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int whichButton) {
        //Remove the selected patient
        boolean isRemoved=SDCard.removeDirofPatient(SelectedID);
        if (isRemoved){
            SelectedID=null;
            Intent in = new Intent();
            in.setClass(DeletePatient.this, DeletePatient.class);
            startActivity(in);
            finish();
        }else{
            Toast.makeText(getApplicationContext(),
                    getString(R.string.delete_not_successful),
                    Toast.LENGTH_SHORT).show();
        }
    }
})
```

As indicated in the code, after deleting the patient by his identification number, the application will check if the delete process was correct and finishes the activity. Immediately after the deletion the patient activity is initiated again.

The user can see that the previous patient list has been updated automatically, providing a sense of dynamism in the program. In case that the deleted patient was the last patient in the list, the user will be informed that there are not more patients in the memory.

Access to the patient data

In this section it is described how the user can access to all data of the patients in the application.

After the item “Information & Data” of the activity is pressed (see Figure 3.47) another activity will initiate. This activity shows a list of all the patients available in the SDCard.

It should be mentioned that in case of there isn´t an external storage memory or there aren´t patients stored in the mobile, the application will inform this to the user.

The illustrated list is generated by the method **listDirPatient** in the class **SDCard**. Subsequently a **MyCustomAdapter** is created which is responsible of setting the *String[]* returned by the **listDirPatient** with the listView of the **ListActivity** (for more

information about the creation of the activities and layout see chapter *3.6.4. MyCustomAdpater and PopupWindow*). This **ListActivity** is in the class **PatientList**.

Figure 3.58 Patient List

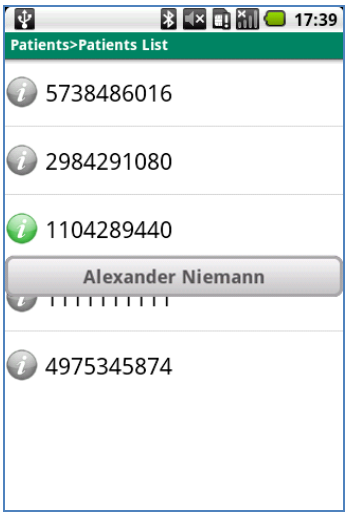


Figure 3.59 Patient List with PopUp Window

As shown, these two activities are ListActivities [50] and the ListView consists of an image and a TextView (thanks to the class **MyCustomAdapter**).

The ImageView Listener is waiting for the moment when the user clicked the image to change the color of it and besides show the name of the patient. This action is performed by the class **PopupWindow**.

Furthermore, the ListView also shows a TextView whose Listener will take the user to the **InfoOrChart** activity.

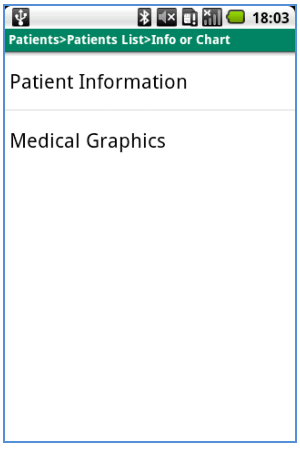


Figure 3.60 Patient Information or Medical Graphics

As shown, the data of the patient has been divided as:

- *Patient Information*: detailed patient personal data, medical record and information about close relatives.
- *Medical Graphics*: detailed pulse and activity/altitude values (received from the HDH wrist device by HL7 messages).

Patient Information

Patient Information is a TabActivity [51] (its design is described in the chapter *3.6.5. TabActivity*) which provides three tabs. Each tab shows information about already mentioned issues.

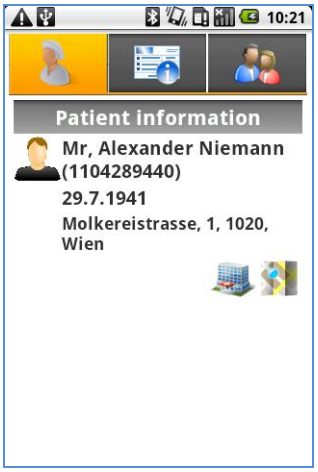


Figure 3.61 Patient Information Tab

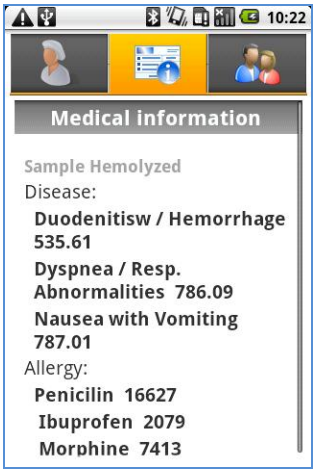


Figure 3.62 Medical Information Tab

Figure 3.63 Relatives Information Tab

These screens display information of the patient if it is available (the data will be read from the corresponding XML file). To do this, a PI type HL7 message must have been received. Otherwise default values are shown in the screen.

In addition to information related to the patient, the user can also perform a search of the address of the patient home or even search of the closest hospitals to the patient´s home using a Google Maps application. Besides, the user can also call to the patient relatives by pressing only one button (without need to look up and dial the phone number of relatives).

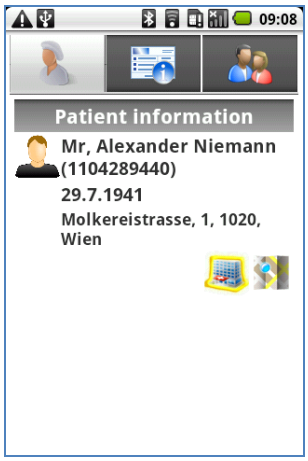


Figure 3.64 Hospital icon clicked

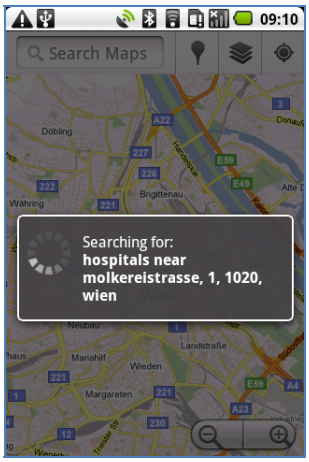


Figure 3.65 Searching for hospitals near the patient address

Figure 3.66 Hospitals near the patient address

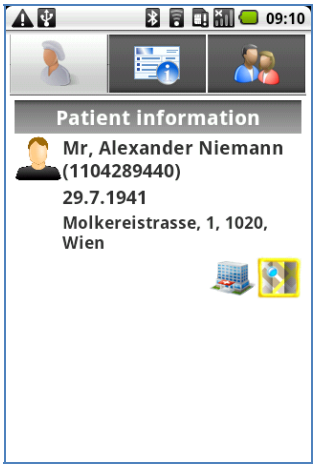


Figure 3.67 Google maps icon clicked

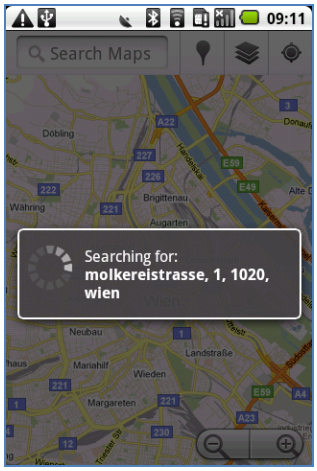


Figure 3.68 Searching for patient address

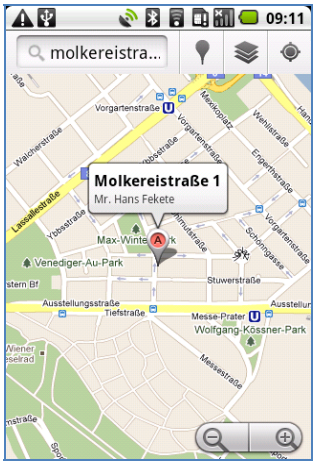


Figure 3.69 Patient address

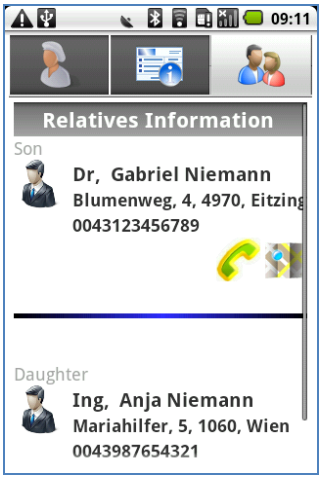


Figure 3.70 Icon Call to a relative clicked

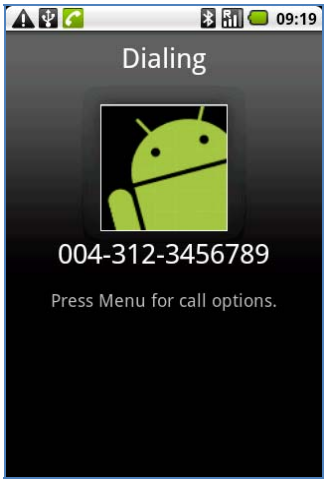


Figure 3.71 Calling to the relative

For this reason there is an Intent defined by a specific action and a URI parameters. In Android it is possible to init known programs as Google applications, making calls, start

the browse etc. by setting appropriate corresponding action and URI parameters. The available standard Intents are shown in the webpage of Android [52].

For instance, the necessary Intents for searching application or for making calls are:

```
Intent mapintent = new Intent(Intent.ACTION_VIEW,Uri.parse("geo:0,0?q="+add1));
```

```
Intent callintent = new Intent(Intent.ACTION_CALL);
callintent.setData(Uri.parse("tel:" + num2));
```

Here, ACTION_VIEW displays the data to the user. This is the most common action performed on data. It is the generic action you can use to get the view of the information that is wanted. The URI “geo:0,0?q=” opens the maps application to the given location or query.

In case of the Intent for making calls, the action is ACTION_CALL and data for URI are defined with this format: tel:telephonenumber [21].

Medical Graphics

On the other hand, if the focus returns to the activity that differed between Patient Information and Medical Graphics, and the button is pressed to init the activity for medical data graphics, a Calendar can be seen.

Calendar

This Calendar will show the current month with the days corresponding for each month. Besides, the days with medical data available are marked with a red point. See the figure:

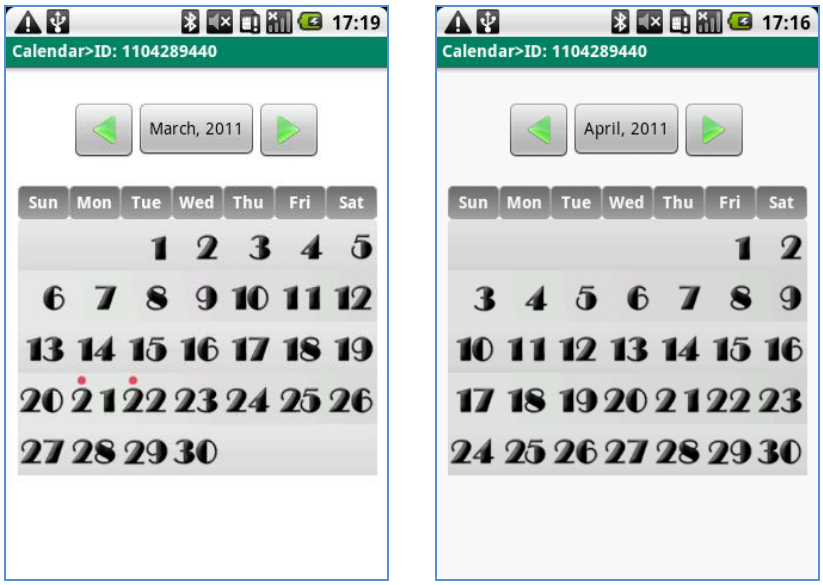


Figure 3.72 Medical data available on the 21st and 22nd of March

Figure 3.73 No medical data available in April

Furthermore, this feature allows the user to move in every month and so, all the months and days where data is available can be seen.

In order to give the mobile user a good usability this calendar file storage system was developed. This Calendar was entirely developed without external libraries. This functionality allows searching the medical data in the external storage faster because it looks for the specific file in the month directly.

Now the development of the Calendar is being explained.

The construction of the layout was done in a way that it can be used for any month of every year. Thanks to that, there is only one layout and it is not necessary to change the layout of the activity each time the user changes the selected month.

The drawn layout is a **RelativeLayout** [53] with a 7x7 **TableLayout** [54] inside. In this **TableLayout** are shown the **ImageViews** which refer to each day of the month. These **ImageViews** are set in a different way depending on the month that the user is examining. For example, each first day of each month is not the same day of the week (For instance, the 03/01/2011 is Tuesday and 04/01/2011 is Friday). Besides, neither all months have the same number of days nor every February month has 28 days (keep in mind that there are leap years).

The first time that the layout is loaded, it is needed to be checked which one is the current month and with which day of the week the current month starts.

```
// Get the actual year and month
Calendar cal=Calendar.getInstance();
SimpleDateFormat month=new SimpleDateFormat("MM");
m=Integer.valueOf(month.format(cal.getTime()));
SimpleDateFormat year=new SimpleDateFormat("yyyy");
y=Integer.valueOf(year.format(cal.getTime()));


// The first day of each month is: Sunday, Monday...
GregorianCalendar gc=new GregorianCalendar(y, m-1, 1);
index=gc.get(Calendar.DAY_OF_WEEK)-1;
```

```
// Paint the correct month, 28, or 29 in February, 30 or 31 in the others months
if (m==1 && m==3 && m==5 && m==7 && m==8 && m==10 && m==12){
    ViewThirtyOneMonth ();
}if  (m==2){
    ViewTwentyEightMonth(gc.isLeapYear(y));
}else{
    ViewThirtyMonth ();
}
```

Depending on the number of days of the month and value of the index attribute (this index refers to the position of the first day of the month in the Table Layout, for example, 1st May 2011 is Sunday and its index will be 1, but 1st June 2011 is Wednesday and its index will be three, and so for every month), the most appropriate **ImageViews** are loaded in order to show the month with its adequate number of days. Of course, each day number has to fit with the specify day of the week.

Furthermore, every time the days of the month are shown, a Thread is launched. It will search all the data available for this month. This action is performed in the **RefreshCalendar** class.

```
// Days with data
RefreshCalendar rfc=new RefreshCalendar(y , m, selectedID, mHandler);
rfc.start();
```

It has to be mentioned that the activity displayed to the user will be updated when the Thread returns the information about the days where there are patient medical data available. This action will be performed by the Handler in the class **Calendar.**

While the Thread **RefreshCalendar** is running it is necessary to keep the same both the variables shared by the Thread and the activity (underbinding that the user could try to access to medical data from another month different from the already chosen/in course one). That is why the buttons “next” and “previous” in the month do not reply when the Thread is running. To avoid problems the Thread is limited to one second which updates the current month with entries of medical data as red dots at the corresponding days and the buttons “next” and “previous” are only available after this Thread has finished.

For this reason, in order to update the screen, the execution of the Thread is controlled by a boolean variable acting as a semaphore. While this variable has the value true this means that the Thread hasn´t finished yet.

```
private final Handler mHandler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {
        case DAYS_WITH_INFORMATION:
        ArrayList<String> infodays =  (ArrayList<String>) msg.obj;
        ArrayList<String> cleandays =CleanRepitedDays(infodays);
        ChangeViews(cleandays);
        CalendActivity.this.dayswithinfo=cleandays;
        infodays.clear();
        // Set to false
        isBusy=false;
        break;
        }
    }
};
```

Once the Thread is finished, the days with medical data inside are highlighted. If there would be several messages for the same day in the SDCard, it is necessary to delete the repeated days from the stored list (in order to avoid updating more than once the same red dot). The next step is to change every image of the **ImageView** related with the day *"without data"* and put in the **ImageView** another image corresponding to day *"with data"* (the day number with a red point in the top left, for example, day1.png is changed to day1s.png etc., drawable-hdpi folder). This task is done in the method **ChangeViews**.

Therefore, thanks to the combination of the Thread, Handler and this method, it is possible to achieve a Calendar which can change dynamically.
Once the data of the Calendar is shown in the activity, you can see that if you press any day (both day with medical data or without medical data) the Calendar replies to it.

This action requires checking the list of days with medical data (which were given back to the program, as said before, by the Thread responsible of the search to update the interface, **RefreshCalendar**). This list must be stored until the activity ends or until the user chooses to view data from other month, then, the list has to be replaced by a new one.

Thanks to the existence of this list (always available for the program for any day the user selects in the current month), it is avoided to have to launch a Thread every time the user changes the day and search again in all files of the SDCard to prove if there are data for this specific day. This way to program the code makes the Calendar very efficient and the program´s reply when a day is pressed is very fast. By doing so, the program has to read the SDCard only once per month (in the month selected by the user).

Since all ImageViews of the TableLayout [54] share the same Listener, when a day is pressed the program has to find out which element of the screen has been clicked and

compares the position of the View returned by the **OnClickListener** method. It also must know which one is the current month in order to obtain the day number.

When the number of the day is obtained, the program is ready to search in the list of "days with data" if this specific day is there. The program will act in one of these ways depending on the answer:

- It will show a Toast to the user, indicating that there is no data available for this day. In this case for example, the user has clicked the day 15th of March, and there isn´t medical information for that day.

Figure 3.74 Not medical data available for this day

- It will go to the next Activity. See that when you pressed the day "with data" (day with red dot), the box of this day changes its color (turns red), then it will go to the next Activity.

Figure 3.75 The user pressed 24th of March and the day is highlighted

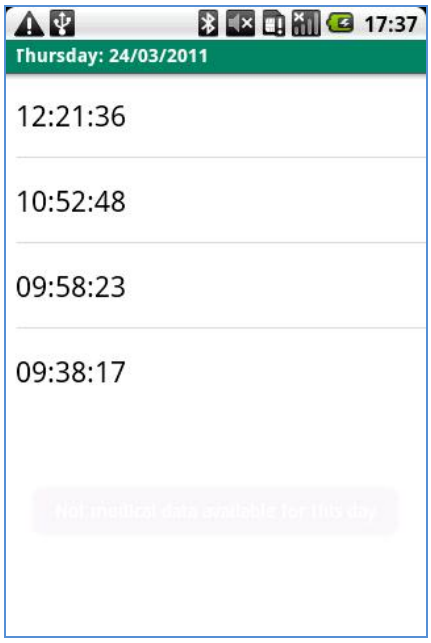


Figure 3.76 List of medical messages stored for the selected day

If the selected day has data, the program continues in the next activity, **Listhours**. It will show all the available hours of the different HL7 messages received in this day (see the Figure 3.76).

The data which shape the ListView are obtained from the SDCard using the **getListDays** method. This time, aside from specifying the year and month, it is also necessary specifying the day. When they are defined, the list is updated with all the hours in the correct format hh:mm:ss.

When an item of the List is pressed (Figure 3.77), a new activity starts, **SelectChart**. As a curiosity, this activity (Figure 3.78) is the unique which has a list with a title and description of the title. Once the user selects the desired chart, the data of the HL7 messages will be shown. The procedure to display the graphs in the Android platform is explained in the chapter *3.5 Graphs*.

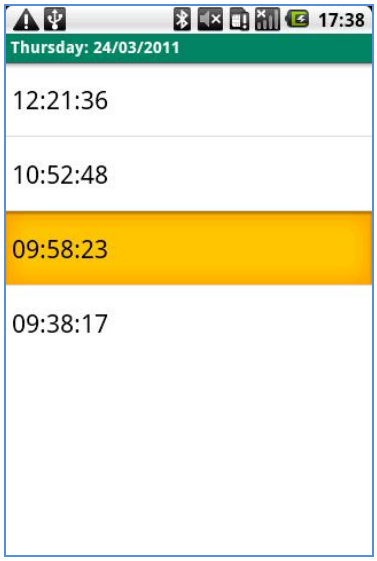


Figure 3.77 Specific medical information selected

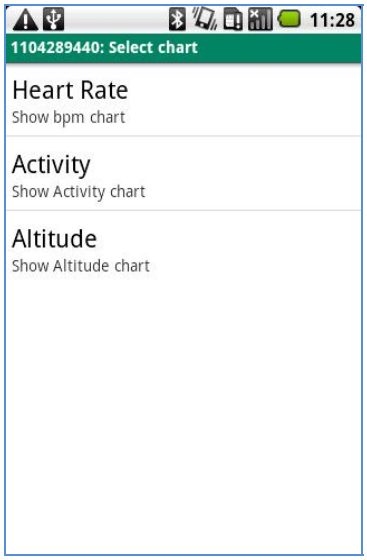


Figure 3.78 Different types of graphs available. Select Chart.

3.5 Graphs

One of the targets of the developed application was that the data of the patient had to be shown in graphs.

It is obvious that having the possibility of seeing the data in graphic form is very attractive, because in this way the user can just have a quick look at it and see the most meaningful data and the tendency of the vital samples of the patient. The user can note the temporal evolution of the patient because every received message is stored depending on the date-time of the reception as well as depending on the information type: “Pulse”, “Activity” or “Latitude” of the monitored patient.

Since Android has a basic graphic package (**graphics** [55]), which provides low level graphics tools such as canvases, color filters, points, and rectangles, in order to implement the graphics in this application the code should have been written almost from scratch.

Therefore, for the purpose of implemented nice graphic views, an open source library was searched in the internet. The used library is **achartengine** [56], an open source library developed by 4ViewSoft.

The truth is that the implementation of the graphs in Android is an issue which is developing continuously. Nowadays, this library provides line, area, scatter, time, bar, pied, bubble, doughnut, range bar and dial charts.

When this development started there wasn´t yet implemented zoom functionality in this library but it had been added months later. That is why the graphics code was rewritten again.

In this chapter it will be explain in detail how the code related with the last version of the graphics in the mobile server was done. The classes related with it are in the **com.upna.AndroidServer.Chart** package. As mentioned above, the used library is **achartengine-versionconZoom.jar**.

In the developed application, once the user selects the date-time of the data of the patient (Figure 3.77) that he would like to see, an activity asking for the type of information appears as shown below. The user can choose between seen the “Activity” or “Altitude” or the “Pulse” data (see above Figure 3.78). The class related with this screen is **SelectChart**.

Here is shown the code of the **SelectChart**:

```
mMenuText=new String[] {"Heart Rate", "Activity", "Altitude"};
mMenuSummary=new String[] {"Show bpm chart", "Show Activity chart", "Show Altitude chart"};
setListAdapter(new SimpleAdapter(this,
        getListValues(),
        android.R.layout.simple_list_item_2,
        new String[] { "Title", "Description" },
        new int[] { android.R.id.text1, android.R.id.text2 }));

private List<Map<String, String>> getListValues() {
    List<Map<String, String>> values = new ArrayList<Map<String, String>>();
    int length = mMenuText.length;
    for (int i = 0; i < length; i++) {
      Map<String, String> v = new HashMap<String, String>();
      v.put("Title", mMenuText[i]);
      v.put("Description", mMenuSummary[i]);
      values.add(v);
    }
    return values;
}
```

Once the type of information is selected, the data as: patient ID, name of the file related with the desired date-time, as well as the information type, are sent by intent to the class responsible for showing the graph, **DoGraph**.

```
@Override
protected void onListItemClick(ListView l, View v, int position, long id) {
    super.onListItemClick(l, v, position, id);
    DoGraph chart=new DoGraph();
    Intent intent = null;

    switch (position){
    case 0:
        intent = chart.execute(getApplicationContext(), file, position);
        break;
    case 1:
        intent = chart.execute(getApplicationContext(), file, position);
        break;
    }
    startActivity(intent);
}
```

The class **DoGraph** extends from **AbstractChart** class and **AbstractChart** implements **IChart** which is an interface. This is why the methods of the **IChart** (getName, getDesc and execute) are defined in **DoGraph**.

Once the data to create the graph are taken by an intent (execute) in **DoGraph**, the code will read the specific file where the desired information is. In this file, the relevant HL7 message is stored, which will become an HL7 object (using the **getHL7** method in the **FrommessageinSDCard** class).

In **DoGraph** there are also defined details as the titles of the axis, colors of the representation or the type of the points desired.

By the method **GetAllDatas** of the **ExtractDatafromFiles** class, the data of the “Activity”, “Altitude” or “Pulse” related to the created HL7 object is obtained. The date

and starting/ending time of the measurement with the appropriate format are obtained shown in the legend of the graph.
In order to create the x axis, the code works with the starting/ending time of the measurement (in millisecond). Obtaining the subtraction of them and dividing with the number of samples of the measure, after rounded the sample frequency of the graph is obtained. In this way, it is possible to represent each point in the graph with the corresponding millisecond.

After estimation of all of this, the data is available to show in both the x axis and y axis. Then, the method **buildRenderer** is used to get a renderer as shown here:

```
// Start the renderer
XYMultipleSeriesRenderer renderer = buildRenderer(colors, styles);
int length = renderer.getSeriesRendererCount();
for (int i = 0; i < length; i++) {
  ((XYSeriesRenderer) renderer.getSeriesRendererAt(i)).setFillPoints(true);
}
```

This renderer, is a **XYMultipleSeriesRenderer** type object, which is obtained by the **buildRenderer** of the **AbstractChart** class.

```
/**
 * Builds an XY multiple series renderer.
 *
 * @param colors the series rendering colors
 * @param styles the series point styles
 * @return the XY multiple series renderer
 */
protected XYMultipleSeriesRenderer buildRenderer(int[] colors, PointStyle[] styles) {
  XYMultipleSeriesRenderer renderer = new XYMultipleSeriesRenderer();
  int length = colors.length;
  for (int i = 0; i < length; i++) {
    XYSeriesRenderer r = new XYSeriesRenderer();
    r.setColor(colors[i]);
    r.setPointStyle(styles[i]);
    renderer.addSeriesRenderer(r);
  }
  return renderer;
}
```

Using this renderer, it is possible to link both color and style of the points for the series of the graph.

The next step is to set all the settings of the chart using the method **setChartSettings** (the renderer, ID, titles, times, data, min and max values, colors of the lines in the axis…) as well as the labels.

```
// Set the settings of the chart
setChartSettings(renderer, HL7objects.get(0).getPid().get_patientIdentifierList(),
        "", titleaxisy, x.get(0)[0].getTime(),
    x.get(0)[AllDatas.get(0).length - 1].getTime(), minValues-10, maxValues+10, Color.WHITE, Color.WHITE );
renderer.setXLabels(8);
renderer.setYLabels(10);
renderer.setShowGrid(true);
renderer.setXLabelsAlign(Align.CENTER);
renderer.setYLabelsAlign(Align.RIGHT);
Intent intent = ChartFactory.getTimeChartIntent(context, buildDateDataset(titles, x, AllDatas),
    renderer, "h:mm:ss a");
return intent;
```

At last, it is necessary to create an intent in such it is possible to create a TimeChart type graph and which will start the graphical view activity. This is done using the **getTimeChartIntent** of the **ChartFactory** class included in the already mentioned **AChartEngine-versionconZoom.jar** library.

An example of the medical data received for the patient with ID 1104289440 (Alexander Niemann). Datetime of the information is illustrated also (2010/05/10 from 12:00 to 12:15).

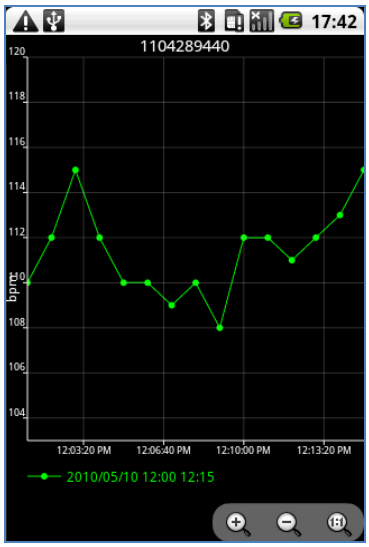


Figure 3.79 Heart Rate

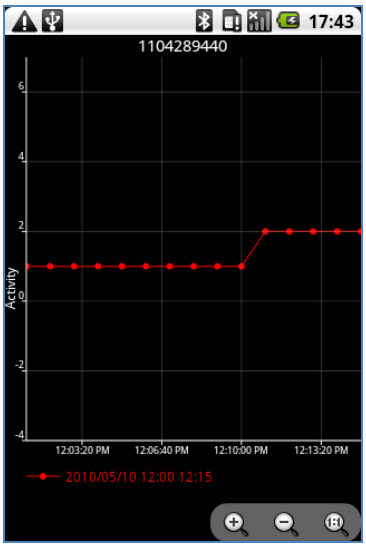


Figure 3.80 Activity

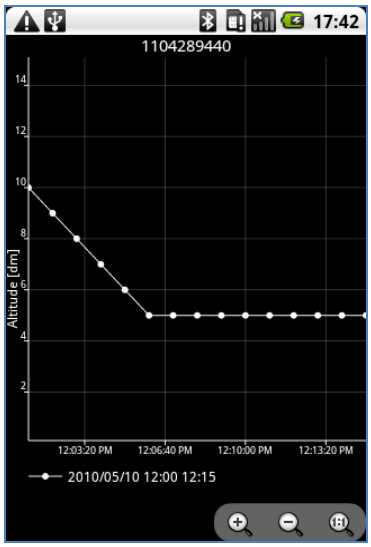


Figure 3.81 Altitude

3.6 Graphical User Interface Design

This chapter deals with the design of the most relevant screens in the Android Server application. The mobile phone used for developing this application is HTC Magic [57].
In the Android platform, three essential features come into play during the design of graphical user interfaces (GUI):

- The Theme of the activity (set by the Android Manifest [45], see also *Annex III. AndroidManifest*).
- The Layout of the activity.
- The code programmed in a class which inherits from one of the activities existing in Android.

3.6.1 Custom Theme for Android Server

As shown along the screenshots of this document, the design of the theme generated in the Android Server application is completely different to the theme that the Android platform users usually see in other applications.

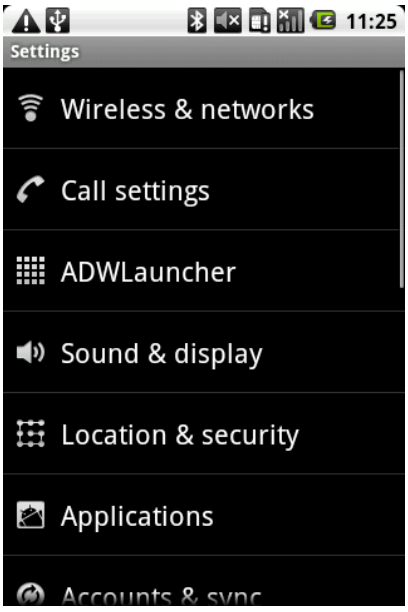


Figure 3.82 Default theme in Android

Figure 3.83 Custom theme in the Android application

You can notice that the background of the screen has been changed. The chosen color was white because it is good suitable for a biomedical application. Besides, the title of the application has been changed to green color in order to bring a nice touch to it, because somehow, it is related with the logo of the University where the development of this Thesis takes part.

Firstly, the Android Manifest should be set so that the activity uses the custom designed theme (more in *Annex III. AndroidManifest*)

```
<!-- About Us -->
    <activity android:name="com.upna.AndroidServer.AboutUs.AboutUs"
              android:theme="@style/customTheme">
    </activity>
```

The theme is created in the theme document in the folder resources/values. As shown in the code above, the Android Manifest refers to this theme.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="customTheme" parent="android:Theme.Light">
        <item name="android:windowTitleBackgroundStyle">@style/WindowTitleBackground</item>
    </style>
</resources>
```

As noted in this code, there is a style associated to the item windowTitleBackgroundStyle. This style is responsible of setting the green color of the theme. The parent parameter is set in white color (the default in Android is black that is why is written .Light).

It is also needed to create a Layout with a TextView inside. The parameters of this TextView will set the format of the letters in the title. The design of this Layout, called **myactivitiestitle.xml**, is saved in resources/layout.

Once the design of the Layout is finished, it has to be loaded like this:

```
// Request the Window to change the theme
this.requestWindowFeature(Window.FEATURE_CUSTOM_TITLE);
setContentView(R.layout.aboutus);
getWindow().setFeatureInt(Window.FEATURE_CUSTOM_TITLE, R.layout.myactivitiestitle);
TextView title = (TextView) findViewById(R.id.myTitleofActivity);
title.setText("About Us");
```

Here, the features of the window are requested and the layout of the activity which is to be shown is loaded (this part is different in each activity because the layouts of every activity are different; in the example the activity is “About Us”). Then, the message that will be shown in the title is set. The green color of the title is given by the Android Manifest because the theme was defined before.

3.6.2 Android Server Home

This screen is the “welcome” or “home” screen in the Android Server application. The design of this screen attempts to simulate some famous structure of “Twitter”. It was decided to choose this way because it brings a special touch to the application because the user is used to handle this kind of screens.

The design of the interface of the activity is loaded by a layout. In order to explain the interface of this “home” activity, there is below a tree-map of all the views and layouts described and used in the androiserver_home.xml.

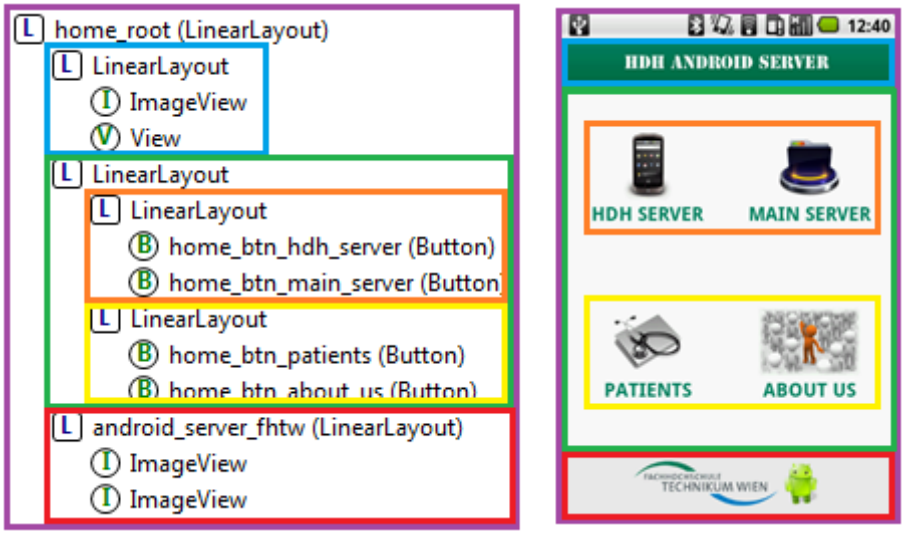


Figure 3.84 Design of the AndroidServeHome activity

This figure shows how the screen has been divided and the different views which take part in, as ImageView or Button. Furthermore, this layout is composed by several Linearlayouts [58] which arrange the views of the screens in different orientations (vertical or horizontal).

The Buttons are designed as follows:

```
<Button android:id="@+id/home_btn_hdh_server"
    style="@style/HomeButton"
    android:text="@string/btn_hdh_server"
    android:drawableTop="@drawable/home_btn_hdh_server"
    android:onClick="onHDHServerClick"/>
```

The buttons of the AndroidServer_Home activity have their own style and their features are set in the same way. The style HomeButton (in styles.xml) sets both the position of the element in the button and the text referred in the code in the text label.

Furthermore, the label drawableTop indicates the position of the image in the button. In this case it is put over the text. Note that the references to the folder drawable aren´t done using a .png image but by one xml file which refers to two different images.

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_focused="true"
    android:state_pressed="true"
    android:drawable="@drawable/home_android_pressed"/>
    <item android:state_focused="false"
    android:state_pressed="true"
    android:drawable="@drawable/home_android_pressed"/>
    <item android:state_focused="true"
    android:drawable="@drawable/home_android_pressed"/>
    <item android:state_focused="false"
    android:state_pressed="false"
    android:drawable="@drawable/home_android_default"/>
</selector>
```

As shown, a selector is described which checks the state of the button. If the button´s state is focused or the button has been pressed, the default image must change and it has to use the highlighted image. In this way, the user has information about pressing the button.

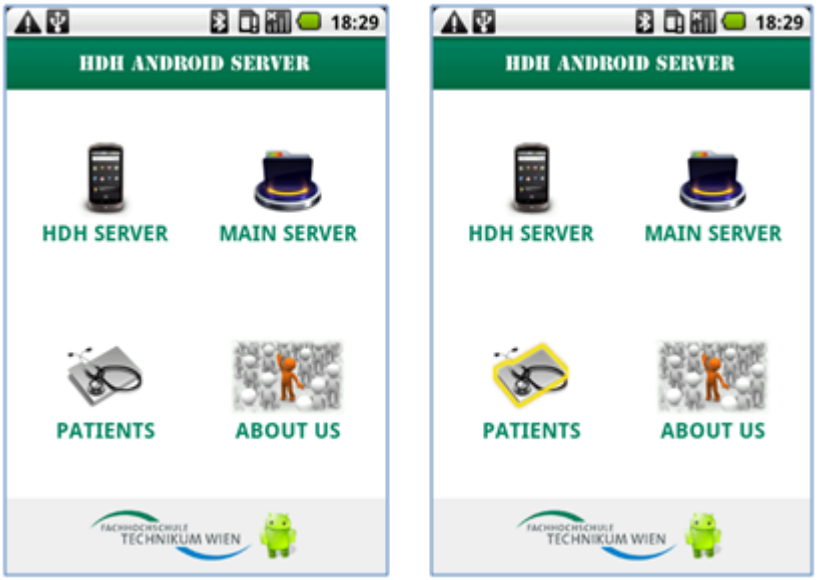


Figure 3.85 User has clicked the PATIENTS button

Continuing with the image of the button, the explanation of the label onClick is given which refers to the method (android:onClick= “onHDHServerClick”). This label registers a listener which will execute a specific method (located in the class where this layout was loaded) once the button is pressed. In this case, the method that it will execute is onHDHServerClick.

On the other hand, in the layout of this activity it can be seen that the custom theme explained in the previous section isn´t used. Instead, a theme of android is used which

allows to have a white screen but without the default title. There is a linear layout (see the blue rectangle in the figure) replacing it.

```
<activity android:name=".AndroidServer_Home"
          android:label="@string/app_name"
          android:theme="@android:style/Theme.Light.NoTitleBar">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

The title in this “home” screen was to be deleted in order to introduce another title with bigger width than the default one, and which also adds a gradient configured in the drawable folder.

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <gradient android:startColor="@color/titlebackgroundcolorgradient"
    android:endColor="@color/titlebackgroundcolor"
    android:angle="90" />
</shape>
```

To conclude this section, it has to be mentioned that these effects, “highlight effect” in the buttons and also “gradients”, are used in other activities in the same way (in the Calendar when the user presses the day with medical information, in the relatives information screen when the user presses the call button or map application button…). The explanation of how to do the coding is the same as explained in previous lines in this chapter.

3.6.3 Bluetooth Server

The important features in the graphical interface in the activity of the Bluetooth Server are its progress bar and popup menu.

Note that the design of this activity is the same as the Wifi Server activity.

Progress Bar

In the Android platform there are several ProgressDialogs [59] available, which show that an activity is processing or that there is a task working in background.

Nevertheless, it was necessary to show that the Bluetooth Server was running. Due to this fact, the best option was to develop an own **ProgressBar**. It was decided to use an icon which resembles an envelope in the **ProgressBar**, which would spin (form 0º to 360º) while it was approaching the right side of the screen. In this way, the user simply

has to look at it and will notice that the service is running. This progress bar that was designed, gives dynamism to the application and it is attractive to the user. The class which deals with this issue is **MyProgressBar** located in the package **com.upna.AndroidServer.views.**

The class **MyProgressBar** extends from LinearLayout due to the fact that this view has to be introduced later in the layout and also implements Runnable because the movement of the progress bar is controlled from a Thread.

In order to achieve the performance of this progress bar, an object named **MyProgresBar** is created in the activity where we want to insert it. In this case the activity is **BluetoothManager**, which automatically will prepare the view using an object of the class **LayoutInflater** [60]. This object will use the layout **myprogressbar**.

```
LayoutInflater inflater = (LayoutInflater) context
        .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
View view = inflater.inflate(R.layout.myprogressbar, null);
addView(view);
```

The layout **myprogressbar** consists in a **RelativeLayout** with five imageviews (each with an image, circle_grey.png).

After adding the view, the method **startAnimation** is called to initiate the thread. This thread will be sleeping for 300ms and a message is sent to the Handler implemented in the **MyProgressBar** class to update the progress bar (more specifically it updates some images of it and so, it will simulate a bar moving from left to right). Once this is done, the thread sleeps again and when it wakes up, another message will be sent to the Handler in order to update again the progress bar (new images of the progress bar are updated now and the optical animation for the user is that the progress bar is moving). This is done until the thread is cancelled. The view of MyProgressBar is cancelled by the method **dismiss**.

Once the performance of the class **MyProgressBar** is explained, the next step is to analyze how the incorporation of the progressbar to the desired activity works.

First of all, the layout of the activity is added, in this case the layout **bluetoothmanager**.

```
<com.upna.AndroidServer.views.MyProgressBar
    android:text="@+id/TextView01"
    android:id="@+id/progressBar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:layout_marginTop="25px">
</com.upna.AndroidServer.views.MyProgressBar>
```

Note that where usually the name of the view is written, now the complete path is shown.

Once the view is loaded in the layout, it is also needed to link it to the code in the background. In this way, an object is obtained which controls the view, i.e., it can start and stop the view.

```
pBar = (MyProgressBar)findViewById(R.id.progressBar);

private void start(){
    pBar.startAnimation();
}
private void stop(){
    pBar.dismiss();
}
```

This is the code which relates the view loaded in the layout with the code of the activity. Both methods start and stop the view.

It is very important to realize that the progress bar must be stopped in the correct moment. In this case, it will be stopped when the user goes out of the activity (**onDestroy**) or the service stops.

On the other hand, the progress bar will be executing in these cases: obviously, when the service is initiated by the user first time, or when the user accesses to this screen and the service was already running before.

Popup Menu

The Android platform gives us the possibility of showing the popup menu [61] which allows hiding some tasks if the screen is already full of them or if the action is of secondary meaning.

This function was included in the **BluetoothManager** activity in order to add the possibility to connect to the HDH (see chapter *Android mobile working as a client with regard to the HDH device*) in the developed Android Server application. As explained there, this function is a task that was included in the application to satisfy the current requirements of the HDH wrist device. For this reason, this extra function is included in a popup menu. See Figure 3.11.

In order to create the menu it is essential to create the menu folder in the project directory (because the menus are stored in this folder, as there are other folders as values, colors etc.). The created menu is called **option_menu** and it is loaded by the class **LayoutInflater**.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/scan"
          android:icon="@android:drawable/ic_menu_search"
          android:title="@string/connect" />
    <item android:id="@+id/discoverable"
          android:icon="@android:drawable/ic_menu_mylocation"
          android:title="@string/discoverable" />
    <item android:id="@+id/disconnect"
          android:icon="@android:drawable/ic_lock_power_off"
          android:title="@string/disconnect" />
</menu>
```

This menu is compounded by three items (see Figure 3.11). Each of them has associated an icon of Android platform and a title.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu, this part is for android like a client
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.option_menu, menu);
    return true;
}
```

This method, **onCreateOptionsMenu**, of the class **Activity**, is overwritten in **BluetoothManager** and allows creating menus. Besides, the method **onOptionsItemSelected** also has to be overwritten to execute the action that must be done after pressed one of the items of the menu. If you want to know more about each of these items go to *3.1.1 Bluetooth Technology*. Once it is done, the menu is added to the activity.

3.6.4 MyCustomAdapter and PopUpWindow

These classes enable to build a kind of list similar to one below. One class is responsible of showing the list and the other to respond when an icon of the list has been pressed.

MyCustomAdpater

MyCustomAdapter is a class generated in the activities, ListActivities are composed by an ImageView and a TextView in each row. This class extends from **ArrayAdapter<String>** because it is needed to adapt a String type list to a ListView.

An example of this issue is the activity **PatientList**. This activity is analyzed to illustrate the behavior of this class. This ListView consists the patientList layout: composed by a LinearLayout with horizontal orientation and an ImageView and a TextView inside.

In the constructor of this class the overwritten method **getView** is called in order to create a row of the ListView (Figure 3.86). Therefore, if the method is called with a list of 10 elements, it will be called 10 times and the position parameters will indicate the element of the list that is creating in that moment.

```
@Override
public View getView( final int position, View convertView, ViewGroup parent) {
    // TODO Auto-generated method stub

    LayoutInflater inflater=getLayoutInflater();
    View row=inflater.inflate(R.layout.patientidlist, parent, false);

    TextView label=(TextView)row.findViewById(R.id.patientid);
    label.setText(idPatient[position]);

    final ImageView icon=(ImageView)row.findViewById(R.id.info);

    //Set the white info icon
    icon.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                PopupWindow dw = new PopupWindow(v, idPatient[position], mHandler);
                dw.showLikePopDownMenu();
                icon.setBackgroundResource(R.drawable.infogreen);
            }
        });

    return row;
}
```

As established in the code, a **LayoutInflater** [60] is used in the patientlist layout and so, the view called row will dispose an ImageView and a TextView. A patient identifier number will be assigned to this TextView (a number which correspond in the idPatient[position]). The String[] idPatient has all the identification numbers of the patients that are currently in the external storage system and as explained before, the position parameter is given by the method **getView**. The ImageView is set by the grey information .png as shown below:

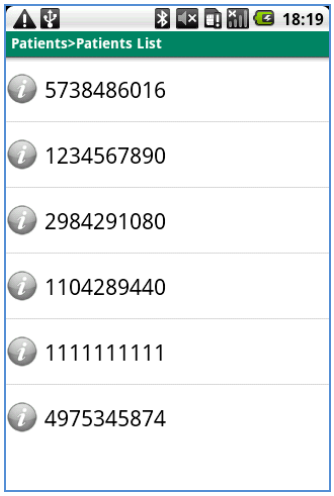


Figure 3.86 List created with MyCustomAdapter

Furthermore, a listener is set to each ImageView and when it is pressed it will named the class **PopupWindow** (it is explained in the next section). Then the color of the information icon will change to green, advising to the user which is the chosen patient (owner of the current PopUp).Once this is done, the list will be illustrated in the screen.

PopupWindow

The **PopupWindow** class is responsible of creating a pop up which will link a patient identification number with the appropriate name and surname. In order to explain this issue, the acitivity **PatientList** is used.

The **PopupWindow** class extends from **BetterPopupWindow** which is in the package **com.upna.AndroidServer.views**. Both are responsible of showing the Popup.

The PopupWindow becomes a character in the code when the user clicks an ImageView of the ListActivity of the Figure 3.86. Then, the method **onClick** of the listener of the ImageView executes. It changes the color of the information icon to green and the following actions start:

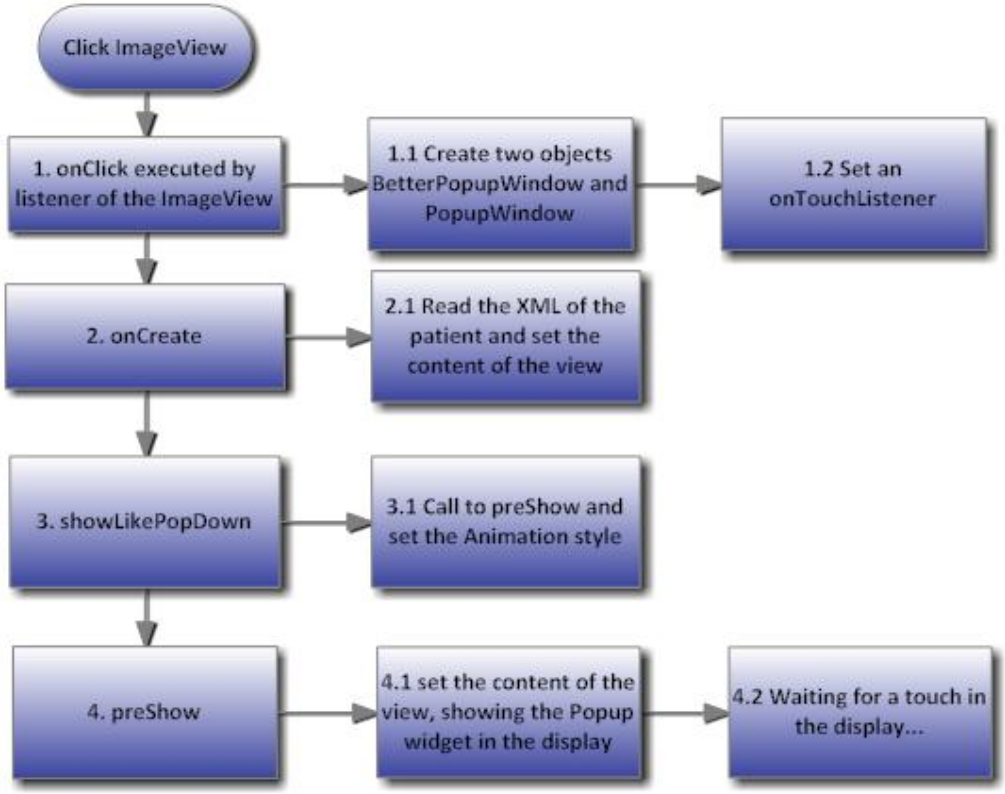


Figure 3.87 Schema of the PopUp Window performance

Note: the realized actions are marked by numbers in sequential order, i.e., the step 1.1 is done inside the step 1 and so on.

Once these actions are done, the following image is shown in the application. The patient identification number 1104289440 is related with the name/surname Alexander Niemann.

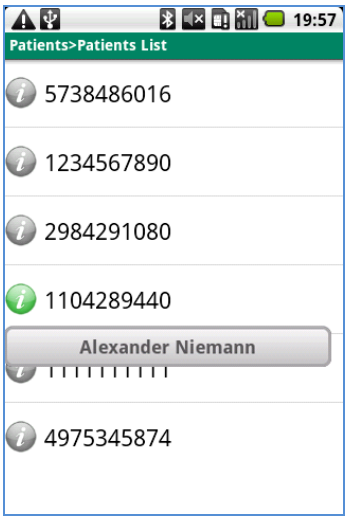


Figure 3.88 Popup Window

Then, when the user clicks anywhere in the screen, the event will be captured by the set **onTouchlistener** and the next code will execute:

```
@Override
public boolean onTouch(View v, MotionEvent event) {

    if(event.getAction() == MotionEvent.ACTION_DOWN) {
        //android.os.Debug.waitForDebugger();
        BetterPopupWindow.this.window.dismiss();
        // Communicate to the UI that something has happened
        BetterPopupWindow.this.mHandler.obtainMessage(0).sendToTarget();
        return true;
    }
    return false;
}
```

This code makes the Popup disappear and then a message is sent using a Handler to the graphic interface; refreshes the ListView and loads it again putting the icon to its default state (again grey color).

3.6.5 TabActivity

The Tabactivities [51] are activities that contain and run multiple embedded activities or views.

To the developed program, it was required to create a TabActivity which shows the most relevant information of the patient (already serialized in the xml file as explained in the chapter *Android Mobile as a Server*, talking about updateXML).

The development of the graphic interface and code is different and more complicated than in other simpler activities. That is why these issues are explained here.

The development of the layout needs a TabHost [62], a TabWidget [63] and a FrameLayout [64].

```
<?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/tabhost"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:padding="5dp">

        <TabWidget
            android:id="@android:id/tabs"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" />

        <FrameLayout
            android:id="@android:id/tabcontent"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:padding="5dp" />
    </LinearLayout>
</TabHost>
```

The TabHost is a container for a tabbed window view. This object holds two children: a set of tab labels that the user clicks to select a specific tab (TabWidget) and a FrameLayout object which displays the contents of that page. This FrameLayout is designed to reserve an area on the screen to display a single item. The individual elements are typically controlled using this container object, rather than setting values on the child elements themselves.

The operation of the TabWidget is this: when the user selects a tab, this object sends a message to the parent container (TabHost) to tell it to switch the displayed page. So the container TabHost is used to add labels, add the callback handler and manage callbacks.

Moreover, the class used to load this type of layout should extend from TabActivity and to add the tabs it should use a TabHost type object and TabSpec [65].

```
intent = new Intent().setClass(this, Medicalinfo.class);
b=new Bundle();
b.putString("SelectedID",selectedID);
intent.putExtras(b);
spec = tabHost.newTabSpec("medicaldata").setIndicator("",
                  res.getDrawable(R.drawable.medicalinformation))
              .setContent(intent);
tabHost.addTab(spec);
```

As shown, it has to ask for a new tab (TabSpec) to the TabHost object and it will set the necessary parameters. In this case, there are given a name, an image and a title. The content has to be added also (**setcontent**) which will set the intent that must do when the user clicks in the tab. Once it is done, it is included in the TabHost.

Concluding, it is all about the generation of the TabActivity.

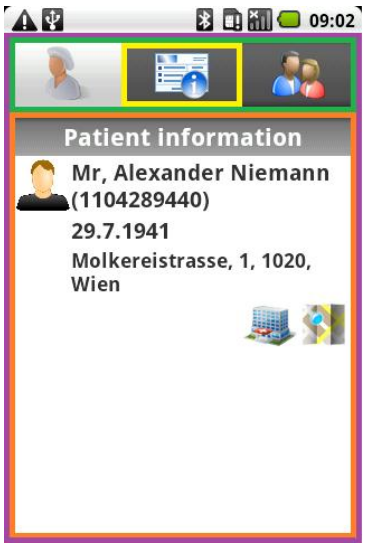


Figure 3.89 TabHost (purple), TabWidget (green), FrameLayout (orange) and TabSpec (yellow)

3.6.6 Calendar

The explanation about the Calendar is given in the chapter *Access to the patient data* section of the *Calendar*. As mentioned there, an activity which shows a Calendar has been developed in order to reveal the days which contain medical information of the patient. In that section is expounded the performance of the code, i.e., how data is searched, the days are loaded etc.
Nonetheless, to realize all the actions it is indispensable to create a layout where all the months are loaded. All about the layout of the Calendar is explained below.

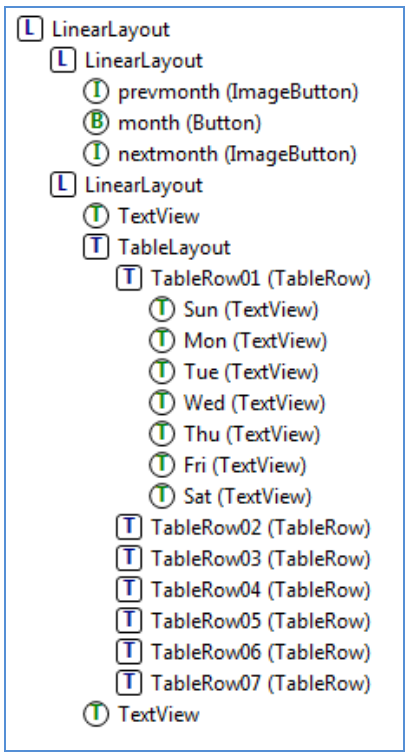


Figure 3.90 Structure responsible of showing the Calendar

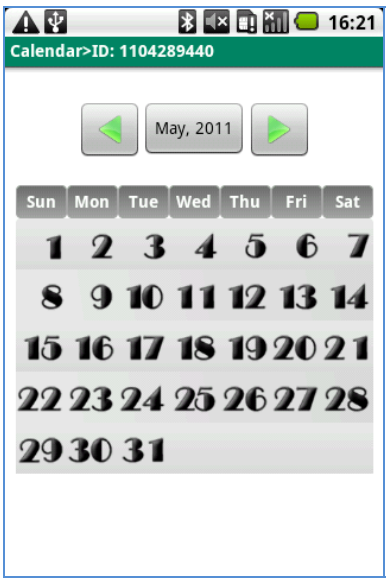


Figure 3.91 Calendar activity

We have several views of different types: ImageButton, Button, ImageView, TableRow.... These views are located in three LinearLayouts [58]. In one of them, there is the Calendar as TableLayout [54].

A TableLayout consists of a number of TableRow objects, each defining a row. In this TableLayout are located in the first row the elements which will be static: the name of the days in the week. In the next rows (TableRow) there are loaded ImageViews which later, by using the code, will have assigned an image depending on the name of the week day related with the first day of the month.

3.6.7 About Us

The activity AboutUs is a screen which presents to the user more about the software developers, where it has been developed, who is the tutor of the project (because it is a Master Thesis Project) and it gives the opportunity to contact the developers by email (see Figures 3.92 and 3.93).

The most interesting fact related with the software development in this activity is its design and the possibility that it gives to open the e-mail by default (or even the user can select in a list the email manager that he wants to use). The e-mail will be opened with the gaps already filled with the developers address.

Firstly, it has to be mentioned that the design has been done using a RelativeLayout [53]. A RelativeLayout is a Layout where the positions of the children can be described in relation to each other or to the parent. This RelativeLayout has two childrens; one is ScrollView [66] (which enables the user to move in the activity) and the other is LinearLayout [58] (where the other Views are set).

In the LinearLayout there are, between several views, two TextViews where a listener was set. Thus, when the user clicks there, they change their style (letter size and color) and the email application initiates. This is done like this:

```
final TextView contactAmagoia=(TextView) findViewById(R.id.tellechea);
contactAmagoia.setOnClickListener(new OnClickListener() {
public void onClick(View v)  {
    //Change the style of the text after press the button
    contactAmagoia.setTextAppearance(getApplicationContext(), R.style.nameemail_pulsado);
    //Launch the email application with the email address filled
    Intent i = new Intent(Intent.ACTION_SEND);
    i.setType("message/rfc822") ;
    i.putExtra(Intent.EXTRA_EMAIL, new String[]{getString(R.string.email_developer1)});
    i.putExtra(Intent.EXTRA_SUBJECT, getString(R.string.subject_developer1));
    i.putExtra(Intent.EXTRA_TEXT, getString(R.string.body_developer1));
    startActivity(Intent.createChooser(i, "Select email application."));
    }
});
```

Figure 3.92 About Us activity

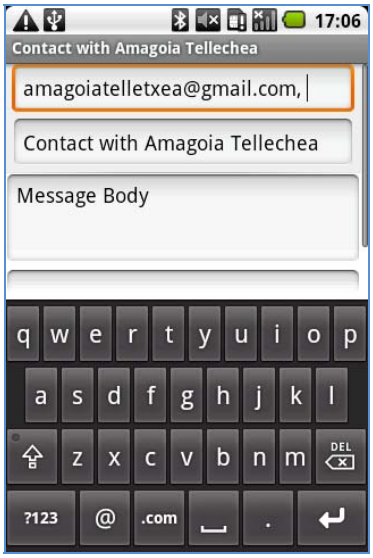


Figure 3.93 Contact with

3.7 Design of simulators

Because this project is a part of a bigger project where other people are working, it was needed to adapt in every moment to the real functionalities finished in that time. That is why during the conduction of this thesis, some programs which act as simulators of other project parts were created in order to be able to test more efficiently the work. In this way it was possible to analyze this program, check the errors and correct them. Thanks to

that, when working with other people and trying connections, message sending/reception and answering… only some details were needed to be redone.

Following there is a brief explanation about the simulators which were created during the development of this master thesis (which became very important for testing purposes).

- **HDH (client) simulator for Bluetooth connection,** for the implemented Android Server Bluetooth application.
- **HDH (client) simulator for TCP/IP connection**, in order to connect to the implemented Wifi Android Server.
- **Android SDK emulator to send GSM messages.** It is the emulator that the Android platform provides.

There were also two simulators for the Android Server application:

- **Simulator of HDH working as the server**, Android phone acting as a Bluetooth client
- **Simulator of the HDH Main Server**.

Due to the fact that both connections are working with real applications (with the real devices; HDH wrist device and Main Server) none of these simulators are included in the next lines because they are not needed currently. Only the first three simulators are explained.

3.7.1 HDH (client) simulator for the Bluetooth connection

This is an application written in C# capable of discover running services and of sending messages to the selected device. See figure below:

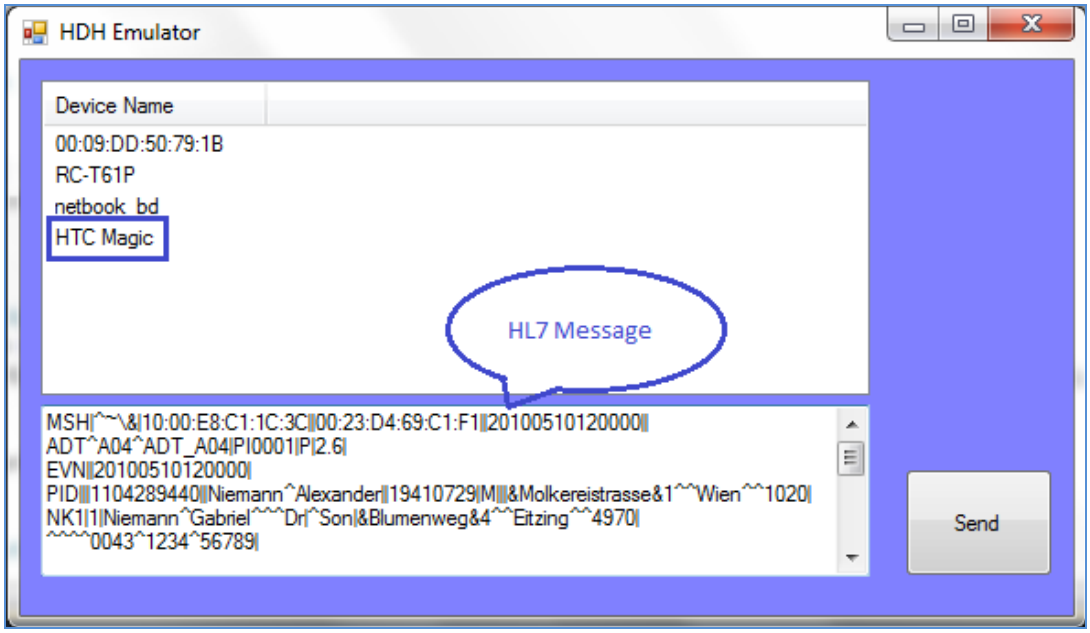


Figure 3.94 Simulator of HDH wrist device

The class **BluetoothClient** in C# has a method called **DiscoverDevices** which allows obtaining a list of all the devices located in the wide range of Bluetooth. This is done in the background with an object of the class **BackgroundWorker**. When it discovers the devices the method **_discoverWorker_RunWorkerCompleted** is called which will list in the ListView all the found devices.

When the user clicks an item of the list above and writes the HL7 message in the text view, the program starts connecting to the Android Server by the method **connect**.

3.7.2 HDH (client) simulator for TCP/IP connection

This schema represents how the HDH (client) should work using the TCP/IP protocol. This functionality will be implemented in future versions of the HDH device.

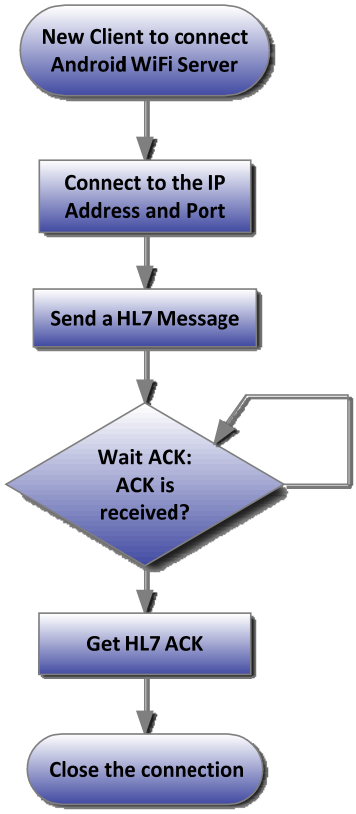


Figure 3.95 Schema of HDH (client) Simulator for TCP/IP connection

The code is available in the project **Quickclient**.

3.7.3 Simulator for GSM messaging

The Android SDK has an emulator which enables to send SMS messages among other capabilities. More information about this tool is available in Android Developers webpage [67] See Figures 4.12 and 4.15.

4 Evaluation and results

4.1 Evaluation

In any project that contains an implementation part, it is imperatively needed to verify the correct behavior of it. In this case, after the implementation of the application debug processes have been done: testing both "black box" and "white box" to ensure the correct behavior of the deployed application.

Black-box test design treats the system as a "black-box", so it doesn't explicitly use knowledge of the internal structure. Black-box test design is usually described as focusing on testing functional requirements [68]. Thus, this test design introduces data and only analyzes the output of the tested object. In general, this test level requires defining all the possible cases and so, the tester only has to verify each output or behavior of the system for a specific input.

Nevertheless, white-box test design enables to peek inside the "box", and it focuses specifically on using internal knowledge of the software to guide the selection of test data. The tester has access to the internal data structures, code and algorithms. The test methods for white-box testing could also use to validate the integrity of the test realized with methods of the black-box. This facilitates the work of the developers, because they can examine parts of the system that rarely would be tested and ensures that all the important points have been analyzed.

4.2 Results

Some of the most important facts of this Master Thesis are the connections achieved with several devices. That is why they have been tested and verified a lot of times when evaluating the software. Some connections had some difficulties that have been successfully solved during the Thesis progress.

Regarding the connections with Android, it is unavoidable to talk about the HL7 package. It is required testing the HL7 package built for Android, which is responsible of parsing the received HL7 messages. It provides various algorithms capable of understanding the messages that are currently running, but due to the fact that the project is continuously progressing, it would be sensitive of possible changes in some features in the future.

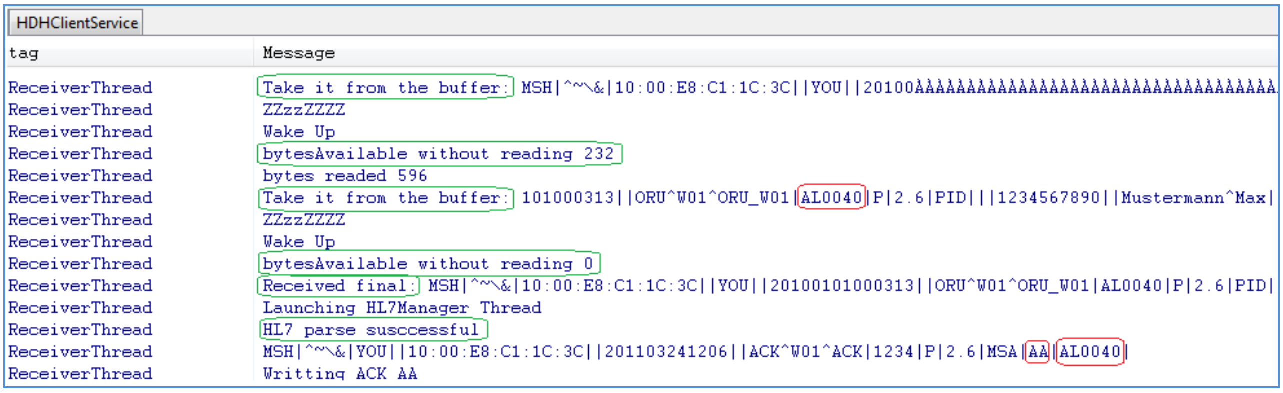
Besides, all the interfaces have been tested and also the internal code which controls them. The small details that caused some kind of error have been solved and now everything works correctly.

With respect to the part of connection between the HDH and Android Server via Bluetooth, as mentioned and explained over the report, several functionalities in the

Android Server have been developed (behavior as client or as server). As shown, the management of the connection in all these cases is similar, except the obvious differences regarding to the device which starts the connection. Once it is established, in the exchange of messages by Bluetooth does not matter who initiated communication. Because of these reasons, the results of them will be explained together.

4.2.1 Communication via Bluetooth between HDH and Android Server

Below there are some views of the Android **log**. The transmission of the messages can be seen: firstly the reception of the message, then the correct parse of it (green) and the ACK dispatched with the same identification as the received message (red).

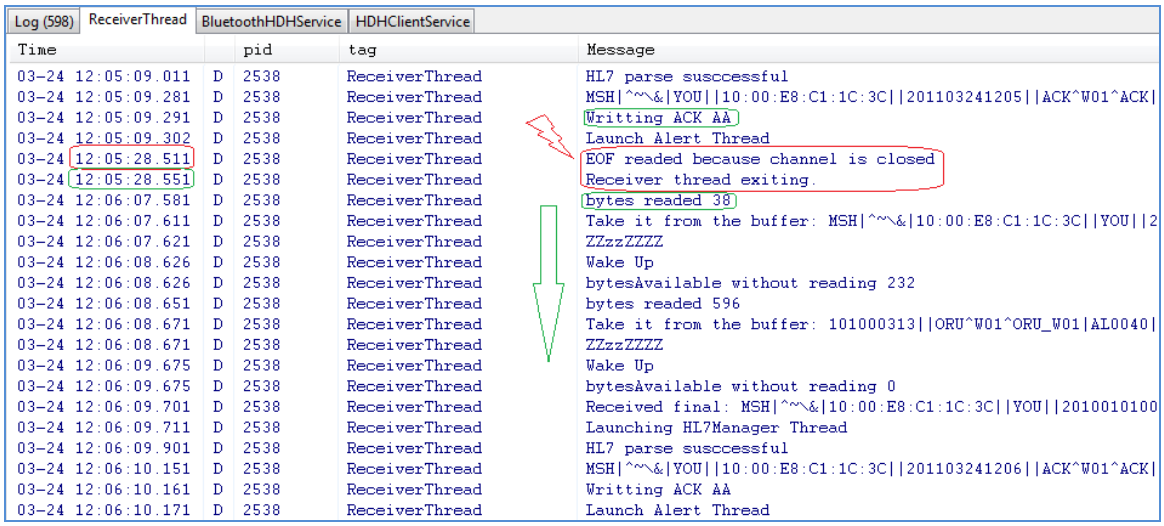


```
HDHClientService
tag             Message
ReceiverThread  Take it from the buffer: MSH|^~\&|10:00:E8:C1:1C:3C||YOU||20100AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ReceiverThread  ZZzzZZZZ
ReceiverThread  Wake Up
ReceiverThread  bytesAvailable without reading 232
ReceiverThread  bytes readed 596
ReceiverThread  Take it from the buffer: 101000313||ORU^W01^ORU_W01|AL0040|P|2.6|PID|||1234567890||Mustermann^Max|
ReceiverThread  ZZzzZZZZ
ReceiverThread  Wake Up
ReceiverThread  bytesAvailable without reading 0
ReceiverThread  Received final: MSH|^~\&|10:00:E8:C1:1C:3C||YOU||20100101000313||ORU^W01^ORU_W01|AL0040|P|2.6|PID|
ReceiverThread  Launching HL7Manager Thread
ReceiverThread  HL7 parse susccessful
ReceiverThread  MSH|^~\&|YOU||10:00:E8:C1:1C:3C||201103241206||ACK^W01^ACK|1234|P|2.6|MSA|AA|AL0040|
ReceiverThread  Writting ACK AA
```

Figure 4.1 Log of the communication HDH-Android Server via Bluetooth

The operation of the application is correct and the communication has been done correctly.

In addition, if Android Server is working as the client in the connection, the state of the connection has to be controlled all the time. Therefore, if the connection fails because the devices are out of the Bluetooth range, the Android has to be reconnected to the HDH. This scenario has been tested and the application works properly. Following the explained behavior is shown.

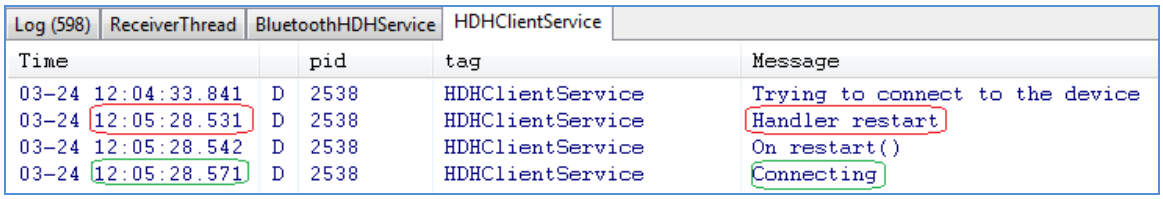


Time		pid	tag	Message
03-24 12:05:09.011	D	2538	ReceiverThread	HL7 parse susccessful
03-24 12:05:09.281	D	2538	ReceiverThread	MSH|^~\&|YOU||10:00:E8:C1:1C:3C||201103241205||ACK^W01^ACK|
03-24 12:05:09.291	D	2538	ReceiverThread	Writting ACK AA
03-24 12:05:09.302	D	2538	ReceiverThread	Launch Alert Thread
03-24 12:05:28.511	D	2538	ReceiverThread	EOF readed because channel is closed
03-24 12:05:28.551	D	2538	ReceiverThread	Receiver thread exiting.
03-24 12:06:07.581	D	2538	ReceiverThread	bytes readed 38
03-24 12:06:07.611	D	2538	ReceiverThread	Take it from the buffer: MSH|^~\&|10:00:E8:C1:1C:3C||YOU||2
03-24 12:06:07.621	D	2538	ReceiverThread	ZZzzZZZZ
03-24 12:06:08.626	D	2538	ReceiverThread	Wake Up
03-24 12:06:08.626	D	2538	ReceiverThread	bytesAvailable without reading 232
03-24 12:06:08.651	D	2538	ReceiverThread	bytes readed 596
03-24 12:06:08.671	D	2538	ReceiverThread	Take it from the buffer: 101000313||ORU^W01^ORU_W01|AL0040|
03-24 12:06:08.671	D	2538	ReceiverThread	ZZzzZZZZ
03-24 12:06:09.675	D	2538	ReceiverThread	Wake Up
03-24 12:06:09.675	D	2538	ReceiverThread	bytesAvailable without reading 0
03-24 12:06:09.701	D	2538	ReceiverThread	Received final: MSH|^~\&|10:00:E8:C1:1C:3C||YOU||2010010100
03-24 12:06:09.711	D	2538	ReceiverThread	Launching HL7Manager Thread
03-24 12:06:09.901	D	2538	ReceiverThread	HL7 parse susccessful
03-24 12:06:10.151	D	2538	ReceiverThread	MSH|^~\&|YOU||10:00:E8:C1:1C:3C||201103241206||ACK^W01^ACK|
03-24 12:06:10.161	D	2538	ReceiverThread	Writting ACK AA
03-24 12:06:10.171	D	2538	ReceiverThread	Launch Alert Thread

Figure 4.2 Android reconnection to the HDH

In the first image (Figure 4.2) it is illustrated that a communication was established and an ACK was written. Suddenly, the channel was closed because the distance between both devices (the distance between HDH and Android) was larger than the Bluetooth range.

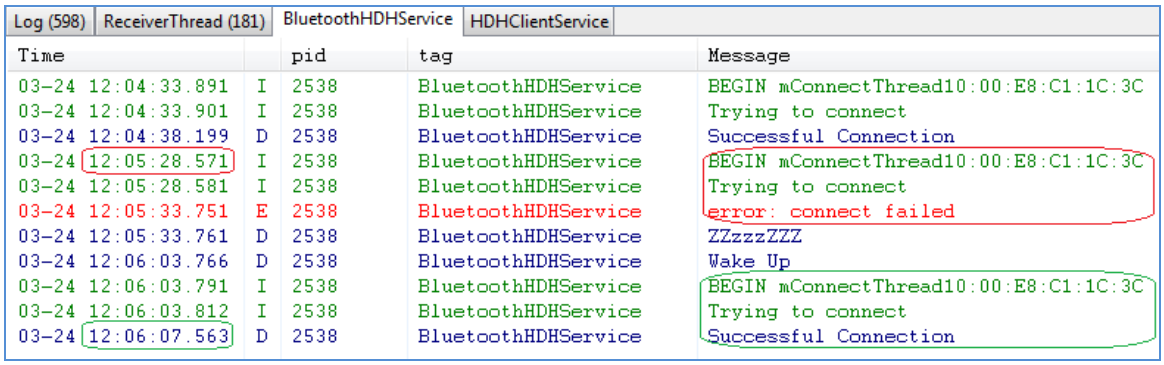
In the screen below (Figure 4.3), it is possible to appreciate that the Handler created in my system initiates again the connection automatically.



Time		pid	tag	Message
03-24 12:04:33.841	D	2538	HDHClientService	Trying to connect to the device
03-24 12:05:28.531	D	2538	HDHClientService	Handler restart
03-24 12:05:28.542	D	2538	HDHClientService	On restart()
03-24 12:05:28.571	D	2538	HDHClientService	Connecting

Figure 4.3 Handler responsible of the reconnection

So, the connection is restarted and the system tries to connect a second time. In that moment, the signal of Bluetooth is weak and the connection intent fails. Then, the system tries to restart the connection again with the HDH and because in this case the devices are closer, the connection is successful.



Time		pid	tag	Message
03-24 12:04:33.891	I	2538	BluetoothHDHService	BEGIN mConnectThread10:00:E8:C1:1C:3C
03-24 12:04:33.901	I	2538	BluetoothHDHService	Trying to connect
03-24 12:04:38.199	D	2538	BluetoothHDHService	Successful Connection
03-24 12:05:28.571	I	2538	BluetoothHDHService	BEGIN mConnectThread10:00:E8:C1:1C:3C
03-24 12:05:28.581	I	2538	BluetoothHDHService	Trying to connect
03-24 12:05:33.751	E	2538	BluetoothHDHService	error: connect failed
03-24 12:05:33.761	D	2538	BluetoothHDHService	ZZzzzZZZ
03-24 12:06:03.766	D	2538	BluetoothHDHService	Wake Up
03-24 12:06:03.791	I	2538	BluetoothHDHService	BEGIN mConnectThread10:00:E8:C1:1C:3C
03-24 12:06:03.812	I	2538	BluetoothHDHService	Trying to connect
03-24 12:06:07.563	D	2538	BluetoothHDHService	Successful Connection

Figure 4.4 Successful reconnection

Many of these attempts were tested and in most of the cases the reconnection was successful.

4.2.2 Android Server via WiFi

The connections done via WiFi technology were always in the same environment, i.e., always inside the private net of the university.

As explained in the chapter *3.7.2 HDH (client) simulator for TCP/IP connection*, a HDH simulator was used which is able to work with WiFi in order to test the communication between the HDH and Android Server via WiFi. (Currently on the HDH there is not a Wifi module, therefore a simulator was used).

The transmissions were tested using the simulator built in Java, obtaining a successful connectivity and message exchanging. Here are shown some **log**s which illustrate the connection between the HDH simulator and Android Server.

```
<terminated> QuickClient [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (29/03/2011 11:21:25)
Connecting to the WiFi Server. IP Address in University Network 10.201.92.92, port: 44444
Connected
Sending HL7 Message: MSH|^~\&|10:00:E8:C1:1C:3C||00:23:D4:69:C1:F1||20100510121500||ORU^W01^ORU_W01|AL0001|P|2.6|
PID|||1104289440||Niemann^Alexander||19410729|M|||&Molkereistrasse&1^^Wien^^1020|
OBR|1||0000^BLA|5^three-channelwaveformrecording^99SVL|||20100510120000|20100510121500|||||||||||||||F|
OBX|1|CD|5&CHN^^99SVL|1|1^ONE^0.5&mv^^100^0&255||||||F|
OBX|2|DTM|5&TIM^^99SVL|1|20100510121500||||||F|
OBX|3|NA|5&WAV^^99SVL|1|110^112^115^112^110^110^109^110^108^112^112^111^112^113^115||||||F|
OBX|4|NA|5&WAV^^99SVL|1|0007^0007^0007^0007^0007^0007^0007^0007^0007^0007^0007^0007^0007^0007^0007^0007||||||F|
OBX|5|NA|5&WAV^^99SVL|1|101^91^81^71^61^51^51^51^51^51^52^52^52^52^52||||||F|
Received: MSH|^~\&|00:23:D4:69:C1:F1||10:00:E8:C1:1C:3C||201103291116||ACK^W01^ACK|1234|P|2.6|
MSA|AA|AL0001|
```

Figure 4.5 Log of the communication HDH-AndroidServer via Wifi. Point of view: HDH simulator.

As noted, we have connected to the private address 10.201.92.92 and port 44444 (which is where the Android Server is hosted). Communication starts and in the trace is shown the sending of a HL7 message. The parsing and related ACK sending is also shown.

Log | ReceiverThread | BluetoothHDHService | HDHClientService | ConnectThreadtoMainServer | Receiver Thread in WifiServer

Time		pid	tag	Message
03-29 11:16:35.530	I	825	Receiver Thread in ...	MSH|^~\&|10:00:E8:C1:1C:3C||00:23:D4:69:C1:F1||20100510121500||ORU^W01^ORU_W01|AL0001|P|2.6|PID|||1
03-29 11:16:35.910	I	825	Receiver Thread in ...	Parse successful
03-29 11:16:36.110	I	825	Receiver Thread in ...	MSH|^~\&|00:23:D4:69:C1:F1||10:00:E8:C1:1C:3C||201103291116||ACK^W01^ACK|1234|P|2.6|MSA|AA|AL0001|
03-29 11:16:36.380	I	825	Receiver Thread in ...	Alert Thread launched

Figure 4.6 Log of the communication HDH-AndroidServer via Wifi. Point of view: AndroidServer.

In this case the message was AL type “alarm”. Due to this fact, the service of alert notification was launched.

Figure 4.7 Alarm message received

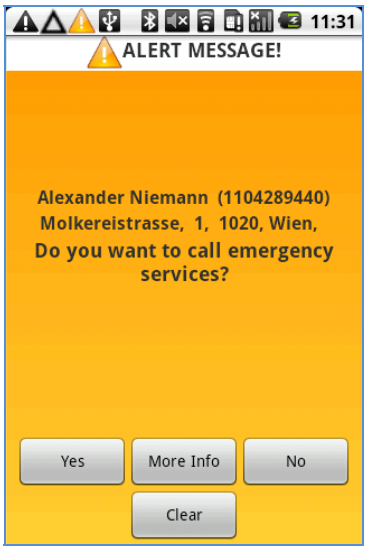


Figure 4.8 Alert notification slid

4.2.3 Connection with the HDH Main Server

The Figure 4.9 shows the IP and Port used by the Main Server [47]. In the form of the Figure 4.10 the user introduces the address to connect to it (IP 10.201.94.254 and Port 5555).

The

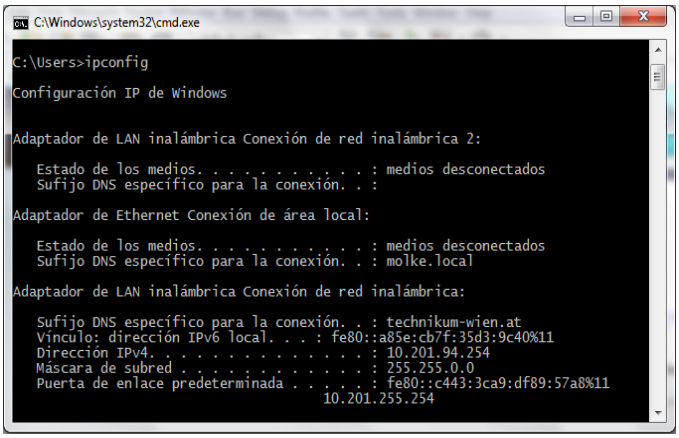


Figure 4.9 IP and Port of the Main Server

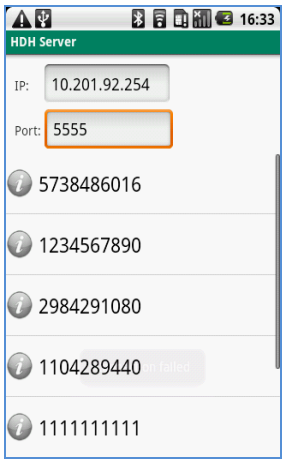


Figure 4.10 HDH Server

following figure illustrates the connection between the Android Server and the HDH Main Server. It is shown the **log** related with the upload of the patient files to the Main Server. In this example, after the successful connection, the files related with the messages AL0000 and AL0001 are transmitted.

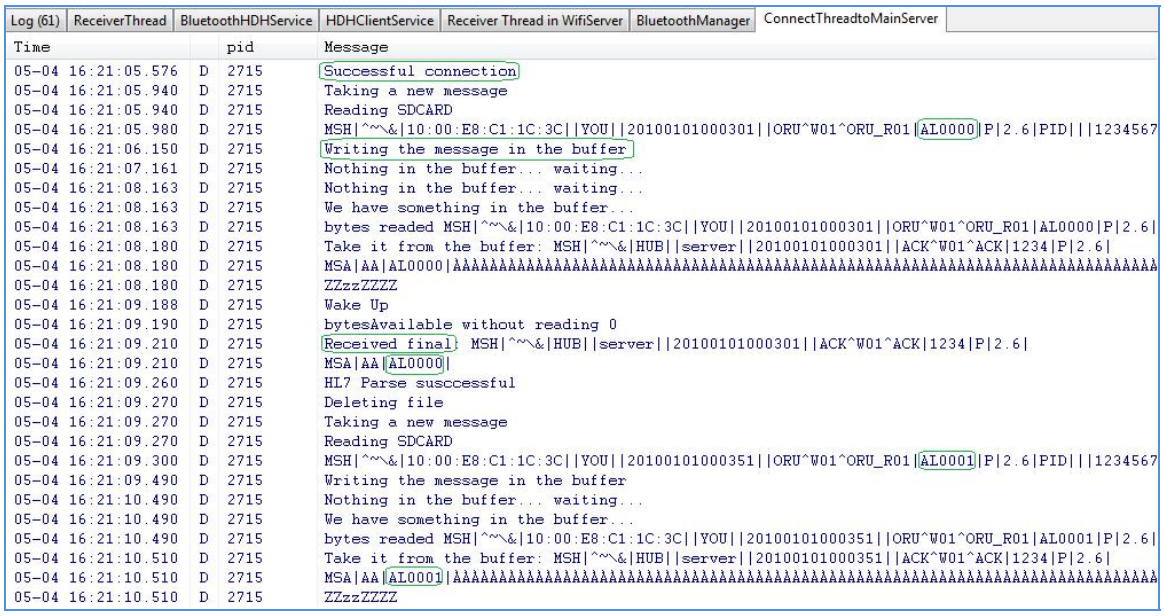


Figure 4.11 Uploading files from Android Server to the Main Server

In the figure above the sending message and the related ACK are underlined (green).

4.2.4 GSM Service

The GSM service has been tested using the tool available in the Android SDK [67]. This tool enables to simulate the connection to a GSM network and so, allows making calls and sending SMS to the Android emulator. In this way, what was done is to simulate that the HDH has integrated a GSM module which allows sending SMS to the Android Server.

Following are shown the results obtained related with the SMS messages.

This first tool enables sending messages. By integrating the Android SDK in the Eclipse environment, it is possible to handle it from there.

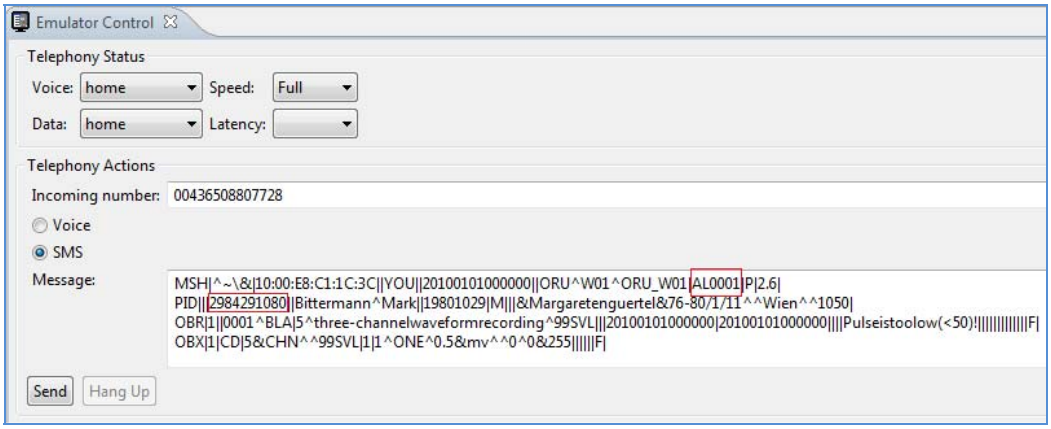


Figure 4.12 Emulator Control, SMS sending

Beneath it is illustrated how the AndroidServerReceiver detects the reception of an SMS and transmits it to the service which will interpret it.

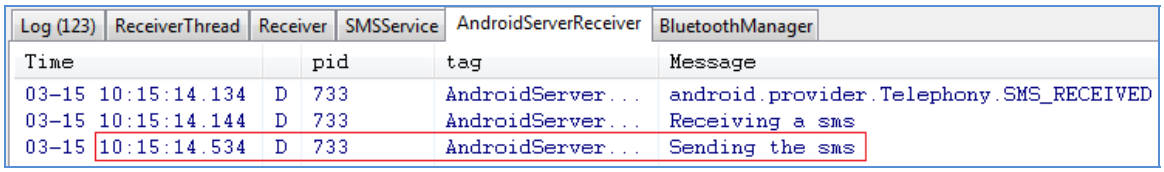


Figure 4.13 AndroidServerReceiver catches an incoming SMS and sends to the SMSService

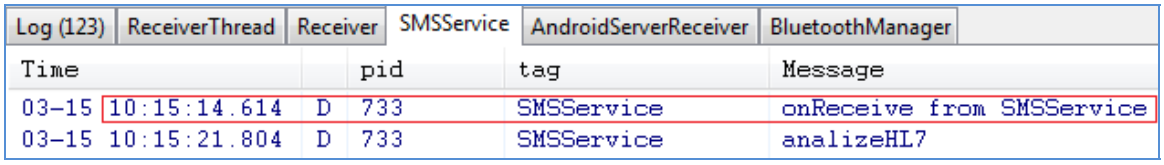


Figure 4.14 SMSService receives the SMS and analyzes it.

Here it is displayed the SMS message in the Android emulator and alarm notification that it has generated.

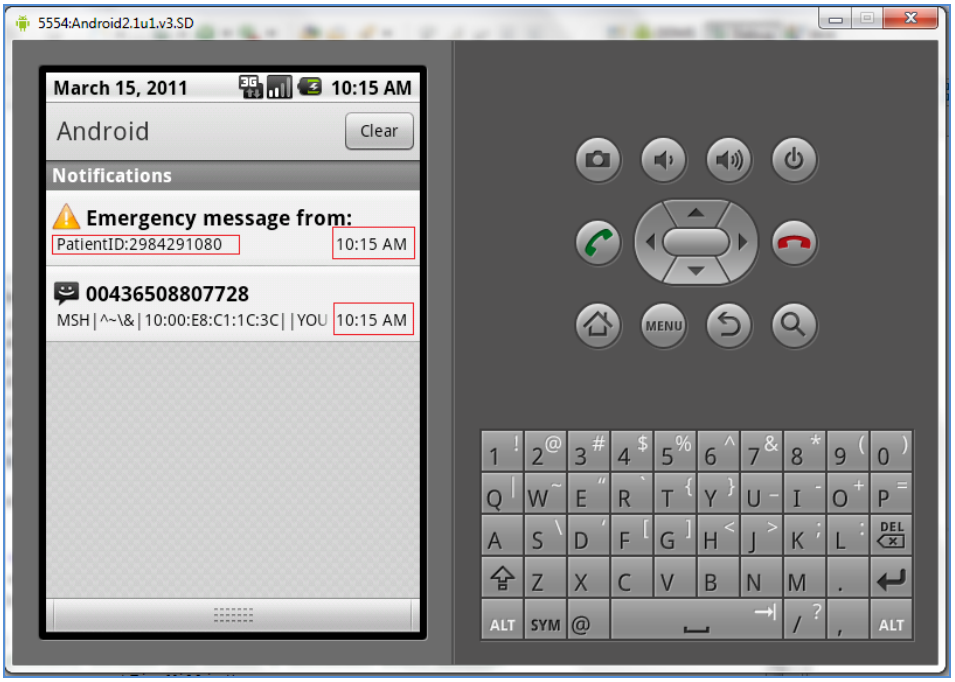


Figure 4.15 Android emulator showing the received SMS

4.2.5 Testing the Graphical User Interface

Regarding to the graphical interface and the programming related with it, some tests were done and no errors were found in the debug. That is why it was considered enough stable in its execution.

Since the mobile phone market is always in constant motion, there are a lot of mobiles which implement Android, all of them with different features, screens... the great thing would be to be able to make that the graphical interface adapts in every kind of mobile [69]. In this way, it would be perfectly visible in all the devices. For the moment, a design was developed which works with the current phone, HTC Magic. If it is expected that all interfaces need to display correctly further tests need to be conducted.

As explained in the reference of developers of Android [69], this issue should be noted for applications that have been developed for platforms older than 1.5, because for later versions the platforms provide services for adapting to the screen. In the case of wanting to show this application in versions with extra large screens, several tests should perform.

5 Conclusions and Future researches

5.1 Conclusions

The completion of the Master Thesis in an e-Health environment has been a very enriching experience. Nowadays, the e-health and medical interoperability is an area that is expanding thanks to several applications developed (and currently developing) by research groups in collaboration with hospitals, service providers and manufacturers of medical devices. Increasingly, companies start investing on the interoperability of their systems because it represents a market that will undoubtedly be claimed by the public.

During this exchange experience with the FHTW, this Android Server has been developed to improve the homecare of patients with illness. Therefore, the patients can be supervised and monitored all the time avoiding going to the hospital every time they have to make a measurement, thanks to it is real time monitoring. Besides, family members, who use this Android Server application and they are not in the same place of the patient, can surely know any time that their relatives are continuously monitored.

In order to face the development of the Android Server, it was necessary to understand how the HL7 standard works and learn about some program languages as Java and Android. The knowledge of this tools is an improvement for my studies because is very useful for my professional career.

On the one hand, dealing with HL7 standard provides, without doubt, a very interesting knowledge. Without the existence of standards, the possibility of transmitting, receiving and interpreting medical messages would be impossible

On the other hand, dealing with Android, which is an open source platform, means that it has a great potential since it enables to integrate completely in other systems. After all, in open source the developers can read, redistribute and modify the source code of an application that is why it grows and progresses by leaps and bounds. The community improves the software, adapts and corrects it so fast that the system becomes a colossal tool. Every day there are more tools of this type available and there is a marked tendency for users at all levels (business, education ...) to migrate much of its applications to it.

In this context, the developed Android Server is capable to understand and interchange HL7 messages with the HDH wrist device and the Main Server by different technologies and it acts in a timely manner. Moreover, the Android Server is capable to deal with patient´s information and personal health data showing the related graphs showing trends of received values.

Because of these reasons, this application represents the first step for solutions in the homecare field related with the Health Data Hub.

5.2 Future researches

This Master Thesis has as a result the first version of the Android Server software. This software enables to exchange medical messages by the standard HL7 and provides a powerful tool for the telemonitoring of patients.

However, as mentioned, the developed software is a first version and some issues could be added in the future:

- Implementation of a tool for the standard HL7 messages in the Android platform. The package developed for the HL7 type messages, is a tool which currently works properly in the present environment. However, it would be appropriate to make a code similar to HAPI [13], which would enable to recognizing any HL7 message. This functionality would provide an independent autonomy regarding to the messages sent from the devices.

- Implementation of a SQLite data base in Android [70]for the security of the data. It would be able to replace the current file system available in the application. SQLite is a relational data base management. Android provides a SQLite offers everything needed for local storage and fast and simply result.

- Implementation of Web services or SSL security in the non-secure connections, as the connection by Internet. This work should be done in cooperation with the developers of the Main Server. This fact enables sending medical type messages, protected from possible intrusions in the connection.

These are possible future research topics which could be implemented for a second version of the Android Server.

In addition, the emerging technologies within eHealth environment have to be kept in mind. The technical features and communication requirements of these technologies are defined depending on the domain where they work. In the following paragraphs there are explained some of these technologies.

BLTE is an alternative to the Bluetooth standard that was introduced in Bluetooth v4.0, and is aimed at very low power applications running off a coin cell. It allows two types of implementation, dual-mode and single-mode. In a dual-mode implementation, Bluetooth low energy functionality is integrated into an existing Classic Bluetooth controller. The

resulting architecture shares much of Classic Bluetooth's existing radio and functionality resulting in a negligible cost increase compared to Classic Bluetooth. Additionally, manufacturers can use current Classic Bluetooth (Bluetooth v2.1 + EDR or Bluetooth v3.0 + HS) chips with the new low energy stack, enhancing the development of *Classic Bluetooth* enabled devices with new capabilities.

Cost-reduced single-mode chips, which will enable highly integrated and compact devices, will feature a lightweight Link Layer providing ultra-low power idle mode operation, simple device discovery, and reliable point-to-multipoint data transfer with advanced power-save and secure encrypted connections at the lowest possible cost. The Link Layer in these controllers will enable Internet connected sensors to schedule Bluetooth low energy traffic between Bluetooth transmissions [71].

ZigBee technology, operates on the IEEE 802.15.4 standard, as this only defines the physical layer and Medium Access Control (MAC). It is targeted at radio-frequency (RF) applications that require a low data rate, long battery life, and secure networking. The hardware is considerably simpler than BT, achieving several years of battery life through minimal power consumption [72].

NFC (Near Field Communication) technology [73]. It can be also very interesting in the eHealth environment, especially in cases that it is not necessary to monitoring the patient all the time (the short range of the NFC limits the capacity to continuous monitoring). In addition, Android has developed the API for the new mobile devices which integrates this chip [74] . More information is available in the *Annex IX. NFC.*

Moreover, related with the research environment, an integration of the **ANT+** (*Annex X. ANT+)* technology in Android would be useful, because this technology is being increasingly used in the Android development. There are some devices which already implement it [75]. ANT+ is primarily designed for collection and transfer of sensor data, to manageable units of various types. The three main areas of operation are sport, wellness and home health. It can be used for data-transfer for a number of devices enabling to get the Android platform closer to the e-Health context.

Within the LAN technologies domain, **Wi-Fi Direct** has to be highlighted. It consists of a series of software protocols (Wi-Fi Peer-to-Peer specification) that allow Wi-Fi devices to communicate with each other directly without needing a wireless Access Point (AP) [76]. In terms of security, it uses the Wi-Fi Protected Setup to create connections (using WPA2) between devices. The work at Continua Health Alliance considers emerging technologies to have particular relevance in interoperability. In February 2011, Continua signed a collaboration agreement with Wi-Fi Alliance for the promotion and adoption of Wi-Fi Direct technology [77].

Bibliography

[1] Standard ISO/IEEE 11073. [Online] http://www.11073.org/. [04/11]

[2] Standard HL7 v2.6. *Health Level Seven International.* [Online] http://www.hl7.org/. [04/11]

[3] Android. [Online] http://www.android.com/. [05/11]

[4] Open Handset Alliance. [Online] http://www.openhandsetalliance.com/. [05/11]

[5] **Savov, Vlad.** engadget. [Online] 01 2011. [03/11]http://www.engadget.com/2011/01/31/canalys-android-overtakes-symbian-as-worlds-best-selling-smart/. [03/11]

[6] Eclipse. [Online] http://www.eclipse.org/. [03/11]

[7] Healthy Interoperability. [Online] http://www.healthy-interoperability.at/. [04/11]

[8] **Gerbovics, F., et al.** Implementation of a wrist wearable Health Data Hub-prototype for context based telemonitoring of elderly people using standardized data transfer methods. Vienna: University of Applied Sciences Technikum Wien, 2010. [04/11]

[9] Continua Health Alliance. [Online] http://www.continuaalliance.org/. [04/11]

[10] ANT TM. [Online] http://www.thisisant.com/. [04/11]

[11] European Committee for Standardization. [Online] http://www.cen.eu/. [04/11]

[12] International Organization for Standardization. [Online] http://www.iso.org. [04/11]

[13] HAPI. [Online] http://hl7api.sourceforge.net/. [02/11]

[14] **Bitterman, Mark.** Development of a Data Model for a wrist wearable Health Device Hub . Vienna : University of Applied Sciences Technikum Wien, 2010. [04/11]

[15] Stat Counter. [Online] http://gs.statcounter.com. [03/11]

[16] Nokia. [Online] http://www.nokia.com/. [03/11]

[17] Meego. [Online] http://conversations.nokia.com/2010/02/15/nokia-and-intel-create-meego-for-new-era-of-mobile-computing/. [03/11]

[18] Blackberry. [Online] http://us.blackberry.com/. [03/11]

[19] Windows Phone. [Online] http://www.microsoft.com/windowsphone/es-es/default.aspx. [03/11]

[20] Apple. [Online] http://www.apple.com/ios/. [03/11]

[21] **Ableson, Frank, Collins, Charlie y Sen, Robi.** *Unlocking Android a developer´s guide.* United States of America : Manning Publications Co., 2009. [05/11]

[22] JetBrains. *IntelliJ IDEA.* [Online] http://www.jetbrains.com/idea/. [03/11]

[23] NetBeans. [Online] http://netbeans.org/. [03/11]

[24] Using DDMS, Dalvik Debug Monitor Server. *Android Developers.* [Online] http://developer.android.com/guide/developing/debugging/ddms.html. [03/11]

[25] **Rogers, Rick, et al.** *Android Application Development.* United States of America : O´Reilly Media, 2009. [04/11]

[26] Installing the ADT Plugin. *Android Developers.* [Online] http://developer.android.com/sdk/eclipse-adt.html#installing. [03/11]

[27] Installing the SDK. *Android Developers.* [Online] http://developer.android.com/sdk/installing.html. [03/11]

[28] Download the Android SDK. *Android Developers.* [Online] http://developer.android.com/sdk/index.html. [03/11]

[29] **Tellechea Pereda, Amagoia.** Development and Implementation of a Data Model for an Adnroid based Telemonitoring Server. Vienna : University of Applied Sciences Technikum Wien, 2010. [05/11]

[30] Service. *Android Developers.* [Online] http://developer.android.com/reference/android/app/Service.html. [04/11]

[31] Activity. *Android Developers.* [Online] http://developer.android.com/reference/android/app/Activity.html. [04/11]

[32] Thread. *Android Developers.* [Online] http://developer.android.com/reference/java/lang/Thread.html. [03/11]

[33] Notifications. *Android Developers.* [Online] http://developer.android.com/guide/topics/ui/notifiers/notifications.html. [03/11]

[34] Bluetooth. *Android Developers.* [Online] http://developer.android.com/guide/topics/wireless/bluetooth.html. [03/11]

[35] Context. *Android Developers.* [Online] http://developer.android.com/reference/android/content/Context.html. [03/11]

[36] Handler. *Android Developers.* [Online] http://developer.android.com/reference/android/os/Handler.html. [04/11]

[37] Bluetooth Server Socket. *Android Developers.* [Online] http://developer.android.com/reference/android/bluetooth/BluetoothServerSocket.html. [04/11]

[38] Bluetooth Socket. *Android Developers.* [Online] http://developer.android.com/reference/android/bluetooth/BluetoothSocket.html. [04/11]

[39] InputStream. *Android Developers.* [Online] http://developer.android.com/reference/java/io/InputStream.html. [03/11]

[40] Simple XML Serialization. [Online] http://simple.sourceforge.net/. [03/11]

[41] UUID, Universally Unique IDentifier. *Android Developers.* [Online] http://developer.android.com/reference/java/util/UUID.html. [04/11]

[42] UUID for SPP connection (Bluetooth Device). *Android Developers.* [Online] http://developer.android.com/reference/android/bluetooth/BluetoothDevice.html. [04/11]

[43] ServerSocket. *Android Developers.* [Online] http://developer.android.com/reference/java/net/ServerSocket.html. [03/11]

[44] Toast. *Android Developers.* [Online] http://developer.android.com/reference/android/widget/Toast.html. [02/11]

[45] The AndroidManifest.xml File. *Android Developers.* [Online] http://developer.android.com/guide/topics/manifest/manifest-intro.html. [04/11]

[46] BroadcastReceiver. *Android Developers.* [Online] http://developer.android.com/reference/android/content/BroadcastReceiver.html. [04/11]

[47] **Khumrin, Piyapong.** Design and Implementation of Electronic Health Records for the Health Device Hub of the Healthy Interoperability Framework within an HDH project. Vienna : University of Applied Sciences Technikum Wien, 2010. [04/11]

[48] **Jeffrey, E.F. Friedl.** *Mastering regular expressions.* United States of America : O´Reilly Media, Inc., 2006. [03/11]

[49] Using HAPI. *SourceForge.* [Online] http://hl7api.sourceforge.net/using_hapi.html. [02/11]

[50] ListActivity. *Android Developers.* [Online] http://developer.android.com/reference/android/app/ListActivity.html. [03/11]

[51] TabActivity. *Android Developers.* [Online] http://developer.android.com/reference/android/app/TabActivity.html. [03/11]

[52] Intent. *Android Developers.* [Online]
http://developer.android.com/reference/android/content/Intent.html. [03/11]

[53] RelativeLayout. *Android Developers.* [Online]
http://developer.android.com/reference/android/widget/RelativeLayout.html. [03/11]

[54] TableLayout. *Android Developers.* [Online]
http://developer.android.com/reference/android/widget/TableLayout.html. [03/11]

[55] Graphics. *Android Developers.* [Online]
http://developer.android.com/guide/topics/graphics/index.html. [03/11]

[56] aChartEngine. *4ViewSoft.* [Online] http://achartengine.org/. [03/11]

[57] HTC Magic. [Online] http://www.htc.com/www/product/magic/overview.html.
[04/11]

[58] LinearLayout. *Android Developers.* [Online]
http://developer.android.com/reference/android/widget/LinearLayout.html. [03/11]

[59] Creating Dialogs. *Android Developers.* [Online]
http://www.htc.com/www/product/magic/overview.html. [03/11]

[60] LayoutInflater. *Android Developers.* [Online]
http://developer.android.com/reference/android/view/LayoutInflater.html. [03/11]

[61] Menu. *Android Developers.* [Online]
http://developer.android.com/reference/android/view/Menu.html. [03/11]

[62] TabHost. *Android Developers.* [Online]
http://developer.android.com/reference/android/widget/TabHost.html. [03/11]

[63] TabWidget. *Android Developers.* [Online]
http://developer.android.com/reference/android/widget/TabWidget.html. [03/11]

[64] FrameLayout. *Android Developers.* [Online]
http://developer.android.com/reference/android/widget/FrameLayout.html. [03/11]

[65] TabSpec. *Android Developers.* [Online]
http://developer.android.com/reference/android/widget/TabHost.TabSpec.html.
[03/11]

[66] ScrollView. *Android Developers.* [Online]
http://developer.android.com/reference/android/widget/ScrollView.html. [03/11]

[67] Emulator of Android SDK. *Android Developers.* [Online]
http://developer.android.com/guide/developing/tools/emulator.html. [03/11]

[68] **Laycock, Gilbert Thomas.** *The theory and practice of specification based software testing.* United Kingdom : Dept of Computer Science, Sheffield University, 1993. [04/11]

[69] Supporting Multiple Screens. *Android Developers.* [Online] http://developer.android.com/guide/practices/screens_support.html. [04/11]

[70] SQLiteDataBase. *Android Developers.* [Online] http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html. [05/11]

[71] BTLE Technology. *World News Bluetooth Wikipedia.* [Online] http://wn.com/Bluetooth. [05/11]

[72] ZigBee technology. *Computer Science and Communications Research Unit.* [Online] http://wiki.uni.lu/secan-lab/ZigBee+technology+in+sensor+network.html. [05/11]

[73] NFC Forum. [Online] http://www.nfc-forum.org/. [04/11]

[74] Near Field Communication API. *Android Developers.* [Online] http://developer.android.com/guide/topics/nfc/index.html. [04/11]

[75] ANT Android API. *ANT TM.* [Online] http://www.thisisant.com/pages/developer-zone/android-api. [04/11]

[76] Wi-Fi Direct. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Wi-Fi_Direct. [05/11]

[77] **News, EMR and EHR.** Wi-Fi Alliance and Continua Health Alliance collaborate on Personal connected health. *EMR and Health IT News.* [Online] February 2011. http://www.emrandhipaa.com/news/2011/02/22/wi-fi-alliance-and-continua-health-alliance-collaborate-on-personal-connected-health/. [05/11]

[78] Java Specification Request, JSR. *Java ME & Optional Packages.* [Online] http://mobilegametutorials.blogspot.com/2010/03/java-me-optional-packages.html. [05/11]

[79] Feature Highlights. *ANT TM.* [Online] http://www.thisisant.com/technology/feature-highlights. [04/11]

[80] News. *ANT TM.* [Online] http://www.thisisant.com/news/stories/ant-enabled-live-google-data-feed. [04/11]

[81] Tortoise. *Subversion.* [Online] http://tortoisesvn.tigris.org/. [03/11]

List of Figures

CHAPTER 1

CHAPTER 2

CHAPTER 3

CHAPTER 4

List of Abbreviations

ADT: Android Developer Tools
ANSI: American National Standard Institute
AOSP: Android Open Source Project
API: Application Program Interfaces
CEN: European Committee for Standardization
DDMS: Dalvik Debug Monitor Server
DVM: Dalvik Virtual Machine
FHTW: Fachhochschule Technikum Wien
GSM: Global System for Mobile Communications
HDF: Hierarchical Data Format
HDH: Health Data Hub
HIO: Healthy Interoperability
HL7: Health Level Seven
ICT: Information and Communications Technology
ICU: Intensive Care Unit
IDE: Integrated Development Environment
IEEE: Institute of Electrical and Electronics Engineering
IP: Internet Protocol
ISO: International Standards Organization
Java ME: Java Micro Edition
JDT: Java Development Tools
JVM: Java Virtual Machine
LAN: Local Area Network
L2CAP: Logical Link Control and Adaptation Protocol
NFC: Near Field Communication
SPP: Serial Port Profile
OHA: Open Handset Alliance
OSI: Open Systems Interconnection
PAN: Personal Area Network
PDA: Personal Digital Assistant
PDT: PHP Development Tools
PHD: Personal Health Devices
POC: Point Of Care
RFCOM: Radio Frequency Communication
RFID: Radio Frequency IDentification
SDK: Software Development Kit.
SDP: Session Description Protocol
SMS: Short Message Service
SRAM: Static Random Access Memory

SVN: Subversion
UAS: University of Applied Sciences
UUID: Universally Unique Identifier
WPAN: Wireless Personal Area Networks
XML: Extensible Markup Language

Annexes

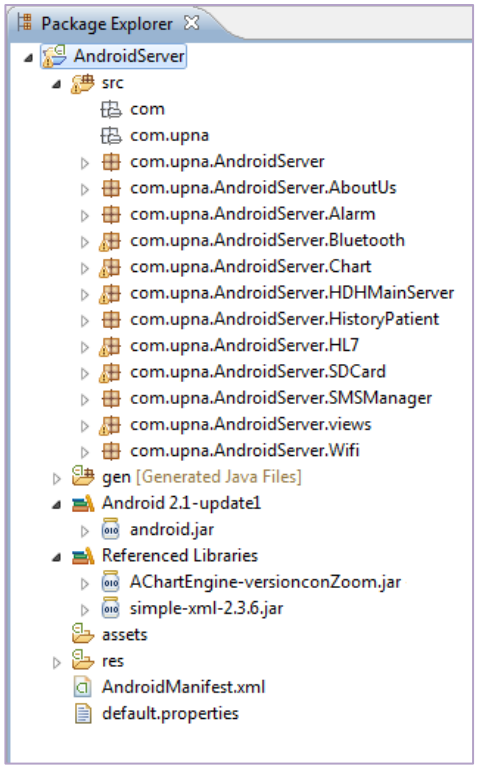
I. External storage system: SDCard

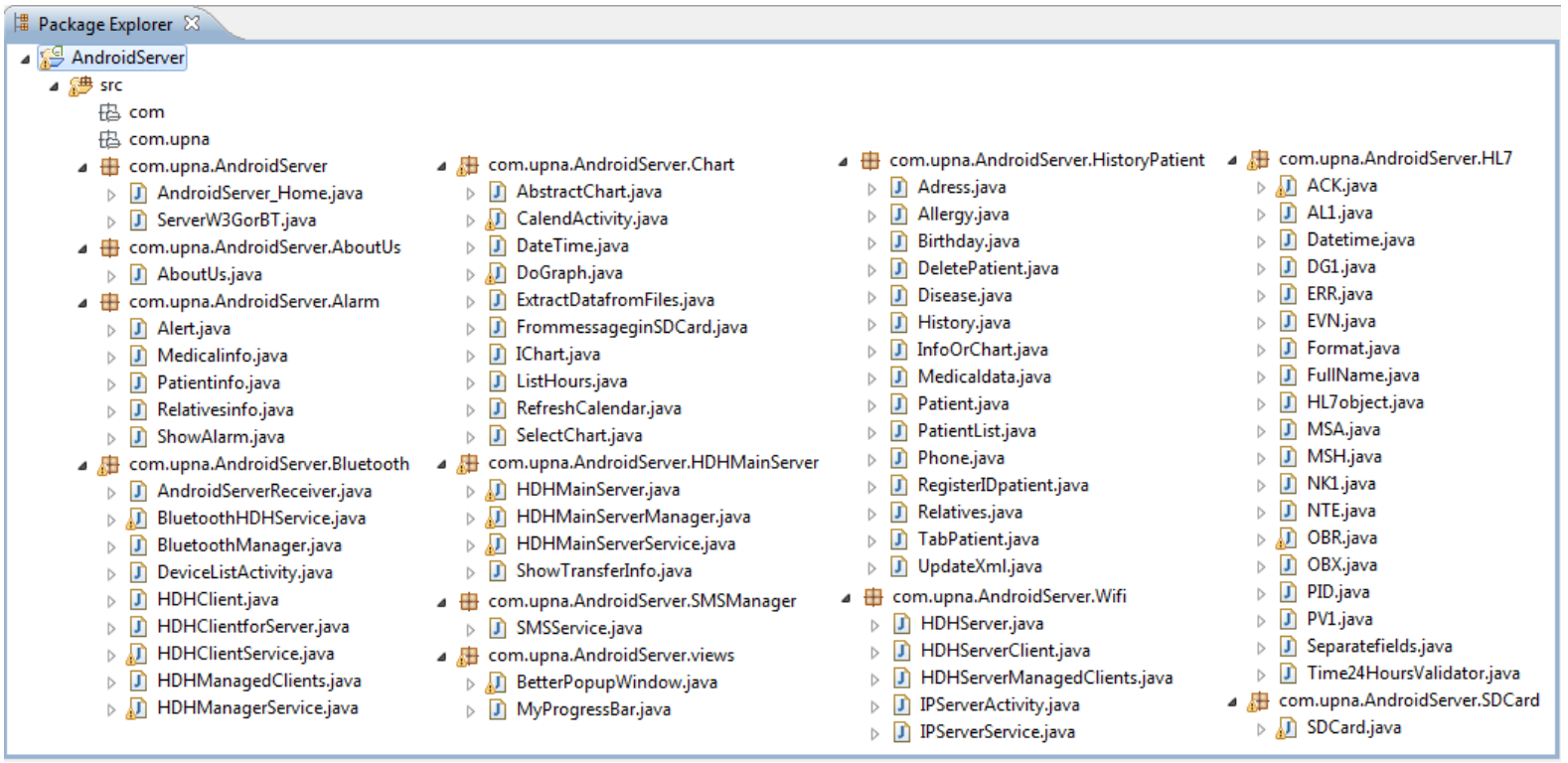
The class **SDCard** in the **com.upna.AndroidServer.SDCard** package is responsible of doing all the I/O operations in the mobile phone external storage system, for the Android Server application.

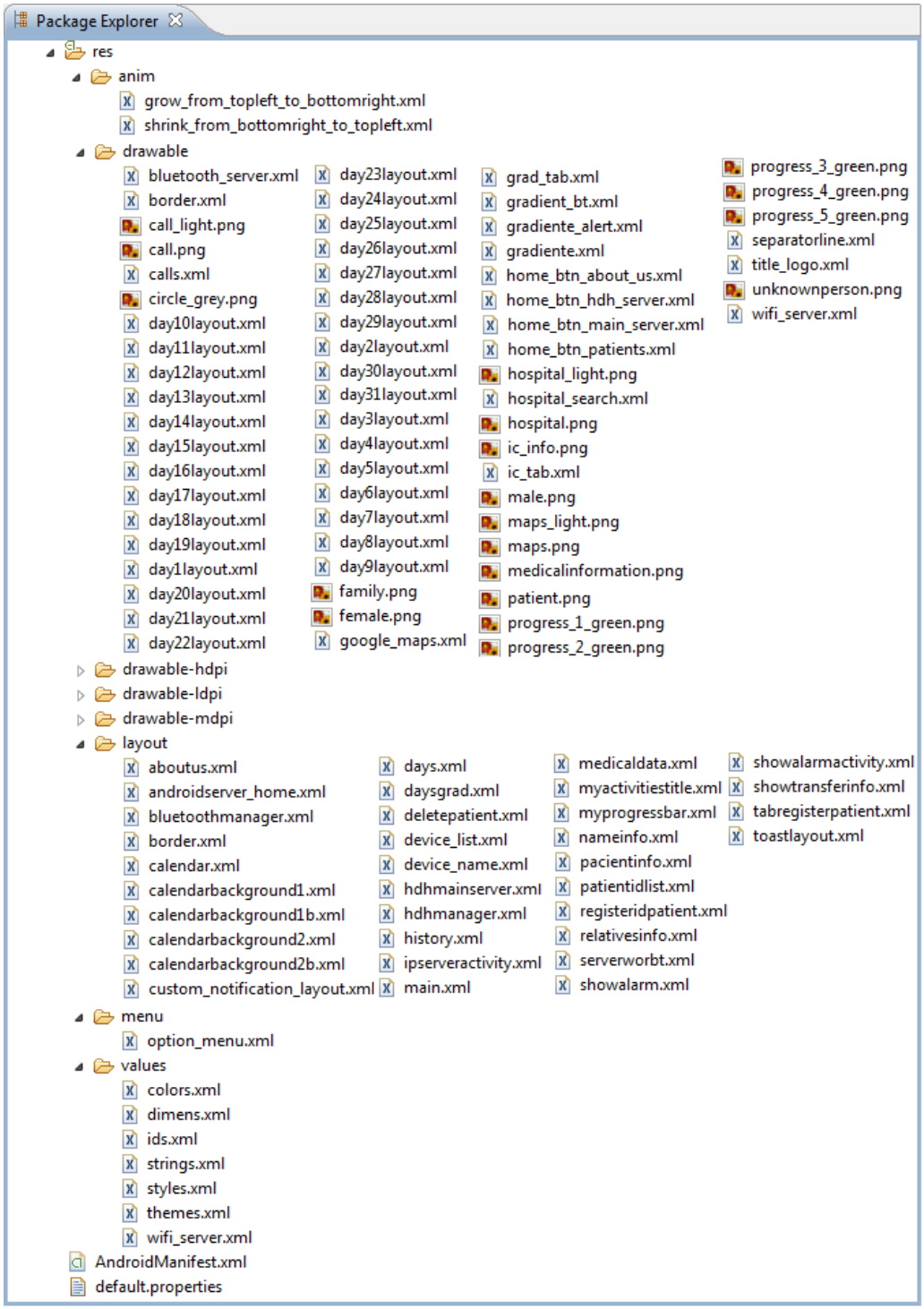
The **most used** methods are shown in the list below:

Method	Description
hasStorage ()	Returns a Boolean which indicates if the SDCard target is mounted on the phone.
existFile(File fileCheck)	Returns a Boolean which indicates if the file exists and it is in readable mode.
existPatient()	Returns a Boolean which indicates if the patient exists and it is in readable mode.
listDirPatient()	Returns a String[] which list all the existing patients.
createDirofPatient(String idPatient)	Returns Boolean if the folders for the current patient (idPatient) are created. If there is no patient in the storage system it creates the patient file.
removeDirofPatient(String idPatient)	Returns a Boolean indicating if the patient structure has been deleted from the system. Deletes all the internal files.
deleteDirectory(File path)	Returns a Boolean indicating if the files have been deleted (in recursive way) in the path folder.
writeMessage(String message, HL7object hl7)	Stores the message in the SDCard, using the Identification Number of the HL7. See figure 3.50. Depending on the HL7 message type, it is stored in the specify folder (STAL or PI).
dirListByDescendingDate(File folder)	Returns File[]listing the files by descending date.
dirListByAscendingDate(File folder)	Returns File[] listing the files by ascending date.
dirListByDescendingName(File folder)	Returns File[] listing the files by descending name.
dirListByAscendingName(File folder)	Returns File[] listing the files by ascending name.
readFile(String filename)	Returns a String representing the content of the read file (filename).

II. AndroidServer Project Directory







III. Android Manifest

The manifest presents essential information about the application to the Android system, information the system must have before it can run any of the application's code. Every application must have an AndroidManifest.xml file in its root directory.

This one is the AndroidManifest of my Android Server application:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
        xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.upna.AndroidServer"
      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.RECEIVE_SMS" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.CALL_PHONE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>


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

<!-- AndroidServer -->
<activity android:name=".AndroidServer_Home" android:label="@string/app_name"
android:theme="@android:style/Theme.Light.NoTitleBar">
       <intent-filter>
             <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
       </intent-filter>
</activity>

<activity android:name="com.upna.AndroidServer.ServerW3GorBT"
android:theme="@android:style/Theme.Light.NoTitleBar" >
</activity>

<!-- About Us -->
<activity android:name="com.upna.AndroidServer.AboutUs.AboutUs"
android:theme="@style/customTheme">
</activity>

<!-- Alarm -->
<activity android:name="com.upna.AndroidServer.Alarm.Patientinfo"/>
```

```
<activity android:name="com.upna.AndroidServer.Alarm.Medicalinfo"/>
<activity android:name="com.upna.AndroidServer.Alarm.Relativesinfo"/>
<activity android:name="com.upna.AndroidServer.Alarm.ShowAlarm"
android:theme="@android:style/Theme.Light.NoTitleBar">
</activity>

<!-- Bluetooth -->
<receiver android:name="com.upna.AndroidServer.Bluetooth.AndroidServerReceiver"
android:enabled="true">
      <intent-filter>
            <action android:name="android.provider.Telephony.SMS_RECEIVED" />
            <action android:name="android.content.Intent.ACTION_BOOT_COMPLETED"/>
           <action android:name="android.intent.action.BOOT_COMPLETED"/>
            <category android:name="android.intent.category.HOME"/>
      </intent-filter>
</receiver>
<activity android:name="com.upna.AndroidServer.Bluetooth.BluetoothManager"
android:theme="@android:style/Theme.Light.NoTitleBar">
</activity>
<activity android:name="com.upna.AndroidServer.Bluetooth.DeviceListActivity"
android:label="@string/select_device"
android:theme="@android:style/Theme.Dialog"
android:configChanges="orientation|keyboardHidden">
</activity>
<service android:name="com.upna.AndroidServer.Bluetooth.HDHClientService" />
<service android:name="com.upna.AndroidServer.Bluetooth.HDHManagerService" />

<!-- Chart -->
<activity android:name="com.upna.AndroidServer.Chart.CalendActivity"
android:theme="@style/customTheme">
</activity>
<activity android:name="com.upna.AndroidServer.Chart.ListHours"
android:theme="@style/customTheme">
</activity>
<activity android:name="com.upna.AndroidServer.Chart.SelectChart"
android:theme="@style/customTheme">
</activity>
<activity android:name="org.achartengine.GraphicalActivity"
android:theme="@android:style/Theme.NoTitleBar">
</activity>

<!-- HDHMainServer -->
<activity android:name="com.upna.AndroidServer.HDHMainServer.HDHMainServer"
android:theme="@style/customTheme">
</activity>
<service android:name="com.upna.AndroidServer.HDHMainServer.HDHMainServerService"/>
<activity android:name="com.upna.AndroidServer.HDHMainServer.ShowTransferInfo"
android:theme="@style/customTheme">
</activity>

<!-- HistoryPatient -->
<activity android:name="com.upna.AndroidServer.HistoryPatient.DeletePatient"
android:theme="@style/customTheme">
```

```
</activity>
<activity android:name="com.upna.AndroidServer.HistoryPatient.History"
android:theme="@style/customTheme">
</activity>
<activity android:name="com.upna.AndroidServer.HistoryPatient.InfoOrChart"
android:theme="@style/customTheme">
</activity>
<activity android:name="com.upna.AndroidServer.HistoryPatient.PatientList"
android:theme="@style/customTheme">
</activity>
<activity android:name="com.upna.AndroidServer.HistoryPatient.RegisterIDpatient"
android:windowSoftInputMode="stateUnchanged"
android:theme="@android:style/Theme.Light.NoTitleBar">
</activity>
<activity android:name="com.upna.AndroidServer.HistoryPatient.TabPatient"
android:theme="@android:style/Theme.Light.NoTitleBar">
</activity>

<!-- Wifi -->
<activity android:name="com.upna.AndroidServer.Wifi.IPServerActivity"
android:theme="@android:style/Theme.Light.NoTitleBar">
</activity>
<service android:name="com.upna.AndroidServer.Wifi.IPServerService"/>

<!-- SMS -->
<service android:name="com.upna.AndroidServer.SMSManager.SMSService" />

</application>
</manifest>
```

IV. Table of the HL7 messages components

MSH			
MSH-1: Field Separator (ST)	**MSH-8: Security (ST)**	MSH-15: Accept Acknowledgment Type (ID)	MSH-22: Sending Responsible Organization (XON)
MSH-2: Encoding Characters (ST)	**MSH-9: Message Type (MSG)**	MSH-16: Application Acknowledgment Type (ID)	MSH-23: Receiving Responsible Organization (XON)
MSH-3: Sending Application (HD)	**MSH-10: Message Control ID (ST)**	MSH-17: Country Code (ID)	MSH-24: Sending Network Address (HD)
MSH-4: Sending Facility (HD)	**MSH-11: Processing ID (PT)**	MSH-18: Character Set (ID)	MSH-25: Receiving Network Address (HD)
MSH-5: Receiving Application (HD)	**MSH-12: Version ID (VID)**	MSH-19: Principal Language Of Message (CWE)	
MSH-6: Receiving Facility (HD)	MSH-13: Sequence Number (NM)	MSH-20: Alternate Character Set Handling Scheme (ID)	
MSH-7: Date/Time of Message (DTM)	MSH-14: Continuation Pointer (ST)	MSH-21: Message Profile Identifier (EI)	

PID			
PID-1: Set ID - PID (SI)	**PID-11: Patient Address (XAD)**	PID-21: Mother's Identifier (CX)	PID-31: Identity Unknown Indicator (ID)
PID-2: Patient ID (CX)	PID-12: County Code (IS)	PID-22: Ethnic Group (CWE)	PID-32: Identity Reliability Code (IS)
PID-3: Patient Identifier List (CX)	PID-13: Phone Number - Home (XTN)	PID-23: Birth Place (ST)	PID-33: Last Update Date/Time (DTM)
PID-4: Alternate Patient ID - PID (CX)	PID-14: Phone Number - Business (XTN)	PID-24: Multiple Birth Indicator (ID)	PID-34: Last Update Facility (HD)
PID-5: Patient Name (XPN)	PID-15: Primary Language (CWE)	PID-25: Birth Order (NM)	PID-35: Species Code (CWE)
PID-6: Mother's Maiden Name (XPN)	PID-16: Marital Status (CWE)	PID-26: Citizenship (CWE)	PID-36: Breed Code (CWE)
PID-7: Date/Time of Birth (DTM)	PID-17: Religion (CWE)	PID-27: Veterans Military Status (CWE)	PID-37: Strain (ST)
PID-8: Administrative Sex (IS)	PID-18: Patient Account Number (CX)	PID-28: Nationality (CWE)	PID-38: Production Class Code (CWE)
PID-9: Patient Alias (XPN)	PID-19: SSN Number - Patient (ST)	PID-29: Patient Death Date and Time (DTM)	PID-39: Tribal Citizenship (CWE)
PID-10: Race (CWE)	PID-20: Driver's License Number - Patient (DLN)	PID-30: Patient Death Indicator (ID)	

OBR			
OBR-1: Set ID - OBR (SI)	OBR-14: Specimen Received Date/Time (DTM)	OBR-27: Quantity/Timing (TQ)	OBR-40: Transport Arrangement Responsibility (CWE)
OBR-2: Placer Order Number (EI)	OBR-15: Specimen Source (SPS)	OBR-28: Result Copies To (XCN)	OBR-41: Transport Arranged (ID)
OBR-3: Filler Order Number (EI)	OBR-16: Ordering Provider (XCN)	OBR-29: Parent (EIP)	OBR-42: Escort Required (ID)
OBR-4: Universal Service Identifier (CWE)	OBR-17: Order Callback Phone Number (XTN)	OBR-30: Transportation Mode (ID)	OBR-43: Planned Patient Transport Comment (CWE)
OBR-5: Priority (ID)	OBR-18: Placer Field 1 (ST)	OBR-31: Reason for Study (CWE)	OBR-44: Procedure Code (CNE)
OBR-6: Requested Date/Time (DTM)	OBR-19: Placer Field 2 (ST)	OBR-32: Principal Result Interpreter + (NDL)	OBR-45: Procedure Code Modifier (CNE)
OBR-7: Observation Date/Time # (DTM)	OBR-20: Filler Field 1 + (ST)	OBR-33: Assistant Result Interpreter + (NDL)	OBR-46: Placer Supplemental Service Information (CWE)
OBR-8: Observation End Date/Time # (DTM)	OBR-21: Filler Field 2 + (ST)	OBR-34: Technician + (NDL)	OBR-47: Filler Supplemental Service Information (CWE)
OBR-9: Collection Volume (CQ)	OBR-22: Results Rpt/Status Chng - Date/Time + (DTM)	OBR-35: Transcriptionist + (NDL)	OBR-48: Medically Necessary Duplicate Procedure Reason (CWE)
OBR-10: Collector Identifier (XCN)	OBR-23: Charge to Practice + (MOC)	OBR-36: Scheduled Date/Time + (DTM)	OBR-49: Result Handling (IS)
OBR-11: Specimen Action Code (ID)	OBR-24: Diagnostic Serv Sect ID (ID)	OBR-37: Number of Sample Containers * (NM)	OBR-50: Parent Universal Service Identifier (CWE)
OBR-12: Danger Code (CWE)	OBR-25: Result Status + (ID)	OBR-38: Transport Logistics of Collected Sample * (CWE)	
OBR-13: Relevant Clinical Information (ST)	OBR-26: Parent Result + (PRL)	OBR-39: Collector's Comment * (CWE)	

OBX			
OBX-1: Set ID - OBX (SI)	**OBX-8: Abnormal Flags (IS)**	OBX-15: Producer's ID (CWE)	OBX-22: Mood Code (CNE)
OBX-2: Value Type (ID)	**OBX-9: Probability (NM)**	OBX-16: Responsible Observer (XCN)	OBX-23: Performing Organization Name (XON)
OBX-3: Observation Identifier (CWE)	**OBX-10: Nature of Abnormal Test (ID)**	OBX-17: Observation Method (CWE)	OBX-24: Performing Organization Address (XAD)
OBX-4: Observation Sub-ID (ST)	**OBX-11: Observation Result Status (ID)**	OBX-18: Equipment Instance Identifier (EI)	OBX-25: Performing Organization Medical Director (XCN)
OBX-5: Observation Value (varies)	**OBX-12: Effective Date of Reference Range (DTM)**	OBX-19: Date/Time of the Analysis (DTM)	
OBX-6: Units (CWE)	OBX-13: User Defined Access Checks (ST)	OBX-20: Observation Site (CWE)	
OBX-7: References Range (ST)	OBX-14: Date/Time of the Observation (DTM)	OBX-21: Observation Instance Identifier (EI)	

EVN
EVN-1: Event Type Code (ID)
EVN-2: Recorded Date/Time (DTM)
EVN-3: Date/Time Planned Event (DTM)
EVN-4: Event Reason Code (IS)
EVN-5: Operator ID (XCN)
EVN-6: Event Occurred (DTM)
EVN-7: Event Facility (HD)

AL1
AL1-1: Set ID - AL1 (SI)
AL1-2: Allergen Type Code (CWE)
AL1-3: Allergen Code/Mnemonic/Description (CWE)
AL1-4: Allergy Severity Code (CWE)
AL1-5: Allergy Reaction Code (ST)
AL1-6: Identification Date (DT)

NTE
NTE-1: Set ID - NTE (SI)
NTE-2: Source of Comment (ID)
NTE-3: Comment (FT)
NTE-4: Comment Type (CWE)
NTE-5: Entered By (XCN)
NTE-6: Entered Date/Time (DTM)
NTE-7: Effective Start Date (DTM)
NTE-8: Expiration Date (DTM)

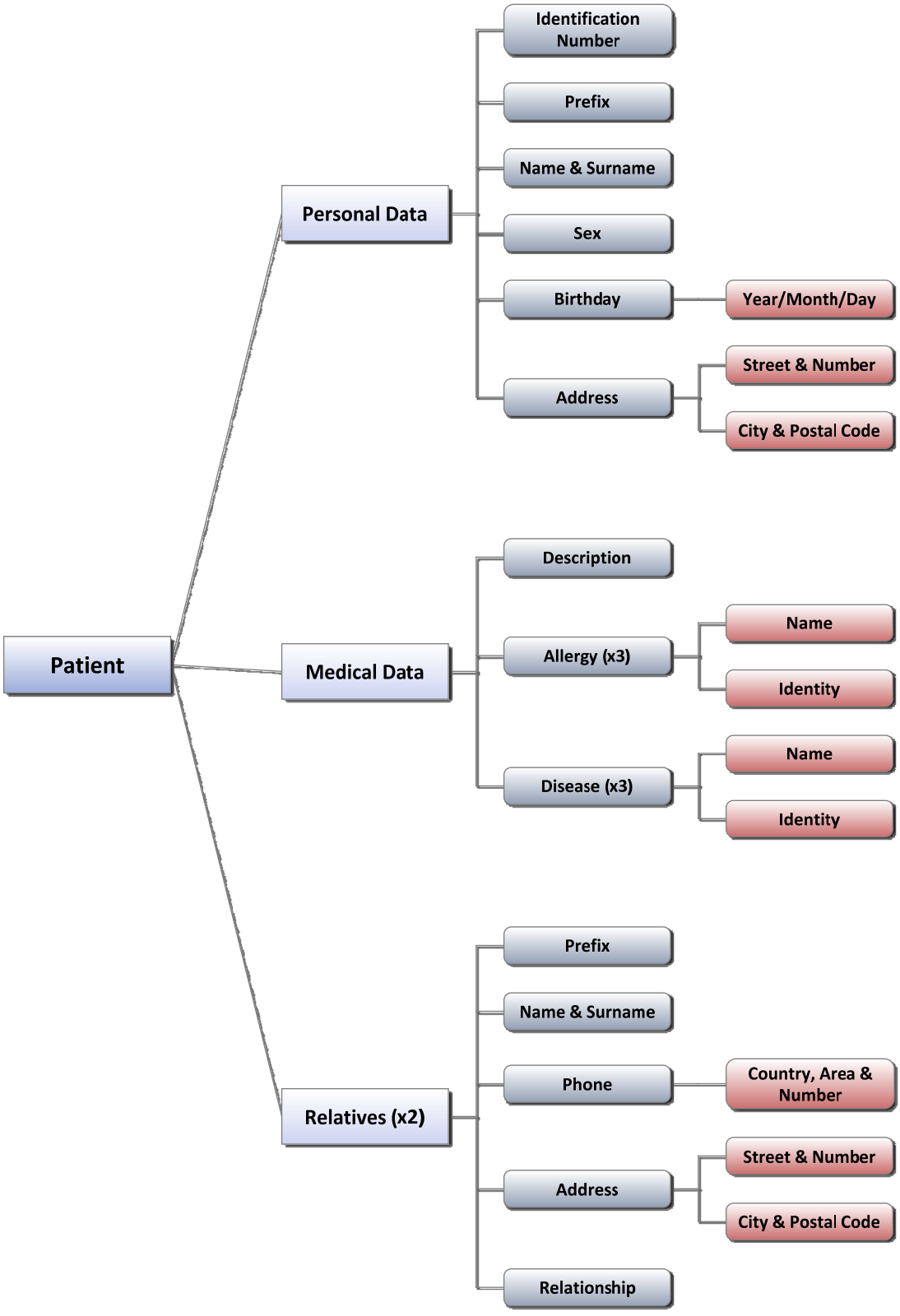
NK1			
NK1-1: Set ID - NK1 (SI)	NK1-11: Next of Kin / Associated Parties Job code/Class (JCC)	NK1-21: Living Arrangement (IS)	NK1-31: Contact Person's Telephone Number (XTN)
NK1-2: Name (XPN)	NK1-12: Next of Kin / Associated Parties Employee Number (CX)	NK1-22: Publicity Code (CWE)	NK1-32: Contact Person's Address (XAD)
NK1-3: Relationship (CWE)	NK1-13: Organization Name - NK1 (XON)	NK1-23: Protection Indicator (ID)	NK1-33: Next of Kin/Associated Party's Identifiers (CX)
NK1-4: Address (XAD)	NK1-14: Marital Status (CWE)	NK1-24: Student Indicator (IS)	NK1-34: Job Status (IS)
NK1-5: Phone Number (XTN)	NK1-15: Administrative Sex (IS)	NK1-25: Religion (CWE)	NK1-35: Race (CWE)
NK1-6: Business Phone Number (XTN)	NK1-16: Date/Time of Birth (DTM)	NK1-26: Mother's Maiden Name (XPN)	NK1-36: Handicap (IS)
NK1-7: Contact Role (CWE)	NK1-17: Living Dependency (IS)	NK1-27: Nationality (CWE)	NK1-37: Contact Person Social Security Number (ST)
NK1-8: Start Date (DT)	NK1-18: Ambulatory Status (IS)	NK1-28: Ethnic Group (CWE)	NK1-38: Next of Kin Birth Place (ST)
NK1-9: End Date (DT)	NK1-19: Citizenship (CWE)	NK1-29: Contact Reason (CWE)	NK1-39: VIP Indicator (IS)
NK1-10: Next of Kin / Associated Parties Job Title (ST)	NK1-20: Primary Language (CWE)	NK1-30: Contact Person's Name (XPN)	

MSA			
MSA-1: Acknowledgment Code	**MSA-3: Text Message**	**MSA-5: Delayed Acknowledgment Type**	**MSA-7: Message Waiting Number**
MSA-2: Message Control ID	**MSA-4: Expected Sequence Number**	**MSA-6: Error Condition (CNE)**	**MSA-8: Message Waiting Priority**

PV1			
PV1-1: Set ID - PV1 (SI)	PV1-14: Admit Source (IS)	PV1-27: Contract Period (NM)	PV1-40: Bed Status (IS)
PV1-2: Patient Class (IS)	PV1-15: Ambulatory Status (IS)	PV1-28: Interest Code (IS)	PV1-41: Account Status (IS)
PV1-3: Assigned Patient Location (PL)	PV1-16: VIP Indicator (IS)	PV1-29: Transfer to Bad Debt Code (IS)	PV1-42: Pending Location (PL)
PV1-4: Admission Type (IS)	PV1-17: Admitting Doctor (XCN)	PV1-30: Transfer to Bad Debt Date (DT)	PV1-43: Prior Temporary Location (PL)
PV1-5: Preadmit Number (CX)	PV1-18: Patient Type (IS)	PV1-31: Bad Debt Agency Code (IS)	PV1-44: Admit Date/Time (DTM)
PV1-6: Prior Patient Location (PL)	PV1-19: Visit Number (CX)	PV1-32: Bad Debt Transfer Amount (NM)	PV1-45: Discharge Date/Time (DTM)
PV1-7: Attending Doctor (XCN)	PV1-20: Financial Class (FC)	PV1-33: Bad Debt Recovery Amount (NM)	PV1-46: Current Patient Balance (NM)
PV1-8: Referring Doctor (XCN)	PV1-21: Charge Price Indicator (IS)	PV1-34: Delete Account Indicator (IS)	PV1-47: Total Charges (NM)
PV1-9: Consulting Doctor (XCN)	PV1-22: Courtesy Code (IS)	PV1-35: Delete Account Date (DT)	PV1-48: Total Adjustments (NM)
PV1-10: Hospital Service (IS)	PV1-23: Credit Rating (IS)	PV1-36: Discharge Disposition (IS)	PV1-49: Total Payments (NM)
PV1-11: Temporary Location (PL)	PV1-24: Contract Code (IS)	PV1-37: Discharged to Location (DLD)	PV1-50: Alternate Visit ID (CX)
PV1-12: Preadmit Test Indicator (IS)	PV1-25: Contract Effective Date (DT)	PV1-38: Diet Type (CWE)	PV1-51: Visit Indicator (IS)
PV1-13: Re-admission Indicator (IS)	PV1-26: Contract Amount (NM)	PV1-39: Servicing Facility (IS)	PV1-52: Other Healthcare Provider (XCN)

DG1-1: Set ID - DG1 (SI)	DG1-8: Diagnostic Related Group (CNE)	DG1-15: Diagnosis Priority (ID)	DG1-22: Parent Diagnosis (EI)
DG1-2: Diagnosis Coding Method (-)	DG1-9: DRG Approval Indicator (ID)	DG1-16: Diagnosing Clinician (XCN)	DG1-23: DRG CCL Value Code (CWE)
DG1-3: Diagnosis Code - DG1 (CWE)	DG1-10: DRG Grouper Review Code (IS)	DG1-17: Diagnosis Classification (IS)	DG1-24: DRG Grouping Usage (ID)
DG1-4: Diagnosis Description (-)	DG1-11: Outlier Type (CWE)	DG1-18: Confidential Indicator (ID)	DG1-25: DRG Diagnosis Determination Status (IS)
DG1-5: Diagnosis Date/Time (DTM)	DG1-12: Outlier Days (NM)	DG1-19: Attestation Date/Time (DTM)	DG1-26: Present On Admission (POA) Indicator (IS)
DG1-6: Diagnosis Type (IS)	DG1-13: Outlier Cost (CP)	DG1-20: Diagnosis Identifier (EI)	
DG1-7: Major Diagnostic Category (CNE)	DG1-14: Grouper Version And Type	DG1-21: Diagnosis Action Code (ID)	

V. Patient Structure



VI. Java vs. Android

The next lines address issues from a technical insight. Firstly the Java technology is mentioned and the parts that compound it. Later the main topic is the Android platform and in conclusion, there commonalities and differences are explained.

What is Java?

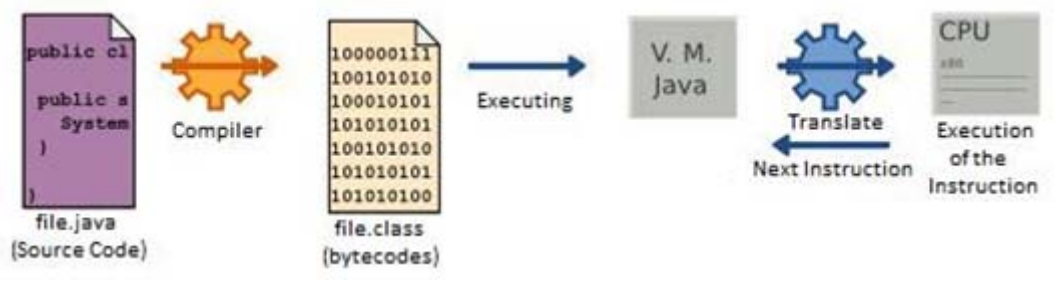
Java platform covers a whole range of technologies based on several concepts:

- A programming language to write applications.
- A set of libraries that facilitate the working developers. The main Sun APIs are divided into 3 large blocks:
  - Java SE: targeting desktop applications applets...
  - Java ME: focused on consumer resource-limited devices.
  - Java EE: for enterprise applications (web, distributed).
- Tools (JDK) to generate applications from source code.
- A virtual machine to run an application on any computer (JVM).
- Portability, is what defines a Java technology: to run an application anywhere, either in a Windows operating system, Linux, Mac, mobile phone or a browser.

This is an incredible amount of different environments, the secret of the Java´s portability is its virtual machine.

A program written in C or C++, is usually dependent on the platform (processor, operating system, etc.). However, Java programs are compiled in an intermediate format called bytecode. The bytecodes are not - normally - executable directly on any platform. However, a special program, the virtual machine, can translate them into code machine interpretable by the device.

Of course, a java program requires installing previously a virtual machine in the computer.



Traditionally, mobile devices have not had as much power as desktop computers. Therefore Sun Microsystems created a version of Java with limited library called Java ME (Micro edition), distinguishing it from Java SE (Standard edition, the complete version).

Java ME and Java SE are also different in some respects of their virtual machine and language. While the operation is identical (syntax and interpretation of bytecodes) VM Micro edition has only a subset of the functionalities of its big brother. The specifications of both are are documented in their respective Java Specification Request (JSR) [78].

Android

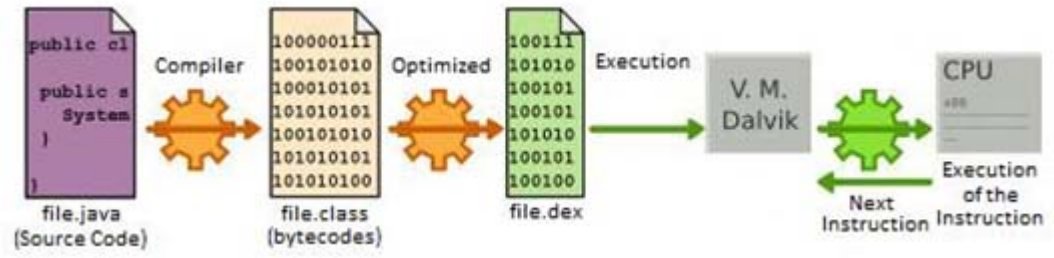
Google wanted to compete in the sphere of smartphones to position their search services. The Open Handset Alliance, a consortium led by Google and other major companies such as Intel, Motorola, Samsung and HTC, released in 2008 the Android operating system. The kernel was based on Linux and used many open project sources as SQLite or Webkit. The entire platform is distributed as free software, although the owner Google or some mobile companies can add sometimes non-free applications.

Regarding the development of applications, they wanted an environment comfortable for programmers and in where it was quick and easy to put existing applications. This is why they chose Java. It is a very popular language and there were many applications written in it, besides to several integrated graphical development environment.

However, using Java ME involved the purchase of licenses to Sun Microsystems and according to them, wasn't prepared for the next generation of smart phones. On the other hand, using the virtual machine and Java SE APIs directly - which had not been optimized for small devices - did not fit Android needs.

All these reasons made that Google developed other virtual machine: Dalvik [21]. Dalvik is the virtual machine used to run Android applications. But it is not actually a Java virtual machine.

The next lines show how the process of developing an Android application is:



Google Tools allows compiling Java code into byte code. The byte code is transformed into the binary format of the Dalvik machine: *.dex*. Dalvik is optimized for low memory (this is possible because it is a machine based on records and not on stacks) and is designed to run each application isolated in a different virtual machine.

In addition, Android provides a relatively complete subset of the API of Java SE. In this way, creating or porting java applications, whether in source or binary format results (in some cases) is easy to handle.

However, the Google API is not 100% compatible with the JavaSE. Libraries of the graphical environment (Swing, Awt) are not available on Android. In addition, concepts such as security management have been completely reinvented.

The result is a platform that is not strictly "Java technology" but simply uses the Java based language for creating applications.

Conclusion

Android and Java are two different platforms which share the same language, as well as basic programming libraries. In any case, today more than ever, this programming language is present in nearly all environments in the desktop, web and mobile devices in any of its variants.

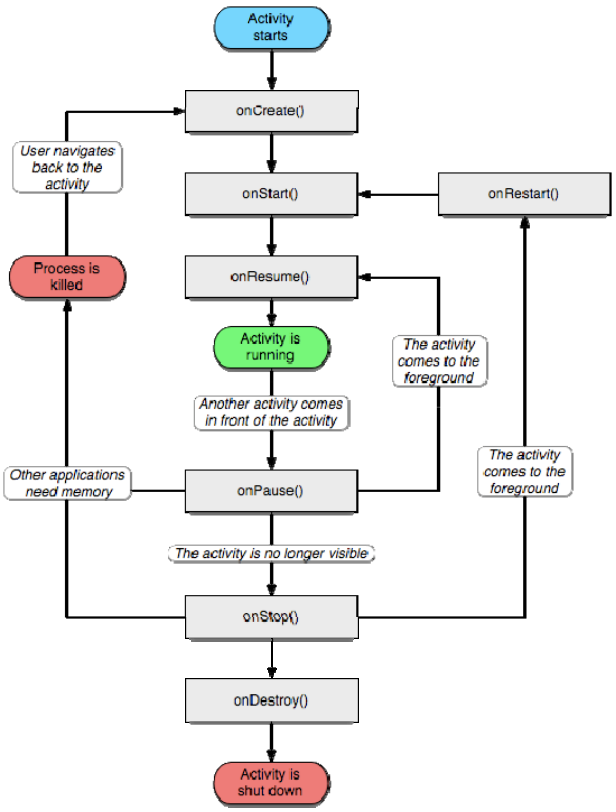
VII. Activity

An Activity [31] *is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window in which UI components can be set with* ***setContentView****(View). While activities are often presented to the user as full-screen windows, they can also be used in other ways: as floating windows (via a theme with windowIsFloating set) or embedded inside of another activity (using ActivityGroup).*

There are two methods almost all subclasses of Activity will implement [31]*:*

- ***onCreate****(Bundle) is where you initialize your activity. Most importantly, here you will usually call* ***setContentView****(int) with a layout resource defining your UI, and using* ***findViewById****(int) to retrieve the widgets in that UI that you need to interact with programmatically.*
- ***onPause****() is where you deal with the user leaving your activity. Most importantly, any changes made by the user should at this point be committed (usually to the ContentProvider holding the data).*

However, there are more methods in lifecycle of an Activity, which are represented in the next figure. These methods depend on the states which are found [31]*:*



- *If an activity in the foreground of the screen (at the top of the stack of activities), it is active or running.*
- *If an activity has lost focus but is still visible (that is, a new non-full-sized or transparent activity has focus on top of your activity), it is paused. A paused activity is completely alive (it maintains all state and member information and remains attached to the window manager), but can be killed by the system in extreme low memory situations.*
- *If an activity is completely obscured by another activity, it is stopped. It still retains all state and member information, however, it is no longer visible to the user so its window is hidden and it will often be killed by the system when memory is needed elsewhere.*
- *If an activity is paused or stopped, the system can drop the activity from memory by either asking it to finish, or simply killing its process. When it is displayed again to the user, it must be completely restarted and restored to its previous state.*

*To be of use with Context.**startActivity**(), all activity classes must have a corresponding <activity> declaration in their package's AndroidManifest.xml* [31].

VIII. Service

Most confusion about the Service [30] *class actually revolves around what it is not:*

- *A Service is not a separate process. The Service object itself does not imply it is running in its own process; unless otherwise specified, it runs in the same process as the application it is part of.*
- *A Service is not a thread. It is not a means itself to do work off of the main thread (to avoid Application Not Responding errors).*

Thus a Service itself is actually very simple, providing two main features [30]*:*

- *A facility for the application to tell the system about something it wants to be doing in the background (even when the user is not directly interacting with the application). This corresponds to calls to Context.startService(), which ask the system to schedule work for the service, to be run until the service or someone else explicitly stop it.*
- *A facility for an application to expose some of its functionality to other applications. This corresponds to calls to Context.bindService(), which allows a long-standing connection to be made to the service in order to interact with it.*

*There are two reasons that a service can be run by the system. If someone calls Context.**startService**() then the system will retrieve the service (creating it and calling its **onCreate**() method if needed) and then call its **onStartCommand**(Intent, int, int) method with the arguments supplied by the client. The service will at this point continue running until Context.**stopService**() or **stopSelf**() is called.*

*Clients can also use Context.**bindService**() to obtain a persistent connection to a service. This likewise creates the service if it is not already running (calling onCreate() while doing so), but does not call onStartCommand(). The client will receive the **IBinder** object that the service returns from its **onBind**(Intent) method, allowing the client to then make calls back to the service. The service will remain running as long as the connection is established (whether or not the client retains a reference on the service's IBinder.*

Global access to a service can be enforced when it is declared in its manifest's <service> tag. By doing so, other applications will need to declare a corresponding <uses-permission> element in their own manifest to be able to start, stop or bind to the service.

IX. NFC

Near Field Communication (NFC) [73] is a set of short-range wireless technologies, typically requiring a distance of 4cm or less. NFC operates at 13.56mhz, and at rates ranging from 106 kbit/s to 848 kbit/s. NFC communication always involves an initiator and a target. The initiator actively generates an RF field that can power a passive target. This enables NFC targets to take very simple form factors such as tags, stickers or cards that do not require power. NFC peer-to-peer communication is also possible, where both devices are powered.

Compared to other wireless technologies such as Bluetooth or WiFi, NFC provides much lower bandwidth and range, but enables low-cost, un-powered targets and does not require discovery or pairing.

	NFC	Bluetooth	Bluetooth Low Energy
RFID compatible	ISO 18000-3	active	active
Standardisation body	ISO/IEC	Bluetooth SIG	Bluetooth SIG
Network Standard	ISO 13157 etc.	IEEE 802.15.1	IEEE 802.15.1
Network Type	Point-to-point	WPAN	WPAN
Cryptography	not with RFID	available	available
Range	< 0.2 m	~10 m (class 2)	~1 m (class 3)
Frequency	13.56 MHz	2.4-2.5 GHz	2.4-2.5 GHz
Bit rate	424 kbit/s	2.1 Mbit/s	~1.0 Mbit/s
Set-up time	< 0.1 s	< 6 s	< 1 s
Power consumption	< 15mA (read)	varies with class	< 15 mA (xmit)

NFC harmonizes today's diverse contactless technologies, enabling current and future solutions in areas such as:

- Access control
- Consumer electronics
- **Healthcare**
- Information collection and exchange
- Loyalty and coupons
- Payments
- Transport

NFC provides a range of benefits to consumers and businesses, such as:

- **Intuitive**: NFC interactions require no more than a simple touch
- **Versatile**: NFC is ideally suited to the broadest range of industries, environments, and uses
- **Open and standards-based**: The underlying layers of NFC technology follow universally implemented ISO, ECMA, and ETSI standards
- **Technology-enabling**: NFC facilitates fast and simple setup of wireless technologies, such as Bluetooth, Wi-Fi, etc.)
- **Inherently secure**: NFC transmissions are short range (from a touch to a few centimeters)
- **Interoperable**: NFC works with existing contactless card technologies

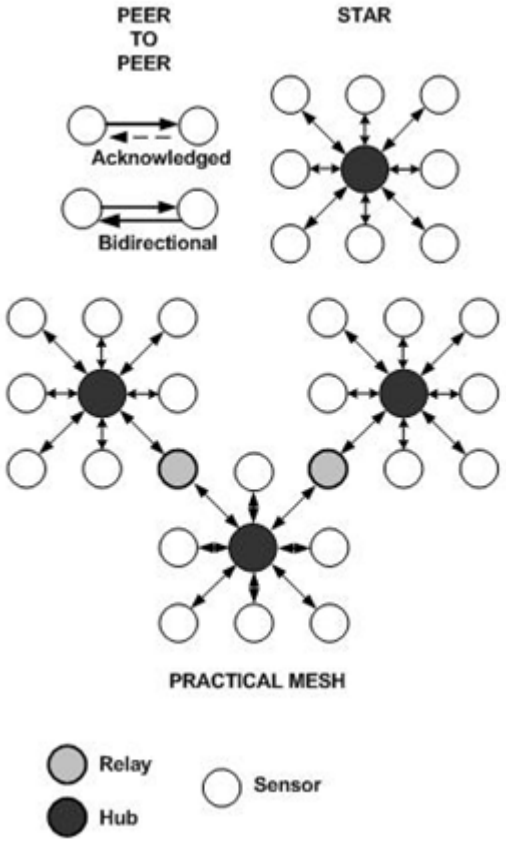
- **Security-ready**: NFC has built-in capabilities to support secure applications

X. ANT

ANT [10] *is a 2.4GHz practical wireless networking protocol and embedded system solution specifically designed for wireless sensor networks (WSN) that require:*

- *ultra low power - runs on a coin cell for years of operation;*
- *highly resource optimized - fits into a compact sized memory;*
- *network flexibility and scalability - self-adaptive and able to do practical mesh,*
- *easy to use with low system cost - operates independently with a single chip*

Proven with millions of deployed nodes, ANT™ is perfectly suited for any kind of low data rate sensor network topologies from peer to peer or star, to practical mesh in personal area networks (PAN) which are well suited for sports, fitness, wellness and home health applications. As well ANT is a practical solution for local area networks (LAN) for homes and industrial automation applications.



Thus, ANT has been designed to simplify network development and optimize network operational efficiency. ANT-powered network nodes can operate for years on coin cells compared to days or months for other technologies such as Bluetooth and ZigBee. Rich in capabilities, ANT provides [79]*:*

- *reliable data communications*
- *flexible and adaptive network operation*
- *cross-talk immunity*

In the official webpage is written a section that has the necessary code samples to implement ANT communication on an Android based phone that has the properly enabled hardware.

Nowadays, there are only some phones from the company Sony Ericsson [75] *with the ANT hardware. In the webpage, they also announce phones from HTC, like HTC Legend* [80]*.*

XI. SVN

This project was carried out on multiple computers therefore a versioning system was needed. Subversion is a content-versioning system, meaning that it records all important changes made in the files.

Tortoise SVN [81] *is an easy-to-use revision control / version control / source control software for Windows and possibly the best standalone Apache Subversion client there is. It is implemented as a Windows shell extension, which makes it integrate seamlessly into the Windows explorer. Since it's not integration for a specific IDE you can use it with whatever development tools you like.*

It is free software released under the GNU General Public License. In a few words, Tortoise SVN manages files and directories over time. Files are stored in a central repository. The repository is much like an ordinary file server, except that it remembers every change ever made to your files and directories. This allows to recover older versions of files and examine the history of how and when data has changed, and who changed It [81].