



UNIVERSIDADE DA BEIRA INTERIOR
Covilhã | Portugal

Mobile Platform-independent Solutions for Body Sensor Network Interface

Orlando Ricardo Esteves Pereira

Submitted to University of Beira Interior in candidature for the degree of Master of
Science in Informatics Engineering

Supervised by Prof. Doutor Joel José Puga Coelho Rodrigues

Departamento de Informática
University of Beira Interior
Covilhã, Portugal
<http://www.di.ubi.pt>

Acknowledgements

First of all, I would like to express my gratitude to my supervisor Professor Joel José Puga Coelho Rodrigues for his expertise, guidance, support and encouragement.

I am most grateful to the University of Beira Interior and Instituto de Telecomunicações (Covilhã), Next Generation Networks and Applications Group (NetGNA), Portugal, in the framework of the Project BodySens, and by the Euro-NF Network of Excellence from the Seventh Framework Programme of EU, for many kinds of all support that was given to me.

I owe particular thanks to iTech-ON and my friends Joel Silva Carvalho and João Paulo Oliveira.

My special thanks to Silvia Paula Melchior Fonseca, Pedro Ferreira Guedes de Carvalho and Ana Sofia Proença da Silva Dias for the text revision of this dissertation.

I am grateful to my closest family, my cousin Filipa Alexandra Esteves Ventura and my grandparents Maria Arlete Carvalho Matos e Ana Maria Campos Duarte for all the support and strength throughout these months of hard work and dedication.

Last, but not the least, my eternal gratitude to my girlfriend Carolina Damasceno Ribeiro Matos Lopes for her love, support, patient and constant encouragement. Is a enormous pleasure to live every day with your company.

My thanks to all the other friends not mentioned here.

Abstract

Body Sensor Networks (BSN) appeared as an application of Wireless Sensor Network (WSN) to medicine and biofeedback. Such networks feature smart sensors (biosensors) that capture bio-physiological parameters from people and can offer an easy way for data collection. A new BSN platform called Sensing Health with Intelligence Modularity, Mobility and Experimental Reusability (SHIMMER) presents an excellent opportunity to put the concept into practice, with suitable size and weight, while also supporting wireless communication via Bluetooth and IEEE 802.15.4 standards.

BSNs also need suitable interfaces for data processing, presentation, and storage for latter retrieval, as a result one can use Bluetooth technology to communicate with several more powerful and Graphical User Interface (GUI)-enabled devices such as mobile phones or regular computers. Taking into account that people currently use mobile and smart phones, it offers a good opportunity to propose a suitable mobile system for BSN SHIMMER-based networks.

This dissertation proposes a mobile system solution with different versions created to the four major smart phone platforms: Symbian, Windows Mobile, iPhone, and Android. Taking into account that, currently, iPhone does not support Java, and Java cannot match a native solution in terms of performance in other platforms such as Android or Symbian, a native approach with similar functionality must be followed. Then, four mobile applications were created, evaluated and validated, and they are ready for use.

Keywords

Wireless sensor networks; Body sensor networks; Healthcare applications; Mobile e-Health; Mobile monitoring application; Mobile and pervasive computing; Android; iPhone; Windows Mobile; Symbian; Mobile operating systems; Biosensor; Biofeedback.

Contents

Acknowledgements	iii
Abstract	v
Keywords	vii
Contents	ix
List of Figures	xiii
List of Tables	xv
Acronyms	xvii
1 Introduction	1
1.1 Introduction	1
1.2 Motivation	5
1.3 Objectives and Problem Definition	6
1.4 Main Contributions	8
1.5 Dissertation Organization	8
2 Related Work	11
2.1 Sensor Networks	11
2.2 Body Sensor Networks	13
2.3 Mobile Computing	14
2.4 Mobile Operating Systems	19

2.5	Mobile platform-independent solution	24
3	Requirements analysis	27
3.1	Essential requirements	27
3.2	Diagrams	28
3.2.1	Class Diagram	28
3.2.2	Activity Diagram	30
3.2.3	User Case	32
3.2.4	Sequence Diagram	33
3.3	SHIMMER platform	37
3.3.1	SHIMMER Software Development Kit	38
4	Mobile Framework	41
4.1	SHIMMER custom firmware	42
4.2	Application Programming Interface (API) custom architecture	43
4.3	Bluetooth support and communication	44
4.4	Algorithms for data acquisition	45
4.5	Classes and functionalities	47
4.6	Custom exceptions	48
4.7	Execution process	49
5	Symbian OS Application	51
5.1	Symbian User interface	52
5.2	Deploy, Tests and Validation	54
6	Windows Mobile Application	57
6.1	Windows Mobile application user interface	58
6.2	Deploy, tests and validation	61
6.3	Windows mobile emulator	64
7	Android OS Application	67
7.1	Android System Architecture	68

7.2	Activity Lifetime	70
7.3	Android application user interface	70
7.4	Deploy, tests and validation	75
7.5	Custom firmware 1.5 Bluetooth API	77
8	iPhone OS Application	79
8.1	iPhone Operating System (OS) architecture	80
8.2	iPhone application user interface	81
8.3	Deploy, tests and Validation	84
8.4	iPhone Comma Separated Values (CSV) auxiliary file format	86
9	Conclusions and Future Work	89
	References	93

List of Figures

1.1	Mobile OS GlobalStats 2009.	4
1.2	Mobile OS GlobalStats January 2009 to June 2010.	5
1.3	Mobile OS GlobalStats Evolution.	6
2.1	Illustration of a body sensor network.	15
2.2	Mobile Operating Systems divisions.	23
3.1	Class diagram.	29
3.2	Android custom Bluetooth classes.	30
3.3	Activity diagram.	31
3.4	Android OS Activity diagram.	32
3.5	User case diagram.	33
3.6	Sequence Diagram - Communication Setup.	34
3.7	Sequence Diagram - Communication Thread.	35
3.8	Sequence Diagram - Data processing.	36
3.9	Sequence Diagram - Data visualization.	37
3.10	SHIMMER front view.	38
3.11	SHIMMER back view.	38
3.12	SHIMMER hardware and dock station.	39
4.1	SHIMMER firmware diagram.	43
5.1	Symbian User Interface.	53
5.2	Symbian Data Presentation.	54

6.1	Windows Mobile debug window.	59
6.2	Windows Mobile preferences windows.	60
6.3	Windows Mobile real time graphical data presentation.	61
6.4	Windows mobile emulator with bluetooth support.	65
7.1	Android OS Version.	71
7.2	Android application debug window.	72
7.3	Android application preferences window.	73
7.4	Android application real time graphical data presentation.	74
8.1	iPhone OS Architecture.	80
8.2	iPhone application debug window.	82
8.3	iPhone application preferences window.	83
8.4	iPhone application real time graphical data presentation.	84
8.5	CSV file format.	87

List of Tables

5.1	Symbian Benchmark Tests.	55
6.1	Windows Mobile Benchmark Tests.	62
7.1	Android Benchmark Tests.	75
8.1	iPhone Benchmark Tests.	85

Acronyms

API	Application Programming Interface
BSN	Body Sensor Networks
CLDC	Connected Limited Device Configuration
CVD	Cardiovascular Diseases
CPU	Central Processing Unit
CSV	Comma Separated Values
DRM	Digital Rights Management
ECG	Electrocardiography
EMG	Electromyography
GDP	Gross Domestic Product
GUI	Graphical User Interface
IDE	Integrated Development Environment
IT	Information Technology
J2ME	Java 2, Micro Edition
LAN	Local Area Network
LCD	Liquid Crystal Display
MAN	Metropolitan Area Network

MID	Mobile Internet Devices
MIDP	Mobile Information Device Profile
nesC	Network Embedded Systems C
OMG	Object Management Group
OS	Operating System
PC	Personal Computer
PDA	Personal Digital Assistant
RF	Radio Frequency
RFComm	Radio Frequency Communication
RAM	Random Access Memory
ROM	Read Only Memory
RSA	Rivest, Shamir and Adleman
S60	60 Series
SAN	Storage Area Network
SD	Secure Digital
SDK	Software Development Kit
SHIMMER	Sensing Health with Intelligence Modularity, Mobility and Experimental Reusability
SPP	Serial Port Profile
TinyOS	Tiny Operating System
UI	User Interface
UML	Unified Modeling Language
USB	Universal Serial Bus

VM Virtual Machine

WAN Wide Area Network

WLAN Wireless Local Area Network

WSN Wireless Sensor Network

Chapter 1

Introduction

1.1 Introduction

People live in a age where technological development is rapidly evolving, driven mainly by Information Technology (IT) sector [1]. Similarly, in medical area there has been a remarkable improvement of medical resources [2], a major factor in increasing the quality and life expectancy of human beings. Another key factor in the increase of the average human life is without a doubt, changing harmful habits for healthy ones. Food standards and sport activities improve and provide better health preservation.

With the quality and life expectancy increasing is identified a large group of the global population, a group consisted mostly by aged population and / or with cognitive deficits. This group has its own characteristics, created not only by the inherent own aging, the social environment where they are inserted, but mainly by the negligence of that population relates to its health. This neglect increased not only through the inherent group characteristics but mainly by the increasing difficulty that the group has on access to any type of health mechanism. Currently there are several mechanisms available that can be good allies of any person for measurement of various medical factors ranging from blood pressure to blood glucose or body temperature. Such mechanisms are portable but, were not fully designed for overall user because the User Interface (UI) and inherent interpretation is not always easy regarding a specific subset of the population like this.

Health care costs are on the rise globally. In USA they stood at 16% of Gross

Domestic Product (GDP) in 2007 and are expected to account for 25% by 2015 [3]. Population aging, the process whereby the median age of the people in a country rises, is occurring worldwide. In 1950, the global proportion of population aged 60 years or over was 8% today it is 11% and by 2050 it is expected to have risen to 22% [4]. The cost of health care for those above the age of 65 is estimated between 2.8 and 3.5 times the other groups [5]. The fertility rate in developed countries has dropped drastically since the 1970s. By 2040 many of these people will have entered retirement; placing the increased cost of their health care on a substantially reduced workforce. These statistics indicate the need for change in the area of health care for those over 60 years old. There is much focus on the advancement of technology to aid in diagnosis and treatment of illnesses but comparatively little attention is paid to applying technology to the more mundane but equally significant health risks encountered by people in their own homes.

Using the extraordinary medical resources currently available, allied to this new reality, more and better health services must be provided especially to this population group. Currently there are solutions that provide the whole population the opportunity of their health to be remotely monitored. This solution is regularly referred to as telemedicine [6]. Telemedicine is important for remote monitoring of the state of a patient health. It makes use of modern information communication and technologies, to provide information and medical care to patients located at a distance. However, this system is somewhat limited and turns up not to properly inform the patient about his/her state of health, because the system was initially developed for a hospital/health center. The record of all activity of the patient is kept and analyzed in a center for that purpose, later on, the patient is informed about his/her health condition in a specific date. In existence of some phenomenon of relapse in the patient health, he/she should be instantly notified so as to enable it to carry out some type of medication to improve his/her health. Currently governments spend much money on costly systems and their application and does not reflect the final investment, so a system focused on the patient is less expensive and will have more success and acceptance. With the introduction of this scenario, it is necessary to offer the patient another way of monitoring their health status. A mobile, practical and easier mechanism, to allow any patient frequent measurements of their vital signs to chronological control of his/her state of health. Allaying this spirit with the latest technological innovations in mobile

devices and wireless communications, patients are presented with tools that have the capability of constant monitoring of his/her vital signs. This tool is named Biofeedback [7]. Biofeedback allows the patient to have directly control on various factors involved in their health from blood pressure, neurological activity, muscle tension, heartbeat heart or even skin temperature.

To allow the mechanism to be practical and mobile at the same time it must have some specific characteristics, so it should not be too large or heavy. Lately most existing biofeedback reading solutions are limited to physical device or with little mobility (laptop). The introduction of a solution that provides seamless mobility and keeping all the ruggedness, reliability and consistency in access to data that a medical application requires is principal factor. The patient must have the capacity to understand and analyze the data show without too much complication. A patient who has access to a system that is combined with portability, robustness and easy of access to data, is a patient with the ability to monitorize his health status or specific medical parameter at any physical location.

Modern mobile phones possess a dizzying array of features and technology. They are mass produced to keep, per unit, low costs and have gained complete acceptance into the lives of most in the entire world. While it was historically difficult to access the capabilities of many of these devices, successive generations have exposed more and more functionality. Many devices have processor and memory capabilities comparable to desktop computers of yesteryear. The long range, high speed communication abilities, along with low powered personal area network radios coupled with their inherent ubiquity, make them the perfect target for research of this nature.

The increasing number and importance of mobile devices has triggered intense competition among mobile OS creators and developers. Oldest companies such as Nokia, Microsoft and Apple are in a bid to sustain and capture bigger market share, while newer companies, that base their OS on Linux, attempt to gain their market share. A visual presentation of the market share is presented on Figure 1.1. It presents a graphic line display of the top eight mobile OS market share from January to September 2009 in Europe. The results show that iPhone OS and Symbian OS clearly lead the table.

The market is constantly evolving and changing. Therefore figure 1.2 presents the graphical display of the top eight mobile OS market share from January 2009 to June

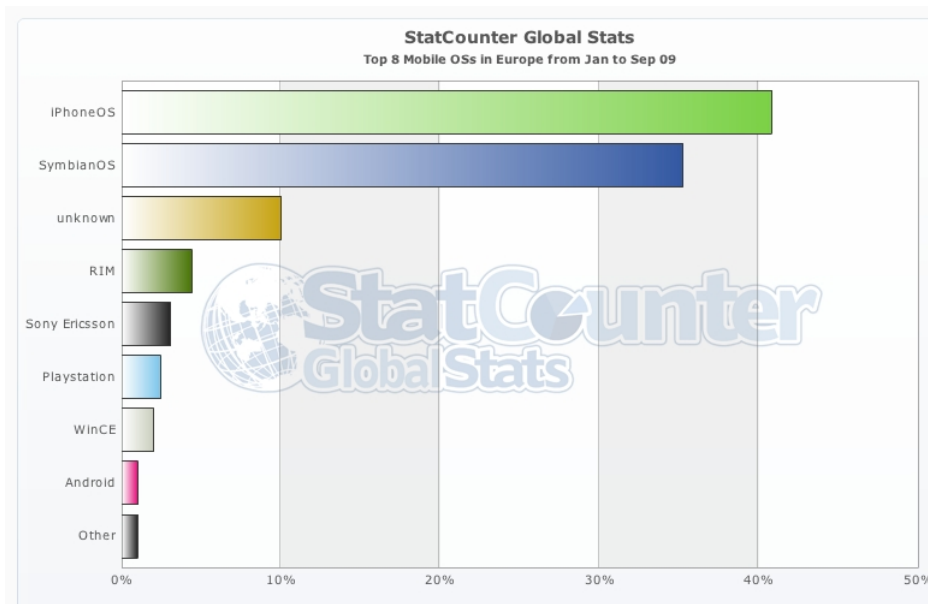


Figure 1.1: Mobile OS GlobalStats 2009.

2010. In this table, emergent technology such as Google Android and RIM Blackberry are gradually gaining ground and market share.

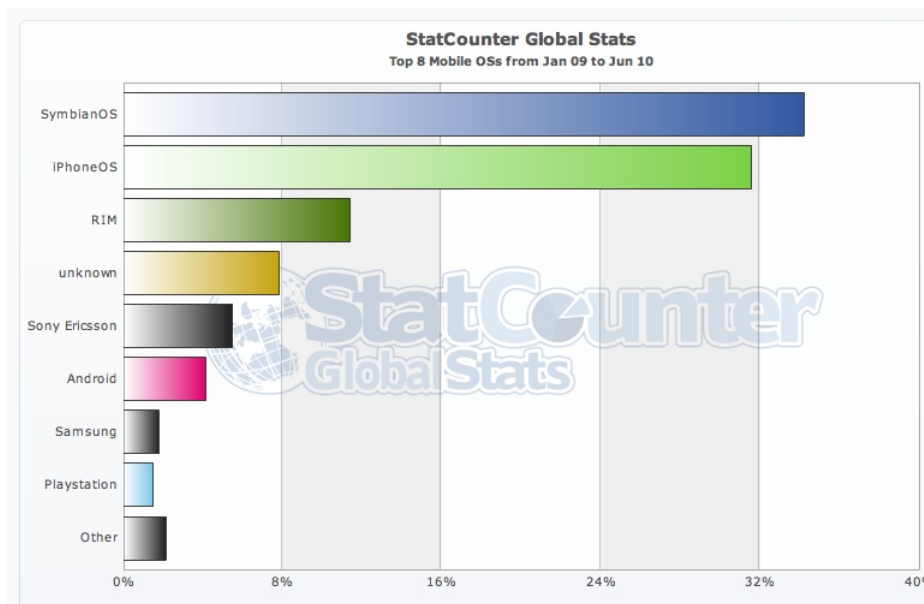


Figure 1.2: Mobile OS GlobalStats January 2009 to June 2010.

Figure 1.3 presents the evolution of each mobile OS on a period of 15 months. Looking at the table, can be verified that Symbian and iPhone are struggling for the pole position. Blackberry RIM, PALM and Android are gradually evolving, but they lack a very significant market share compared to their direct rivals. Nevertheless the stats can change quickly and a mobile OS which is currently low rated can climb the table and take the top of the same because the technology is always evolving and everyday is a new day for new features, characteristics and fashions.

1.2 Motivation

The highlight for this research work is the fact that is included in an area of great interest to the scientific and medical community. Technologies such as WSN and BSN became a part of our daily life, whether for personal or professional use. Still, it does not exist a complete implementation of such application as described in this dissertation in the literature, so this become a major motivation. In literature there are only implementation for a single device or mobile OS at a time. Freedom of choice of mobile OS furniture allowed us to analyze a wider range of OS and particularities of each system.

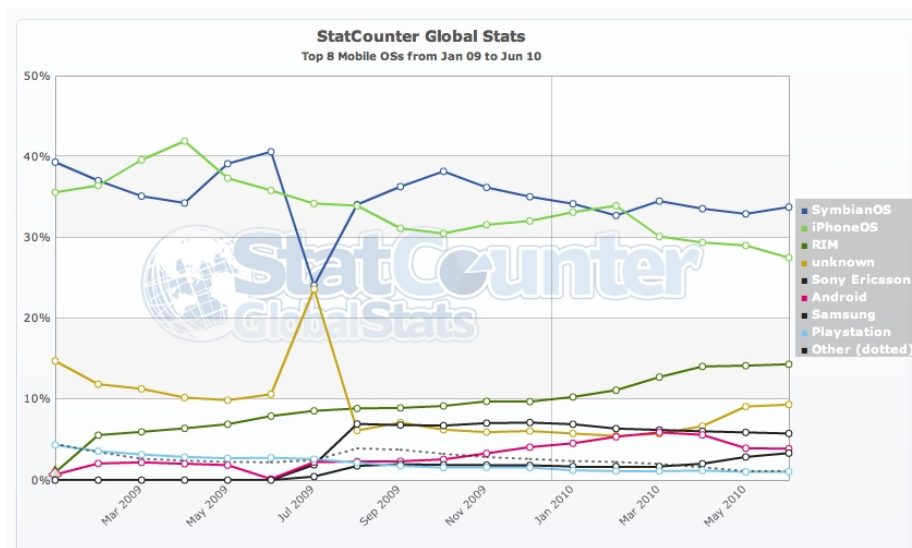


Figure 1.3: Mobile OS GlobalStats Evolution.

1.3 Objectives and Problem Definition

In this dissertation we try to demonstrate how mobile commodity consumer technologies can be used with BSN technology to produce a basic but useful overview of the state of a persons biofeedback parameter. While the data produced may not be as detailed as a checkup from a professional, the always-on nature of the technology may allow us to capture data that is otherwise unobtainable. Giving the patients themselves access to the data, allows them to better care for their own health. If a physiotherapist was to recommend a minimum and maximum amount of activity per day then the patient could easily be informed as to their progress in reaching such goals without the therapists intervention. By transferring this type of basic health care from clinical premises to the patient home significant strain can be removed from the health care system while improving the patient quality of life.

The data produced by the system could be remotely analyzed by professionals but also by expert systems which may be able to identify more subtle health issues than a standard check-up. By making use of real-time connectivity, emergency health conditions such as a heart attack can be quickly identified and will allow the system to immediately notify emergency services and provide potentially critical data about the conditions before the emergency. Finally, the data of multiple patients with a specific illness could possibly be collected and compared to better understand the effects of the illness.

This dissertation involves the creation of a mobile API targeted to several mobile devices that enables the acquisition, monitoring and treatment of signals from a BSN. The application developed for mobile devices intend to provide full mobility to health professionals and patients themselves. It communicate with the base station automatically, i.e. after the initial configuration steps, the user will not have to worry about the communication or even how it is being treated. Therefore several patients can make multiple biofeedback readings by collecting real-time values, processing and present them, allowing a better monitoring and especially a more timely and adjusted treatment without further delays. Data presentation will be made in a very simplified and dynamic form, so as to be easily understood.

To reach this objective the following intermediate objectives were identified and performed:

- Construction of a custom firmware to the SHIMMER-based device.
- Creation and deployment of a mobile API on iPhone OS.
- Completion of a test application with a SHIMMER-based BSN on the iPhone.
- Construction and testing of a mobile API on Symbian OS.
- Execution of a test application with a SHIMMER-based BSN on a device running Symbian OS.
- Construction and deployment of a mobile API on Android OS.
- Performance of a test application with a SHIMMER-based BSN on the Android device.

After the initial objectives were achieved, one more API targeting another mobile device was developed. Like this two new objectives were defined:

- Development and deployment of a mobile API on Windows Mobile OS.
- Completion of a test application with a SHIMMER-based BSN on the Windows Mobile device.

This work of research and engineering work is expected to produce not only the desired API, but also to provide the opportunity to disseminate the knowledge and software through, at least, a conference paper.

1.4 Main Contributions

This section is devoted to the scientific contributions of this dissertation to the state-of-the-art in wireless sensor networks, body sensor networks and mobile computing.

The first contribution is as part of a scientific paper that describes a symbian based mobile solution for intra-body temperature monitoring. The paper will be presented at 12th International Conference on E-Health Networking, Applications and Services (IEEE Healthcom 2010), at Lyon, France, on July (01-03), 2010 [8].

The second contribution is as part of a scientific paper that describes a mobile core-body temperature monitoring system using the Android OS API. The paper is presented at 5th International Conference on Body Sensor Networks (BodyNets-10), at Corfu Island, Greece on September (10-12), 2010 [9].

1.5 Dissertation Organization

This dissertation is organized in nine chapters and the chapters are organized as follows. This chapter, the first, presents the context of the dissertation, focusing the topics under studies, the motivation, the definition of the problem and main objectives, the dissertation organization and its main contributions.

Chapter 2 - Related Work - Presents the literature review on sensor network and BSN combined with mobile computing. Next, most related and relevant works highlighting biofeedback phenomena detection and acquisition using mobile devices for data display. Then, a variety of mobile OS are exploit. Finally, an introduction to our application is presented.

Chapter 3 - Requirements Analysis - This chapter presents all the requirements analysis for both mobile applications and custom SHIMMER firmware to support the data transferred between the base station and the inherent mobile device.

Chapter 4 - Mobile Framework - As the API has been developed for various platforms and systems, this chapter covers all the characteristics and unifications supported by all platforms.

Chapter 5 - Symbian OS Application - This chapter presents the inherent Symbian OS API. The custom user interface, the deploy method and all the tests and validation are presented.

Chapter 6 - Windows Mobile Application - This chapter introduces the inherent Windows Mobile OS API. It presents the custom user interface, the deploy method, custom emulator communication and all the tests and validation.

Chapter 7 - Android OS Application - This chapter presents the inherent Android OS API. It presents the custom user interface, the deploy method, custom bluetooth API and all the tests and validation.

Chapter 8 - iPhone OS Application - This chapter addresses all the creation process inherent to iPhone OS API. The user interface, the deploy method, the custom CSV system and all the tests and validation are presented.

Chapter 9 - Conclusions and Future Work - Presentation of a few more remarks about the dissertation and the future work for the presented work.

Chapter 2

Related Work

This chapter addresses the actual state of sensor network and BSN. The third section of this chapter describes some related works in mobile computing and ubiquitous systems regarding the medical area. After, a diversity of mobile OS are exploit. Finally a demonstration about the advantages of our API is included.

Data networking continues to evolve [10]. The demand for high-speed network infrastructure has been growing at an alarming rate. One way to categorize the different types of computer network designs is by their scope or scale. For historical reasons, the networking industry refers to nearly every type of design as some kind of arena network. Common examples of network types are: Local Area Network (LAN), Wireless Local Area Network (WLAN), Wide Area Network (WAN), Metropolitan Area Network (MAN), Storage Area Network (SAN), among others. LAN and WAN were the original categories of area networks, while the others have gradually emerged over many years of technology evolution. Such technology evolved to new networks types, such as WSN.

2.1 Sensor Networks

Wireless Sensor Networks consists of hundreds or thousands of low cost nodes which could either have a fixed location or randomly deployed to monitor the environment [11]. Due to their small size, they have a number of limitations [12]. Sensors communicate each other directly and the data usually ends at special nodes called base stations

(sometimes they are also referred as sinks). A base station link the sensor to another external network (like a gateway) to disseminate the data sensed for further processing. Base stations have enhanced capabilities over simple sensor nodes, since they must do complex data processing. Through this, they justify that base stations have, in general, better hardware then the simple nodes of the network.

WSN are a trend of the past few years, and they involve deploying a large number of small nodes. The nodes sense various environmental phenomena and report them to other nodes over the network. Sensor nodes are used for deployment in hostile environments over large geographical areas. Sensor networks have been a must have in a variety of domains, the primary domains at which sensor are deployed follow: military monitoring, environmental observation, building monitoring, healthcare.

Sensor networks can be used to monitor environmental changes, an example could be forestal fire detection. They can be randomly deployed in unknown and hostile areas and sense the exact location of a given phenomena. Other examples include water pollution, rainfall detection and observation or even air pollution. The military area uses sensor network for battlefield surveillance; monitor vehicular traffic, track the position of an ally or an enemy. Sensors can also be used in big buildings or factory monitoring climate changes. Thermostats and temperature sensor nodes are deployed all over the building's area. Plus, sensors could be used to monitor vibration that could damage the building structure. Another emergent area where sensors are used is the medial area over the healthcare service. Sensors can be used in biomedical applications to improve the quality of the provided care. Sensors are implanted in the human body to monitor medical problems like cancer and help patients maintain their health through biofeedback phenomena such as temperature analysis, blood pressure detection, Electrocardiography (ECG), Electromyography (EMG), among others.

One of the biggest problems regarding sensor networks is power consumption [13], which is greatly affected by the communication between nodes. One solution to his issue is the introduction of aggregation points to the network. The aggregation points reduces the messages exchanged between nodes and saves energy. Usually, they are regular nodes that receive data from neighboring nodes, perform some processing, and then forward the data to the next node. Another way for energy saving is setting the nodes into sleep mode if they are not needed and wake them up when required. Energy saving is one of the most challenge that deployers face when they create a WSN.

Sensor nodes are minuscule and composed by five main components [14]. The main

components of a typical sensor node include an antenna and a Radio Frequency (RF) transceiver to allow communication with other nodes, a memory unit, a Central Processing Unit (CPU), a battery and the sensor unit itself. Due to sensors limited capabilities, there are a lot of designed issues that must be addressed to achieve an effective and efficient operation of WSN. Since sensor nodes uses batteries for power and they are difficult to replace when consumed (deployed in remote or hostile environment), it is critical to design algorithm, protocols or technologies to minimize the energy consumption. To do so, implementers must reduce communication between sensor nodes, simplify computations and apply lightweight security solutions. Sensor nodes and inherent technology is always evolving and this technology must address some kind of security. However, as all other new technologies, security is not the top priority when designing something new. Security solutions are reticent when applying them to sensor networks. For example, cryptography requires complex processing to provide encryption to the transmitted data. Again, one solution regarding this aspect must be achieved, because in many cases the sensors acquire and transmit data critical for human and such data must not be compromised at any given time.

Sensor networks could be applied in the medical care, providing healthcare monitoring services. From the nature of the body sensing capabilities where the sensors are placed in contact with patient body, the designation evolved to BSN [15]. Such networks come answer to several challenges introduced by the biofeedback technology. The use of WSN on medical area became more then a simple vision, it is a reality [16], because such technology exists in present day.

2.2 Body Sensor Networks

Body sensor networks may be considered a sub branch of WSN and are constantly evolving to the biomedicine [17]. The main characteristic of this type of network concerns with the inherent technology, this is, BSN have the technology that allows the user to gather several phenomena from the human body. Using a specific sensor or a combination of sensors we can perform specific data sensing [18]. For example, it is possible to sense the body temperature, do a full ECG or even a EMG in an efficient and unobtrusive [19] way for the patient. Using vibration sensors, best known for accelerometers, it is possible to read heartbeat rates [20], the person movement or even a muscular extension. Basically, is possible to make full biofeedback [21] data

acquisition through this type of networks.

Parameters of medical data are critical to human life and behavior, so capturing them has to be the most accurate and reliable as possible. Although, a prerequisite for the transmission of data in real time is that the sensors must be attached to the human body. There are scenarios where sensor network require to operate for several constantly acquiring and processing data, so it is absolutely necessary to overcome the limitations of the network so that monitoring is the most reliable and safe as possible. Inherently to the loss of a sensor on this kind of network are the respective data parameter and physician who will also be lost, so is extremely necessary that the implementation of the communication support is the most robust as possible.

In most BSN systems, the sensors will communicate directly with the base station or a device with Bluetooth features. Such solutions regarding bluetooth communication are solutions that will consume more energy and are limited to the inherent characteristics of the bluetooth technology [22]. On the other hand, solutions where the sensors communicate directly with the base station are more complex and have increased probability to cause collision between data especially when the number of sensors used increases, the number of events is high and the real space where the phenomena sensing is less. The collision of data makes them not usable and unusable data, makes mislead both health professionals and the patient itself.

BSN are emerging technologies that provides unique uses in healthcare and other biomedicine applications [23]. Like WSN, the BSN consists of multiple connected nodes which together provide sensing, processing and communication capabilities. Figure 2.1 shows a scenario where a BSN collect human body parameters. Afterward, data is received by the sink node and sent directly to the laptop computer. In this specific case the laptop computer is responsible for gather, process, store and present all the data collected by the BSN.

2.3 Mobile Computing

Physiological applications requires to design solutions to address new challenges in energy efficient, cost and user interface. Powerful mobile devices have arrived at all walks of life and consequently made a mobile device the center of a BSN platform to provide the UI necessary for the user to see all the information regarding the BSN itself. Recent and powerful mobile devices become small processing units and

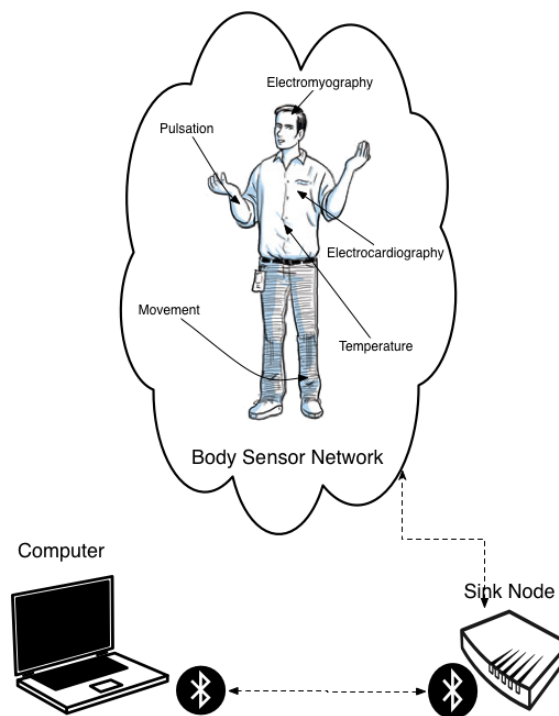


Figure 2.1: Illustration of a body sensor network.

provide developers with tools capable of receive, analyze and present data from any BSN correctly and flawlessly. This section presents some of the development projects regarding mobile platforms solutions to monitor and analyze human body phenomena.

Lin Zhong *et al.* [24] presented a Bluetooth based BSN platform for physiological diary applications. This approach relays on a three layer platform. The platform consists of one mobile phone, a wrist device and multiple sensors. The mobile phone is used as a sink for the BSN and it will manages every network member. The wrist device will be used to accomplish the UI and it will displays useful information under the instruction of the mobile phone, plus, the user can interact with the network conveniently. Through this implementation, sensor data acquisition is made by the mobile phone, which may not be adequate, given the sensible nature of data and reliability of the phone itself for such purposes.

Uwe Maurer *et al.* [25] present a innovative approach to healthcare monitoring through eWatch that is a wrist watch with unique characteristics. The watch is a wearable sensing, notification and computational enable platform highly available, instantly viewable, ideally located for various sensors and unobtrusive to users. eWatch features

a Liquid Crystal Display (LCD) (128x64 pix monochrome) to view information (limited to screen resolution) through a graphical view and some added functionalities like a calendar. The communication is supported via Bluetooth and provides link to a cellular phone or a Personal Computer (PC). A diverse range of sensors can be used such as light, motion, temperature, audio and eWatch provides visual, tactile and audio notification. Despite being a wrist device, it provides ample processing capabilities with multiple day battery life enabling continuous data sensing and user studies.

Reeves *et al.* [26] present an approach with a more specific purpose. Remote monitoring as an active element of care provision packages for older adults has the potential to significantly augment traditional social care. Such monitoring is made through integrated ambient and body sensing in order to sense specific patient data. The aim of the project is to achieve an environment that has ability to adapted to the patient lifestyle and answer all need related to the individual concerned. It is important underline that each person has different threshold values (minimum measurement values), so it is necessary that each patient system is configured for these same measurements. Thresholds values achieve flexibility and support a broad range of individual scenarios. Data can be seen at a mobile device or a PC located at the medical base.

Goh *et al.* [27] designed an integrated ECG beat detector on a Personal Digital Assistant (PDA) platform for the health screening process. For Cardiovascular Diseases (CVD), cardiac patient monitoring systems can play a important and vital role by providing early detection and constant monitoring. The proposed integrated system is centered on a feasible wireless architecture through both wireless and bluetooth communication. Bluetooth is used for short range connectivity between a wearable cardiac sensor module and a PDA. The collected and patient data will be managed by a micro data base on the PDA and wirelessly transferred to the remote server via WLAN. The system can be divided in three layers: Wireless sensor node; PDA and remote server. Plus, through the ECG UI patients can manage their own personal medical records and ECG data. The implementation of the application in PDA was realized on the Microsoft .Net platform using C# while the Microsoft SQL server is used for storing purposes.

Virone *et al.* [28] proposed a system architecture for smart healthcare based on an WSN. It specifically targets living on their own residences or others living on continuous remote health monitoring capable residents. The architecture is multi tiered and is composed by heterogeneous devices ranging from small sensors (wearable or placed

inside the living room), to mobile devices or more powerful stationary components. The division in five tiers proposes abstraction from each tier. The list of the tiers are the following: Body wearable sensors, Emplaced sensors, Backbone, Back-end databases and Human interfaces. Patient and medical staff can see the patient records on a standard PDA. Through the PDA the caregiver can request real time environmental conditions of the living space or access to the vital signs of the resident through a user friendly interface.

Chen *et al.* [29] proposed a four level hierarchy BSN for monitoring healthcare applications. Such proposal, separates the communication and controls modules, providing easier and faster BSN implementation and reducing hardware costs. There are four levels (System layer, Application layer, Sensor group layer, Sensor layer) and each level have its own characteristics and a relationship with the level below. The System layer is a PC in a hospital and the main actions of this layer are sending commands to the application layer and receiving and present data from it. The Application layer consists if several sensor groups and it will control all the groups. The Sensor group layer consists of several sensor nodes and it will control all the sensors on each group. The sensor layer is the last layer and is compounded by the sensor node itself. This approach offers a great advantage in terms of robustness and reliability but the central system is a standard PC and is intended for hospital facilities rather than personal use.

Dagtas *et al.* [30] present a wireless solution for monitoring people in need of medical assistance. The application relies on the use of a mobile device that supports Java 2, Micro Edition (J2ME) and inexpensive sensors that are best suited for the elderly and home bound people. The architecture has three main functions: Collect signals through a WSN using bluetooth connectivity and ZigBee; Optimized analysis of data; Real time monitoring and alert system. The data acquired will be analyzed directly on the mobile device and sent to some back end server for subsequent storage. Some events were programmed in order to alert the patient if some threshold is reached. The application runs on a Symbian enabled phone and supports Connected Limited Device Configuration (CLDC) and Mobile Information Device Profile (MIDP). The mobile device is the center of this approach and if it fails results in data loss.

Kirovski, Darko *et al.* [31] present Health-OS, a middleware platform for data sensing. The platform supports four key properties: near-transparency, favorable accuracy and reliability vs costs of operation; secure communication; strong analysis. The Health-

OS monitors, stores, transmits, analysis and presents various physiologically based signals. The objective is to unify an open group of sensing mechanisms using several computational resources. It is divided in three components: the sensing module, a mobile device and a PC. Here the mobile device is used only to display some simple information while the complex and complete information is only shown at the PC.

Srdjan Krco *et al.* [32] presented a mobile telemedicine based system, that relies on wireless telecommunication technologies and communication devices. The system uses a variety and intelligent wearable sensors to collect data. Those sensors are organized in a bluetooth network and are controlled through a central application installed on a PDA or a smart phone. Collected data is stored on this device and is periodically transferred to the doctor. Data transfer can be done in two different scenarios: real-time and store-and-forward. The first scenario is self explanatory, the doctor access all the data in real time monitoring. The second mode gather and stores data and periodically forward it to a remote database or to a doctor. Once again the mobile device is used as a central unit.

Bao *et al.* [33] proposes a novel solution to tackle the problem of entity authentication in BSN for mobile health. Physiological signals gathered by sensors have dual functions: for a specific medical application and for sensors in the same BSN to recognize each other by biometrics. This approach uses bluetooth, LAN and WLAN communication through all the points of the network. Data sensed is gathered in a central server and only then its sent to a end user that can access it in a ordinary PC or in a PDA or mobile device.

Jovanov *et al.* [34] presents a real-time personal stress monitor. Such system would benefit individuals by providing continuous data sensing and feedback about their stress levels and by helping their physicians to evaluate stress exposure between visits. This system is based on three layer architecture: The BSN with several sensors, a mobile device that acts as a gateway and a PC located on the physician office. Here the mobile device only acts as a aggregator of data. It have a small interface that is used for small sensor configuration. The data is fully displayed on the physician PC.

Dae-Seok Lee *et al.* [35] proposed a system that aims to measure various physiological vital parameters of patients and elderly persons. His/her health data is wirelessly transferred in Ad-hoc network to remote base station which is connected to the hospital / clinic main server. The main server acts as a display manager and stores only one minute of the gathered data. Afterwards, the recorded data is transmitted to a standard

PC or PDA located on the patient side. The limitation here is that the patient could not see her sensed data in real time, only a period of one minute at a time.

Several approaches exist with different implementations, architectures and design goals, but the same purpose: to provide health and physiological information to the patient, to the medical staff or for both. In this dissertation we aim to provide a easy, robust, secure, reliable and convenient mean to sense, process and show the bio-signals and inherent health information. Since nowadays almost everyone carries around a mobile phone, mobile device or a PDA, the choice to for the end device seems obvious. Moreover, the large majority of mobile phones today feature Bluetooth connection, the necessary condition to connect the device with the sink node of the BSN.

2.4 Mobile Operating Systems

Like a PC OS, a mobile OS is the software platform on top of which other programs run on a mobile platform. When the user acquire a mobile device, the manufacturer will have chosen the OS for that specific device. The OS is responsible for determining the basic functions and features available on the device, such as keyboards, communication with applications, wireless connectivity, text messaging, audio and more. The mobile OS will also determine which third-party applications can be used on your device. There are a multitude of technologies in use in the realm of portable computing. Some, such as Palm, have been responsible for the first incarnations of the mobile computing ideal [36], while others, such as Android have entered more recently, in what they hope to be the blurring point of the distinction between phones and a PDA. Mobile OS can be divided in five main sections: Symbian, Blackberry, Apple, Microsoft and Linux. Some of the more common and well-known Mobile OS include the following:

Android [37] is an OS for modern mobile devices. It is developed by Google and the Open Handset Alliance. Android is based upon the Linux kernel and is available under an open source license. Several handset manufacturers have announced plans to use the Android OS in upcoming devices. Android applications are written in Java, however unlike the J2ME where Java applications are less privileged than native applications,

Android Java applications form the base of the entire OS. In a similar manner as Symbian applications, Android applications from 3rd parties are able to make use of the majority of the device's functionality such as accessing the camera or 3D acceleration features. This is in support of the Android philosophy that all applications are created equal.

Bada [38] is a mobile OS designed for use in smart phones being developed by Samsung Electronics. This is a emergent mobile OS, so, the first Bada based phone is called Wave S8500 and was released released in 2010. Like Limo and Android, the Bada OS will be designed open source providing developers all the advantages of such approach. Bada has a kernel configurable architecture, which allows the use either of Linux kernel or another real time OS kernel, though the linux kernel is the most common used kernel on mobile devices. Bada devices support a wide range of sensors such as accelerometer, face detection, motion sensing and other. It will support native C++ code but also run J2ME applications.

iPhone OS [39] is a mobile OS fully developed and marketed by Apple Inc. It is used as a default OS of the iPhone, iPad and iPod Touch. The iPhone OS is a stripped version of Mac OS X, and is therefore a Unix like OS. Apple developed iPhone OS as a proprietary OS as has four abstraction layers: Core OS, Core Services, Media and the Touch interface. In terms of software development Apple published a public Software Development Kit (SDK), allowing the developers to make applications for the iPhone, iPad and iPod Touch through Xcode Integrated Development Environment (IDE) and using the programming language Objective C. Despite Apple supply a device simulator, the developer can only deploy his/her application after paying an iPhone Developer Program fee.

Limo [40] is a software platform for mobile devices. The LiMo foundation is a consortium of several handset manufacturers including Motorola and Samsung who have joined together to develop the LiMo Platform, a mobile OS based on Linux. There are over twenty handsets already running the LiMo platform. Limo has a modular plug-in architecture and supports Digital Rights Management (DRM). Limo developers will be able to use (when available) a standard SDK to write applications to Limo platform in three different ways, managed code, running in a Java virtual machine, native code or browser applications. Currently Limo is more of a tool for device creators to use than a platform for software developers to target.

Maemo [41] is another mobile OS targeting mobile devices that uses linux on his

foundation. Maemo is a modified version of Debian Linux distribution, tuned for mobile devices. Like a normal PC it uses the X window system for graphical UI. It also has a package manager similar to the Debian one. Maemo comes with a large number of built-in applications. Due in part to the open source nature of Linux, porting native applications to Maemo is a straightforward procedure. In terms of software development, the developers have a wide range of tools such as: native C, Java, Python, Ruby or even Mono.

MeeGo [42] was announced at Mobile World Congress in early 2010 and is another Linux based open source mobile OS aiming mobile devices. It is developed by Nokia and Intel and aims to merge the Maemo project with the Moblin project. This new platform is intended to run on a variety of hardware platforms including mobile devices, smart phones, in-car information devices, net-books and televisions with internet access. To withstand such devices MeeGo supports both x86 and ARM processors.

Moblin [43] is an open source OS and application stack for Mobile Internet Devices (MID), net-books, nettops and embedded devices. Moblin was built around the Intel Atom processor whose objective is to minimize boot times and power consumption. The Moblin platform offers the developers the way to create and customize a Linux file system for a given device thereby improving its functionality's.

OpenMoko [44] is another project aiming at running open source software on modern mobile devices. The distinction between this and other Linux based platforms is that OpenMoko releases the hardware designs to their devices under an open license. To date two handsets have been released, the Neo 1973 and the Neo FreeRunner. Both devices support the standard GSM bands and allow developers full access to the underlying hardware. The FreeRunner device compares favorably with more commercial handsets by providing wireless networking, two accelerometers and 3D graphics acceleration. Both devices run OpenMoko Linux, a specialized distribution of Linux compiled for the devices but essentially containing the same components as found on PC Linux distributions such as the X window manager or GTK. This allows most existing Linux applications to be ported to OpenMoko, an approach quite novel in the mobile computing world. OpenMoko has relatively little industry backing but has garnered a large following among open source advocates. It is not tied to any particular carrier, nor does it impose or intend on imposing any form of mandatory application or OS signing. It has been shown to work with kernels from other open source projects such as FreeBSD.

Palm webOS [45] was one of the first OS targeting mobile devices, though in its case it was specifically made for Palm PDA and related devices. Palm OS was first released in 1996 to power the brand new range of PDA. The rights to the software have since changed hands several times. While initial versions were groundbreaking in providing mobile computing to a large group of people, subsequent versions have provided little in the way of new functionality to compete with the newer developing mobile phone market. This, coupled with the relatively small line of devices it runs on, have led to its overshadowing by Symbian and other smart phone and mobile devices OS.

PyS60 [46] is an interpreter for the Python programming language that is compatible with Nokia S60 OS. The interpreter is signed by Nokia and therefore can be installed on any S60 compatible device. It allows for the creation of Python applications that do not need to be signed as they are not directly interacting with the OS. Python applications created in this way are somewhat limited in their functionality but are given more control over the device than applications running in the J2ME e.g. through the S60 Python API it is trivial for a script to initiate a phone call.

RIM BlackBerry OS [47] is the proprietary software solution, created by Research In Motion for a specific line of mobile devices called BlackBerry. This platform has a more professional side because it allows complete wireless activation and synchronization with several service providers carriers such as Microsoft Exchange, Lotus Domino or Novell GroupWise. This communication allows the user to synchronize, manage or update all of his/her emails, tasks, contacts and notes. Developers can easily develop applications through native J2ME code or through RIMlets. RIMlet is a CLDC based application that uses BlackBerry specific API and therefore will run only on BlackBerry devices.

Symbian OS [48] is an OS descendent of Psion EPOC designed for mobile devices and smart phones, originally developed by Symbian Ltd. and runs exclusively on ARM processors. It was originally owned by a consortium of handset companies but has recently become wholly owned by Nokia. Symbian allows the installation of binary applications and opens up many aspects of the device's hardware to them e.g. it is possible to completely replace the device manufacturer's default program for handling text messages with one purchased from a 3rd party developer. This level of hardware access is not possible with the J2ME, although J2ME applications are fully supported on Symbian devices. While Symbian describes the underlying OS there are a number

of different UI that run atop it. Applications must be specifically written for each UI, as each interface describes a different class of device.

Windows Phone OS [49], also known as Pocket PC is the second oldest smartphone OS, it was first released in 2000. Two improved variants of this OS, Windows Mobile 6 Professional and Standard were unveiled in 2007. Applications may be developed for Windows Mobile using Microsoft’s .NET framework. Applications written in any of the .NET languages compile to a common byte code that is run by the .NET virtual machine. Similarly to J2ME, the .NET libraries available on Windows Mobile are a reduced subset of the libraries available for the desktop edition. Programs written for one Windows Mobile device should work on any device running the same version of the OS.

Figure 2.2 presents a graphic display of mobile OS for smart-phones, PDA and net-books including the divisions by OS type and manufacturer [50].

Symbian (Nokia, Samsung, LG, Sony Ericsson, etc.)																													
05 9.1	05 9.2	05 9.3	05 9.4	Symbian Platform																									
S60																													
3.0	3.1	3.2	5.0	Symbian^2		Symbian^3		Symbian^4																					
3rd Edition	3rd Edition, Feature Pack 1	3rd Edition, Feature Pack 2	5th Edition (Symbian^1)																										
Research in Motion BlackBerry OS (BlackBerry)																													
4.1 Branch	4.2 Branch	4.5 Branch	4.6 Branch	4.6.1	4.7.0	4.7.1	5.0 Branch																						
4.1.0	4.2.1	4.5.0	4.6.0				5.0.0																						
Apple iPhone OS (iPhone, iPod Touch, iPad)																													
1.0		1.1		1.1.1		1.1.2		1.1.3		1.1.4		1.1.5		2.0		2.1		2.2		3.0		3.1		3.2		4.0			
1.0.1		1.0.2		1.1.1		1.1.2		1.1.3		1.1.4		1.1.5		2.0.1		2.0.2		2.1		2.2		3.0		3.1		3.2		4.0	
Microsoft Windows CE (HTC, Samsung, LG, Toshiba, Sony Ericsson, Dell, Acer, etc.)																													
5.0						5.2						6.0																	
Microsoft Windows Mobile						Microsoft Windows Phone 7						Microsoft Zune OS																	
5.0						6.0						6.1																	
						6.5						7.0																	
						6.5.1						6.5.3																	
						6.5.5						1.0																	
												1.x Branch																	
												2.x Branch																	
												3.x Branch																	
												4.x Branch																	
Linux - Smartphones																													
(HTC, Samsung, LG, Toshiba, Sony Ericsson, Dell, Acer, etc.)				(Nokia)		(Nokia, LG, Intel, etc.)		(Palm)				(Samsung)																	
Google Android				Maemo		MeeGo		webOS				bada																	
1.5				1.6		2.0		2.1		5.0				1.x															
1.0/1.1 Branch				1.2 Branch		1.3 Branch				1.4 Branch																			
1.0.2				1.0.3		1.0.4		1.1.0		1.2.0		1.3.1		1.3.5		1.3.5.1		1.4.0		1.4.1		1.4.1.1							
Google Chrome OS/Chromium OS				Intel Moblin		1.0 (Maemo 6.0)		Ubuntu Netbook Edition																					
Alpha Stages				2.0		2.1		8.04 (LTS)				8.10		9.04		9.10		10.04 (LTS)		10.10									
Linux - Netbooks																													

Figure 2.2: Mobile Operating Systems divisions.

2.5 Mobile platform-independent solution

After some solutions have been analyzed, we will now show the preeminence and key individuals of this approach for data gathering, analyzing and presentation for bio-signals from a BSN. There are different solutions slightly divergent than those previously mentioned, each other with different aims and methods of operation, but ultimately, they all have the same purpose, a purpose to provide correct, robust, reliable but especially simplified information to physicians and patients about the his/her health condition.

This dissertation aims to provide a suitable multi platform to make measurements of physiological parameters, such parameters are directly associated with biofeedback readings. The tool should not only make measurements, but also process and present them properly and in real time in a simplified form for the tool user. The UI in which the information is presented is a crucial part of the tool because the objective not only provide information to the medical staff, but allow a patient to self monitor his/her health parameters in a correct and easy way whenever he/she want and regardless of physical location.

The choices for the target platform where that for tool to be executed was facilitated by the fact that currently, virtually the entire population of developed countries owns an access to a mobile device, be it a mobile phone or PDA. There is, however a condition that the mobile device have to fill. This condition is related to the communication characteristic so the tool can perform their duties properly. The mobile device needs to be equipped with Bluetooth technology.

Previously has been presented several options to support communication between the sensor network and the mobile device. Our approach use a structure that uses a base station as a gateway between the BSN and the mobile device. The sensors continuously collect the information and send it to the base station. The base station is responsible for data aggregation and reconnaissance, plus, it send them via Bluetooth to the mobile device.

Thus a tool was built for different target devices with different OS which aims to provide health professionals and patients seamless mobility coupled with the necessary robustness inherent to an application in the health area. Trough the use of statistical data on the use of mobile OS and inherent characteristics of each OS within the subject, we choose four mobile OS to develop for: Symbian OS, Windows Mobile, Android and

iPhone OS. Each choice was based on four distinct parameters: Innovation, usability, overall utilization and device price. The Symbian approach is quite obvious, the devices supporting it (Nokia mainly) are the cheapest devices and a lot of users had one. Windows Mobile is the mid term approach where we can have cheaper devices and a lot of usability. Android and iPhone development aroused particular interest because they are the emergent technology on mobile devices of twenty one century. Both have a little of our four parameters and provide users an innovative application on a brand new device.

The developed framework will operate in a logical sequence of implementation in all devices. Initially will trigger a search for Bluetooth devices, capable of acquisition of signals from a BSN. After the search be carried out it enables a possible connection between the mobile device and that device (base station), using authentication to ensure data authenticity transferred between them. When the connection between devices is guaranteed, the user will have access to an interface for data visualization, a specific UI to perform biofeedback phenomena readings where is necessary to observe the signal variation over time such as intra body temperature or a ECG.

Chapter 3

Requirements analysis

The requirements analysis, is an essential process for developing a product, since it is here that all specifications and behaviors are studied. When it comes to software development its importance increases dramatically, because all the requirements analyses constrain both the software and its final operation. Therefore this chapter addresses the requirements analysis carried out before the development of our applications.

3.1 Essential requirements

Before the beginning of any software development or design of any diagram, a initial step must be done. It is extremely important to define the essential requirements for the final application. This analysis can sometimes be difficult due to the abstraction required, since it is necessary to have a global vision of something that has not yet been built. Therefore some essential requirements were found: Bluetooth connectivity; easy to use UI; storage of data acquired; alert system. The Bluetooth communication connect both mobile device and the BSN. Since the system must be simple and intuitive for the user the UI must be clean and understandable. The mobile device is used to receive, analyze and display all the incoming data, so it must have the possibility to store all the incoming data for future use. Data acquired belong to medical area and concerns health status of a person, therefore it is extremely important to implement an alert system to inform both user and/or medical staff of any given phenomena.

A good and robust study of the basic requirements can improve all the development

process and lead it to success. The contrary can ruin all the development and the final implementation.

3.2 Diagrams

The Object Management Group (OMG) released the Unified Modeling Language (UML). The purpose of UML was to provide developers with a stable and common design language, that could be used to develop computer applications. Through UML, IT professionals can disseminate system structure and design plans. Since UML is not a methodology, yet it does provide several types of diagrams that, when used within a given methodology, increase the ease of understanding of an application under development.

UML diagrams are divided in three types: Structure, Behavior and Interaction diagrams and each type has its characteristics and implementations. Structure type diagrams includes the following: Class, Component, Composite class, Deployment, Object, Package and Profile diagrams. The Behavior type includes the following: Activity, State machine and Use case diagram. Finally interaction diagrams includes the following: Communication, Interaction, Sequence and Timing diagrams.

In the following sections is possible to analyze the Class Diagram from the Structure type, the User Case and Activity diagram from the Behavior type and the Sequence Diagram from the Interaction type diagrams. Since all solutions were designed and built as just one the classes, methods, objects are the same. Despite all the mobile OS differences, characteristics and specifications, the development process was done to unify all the code produced. Yet there are still differences and those differences are shown where available.

3.2.1 Class Diagram

Class diagram is a representation of the structure and relationships of all classes that serve as a model for objects. Figure 3.1 presents the class diagram but were taken the attributes so as to smooth its readability. Due to the fact that were developed four distinct API would be required to present four distinct diagrams. The development process was done with one purpose in mind, *similarity*. Therefore the classes are similar with the inherent differences from the OS singularities and programming language.

The black color represents the classes and methods that are *equal* on all implementations. The green color represents the classes and methods that are Android OS exclusives. Blue classes and methods represents the Python classes and its exclusivity.

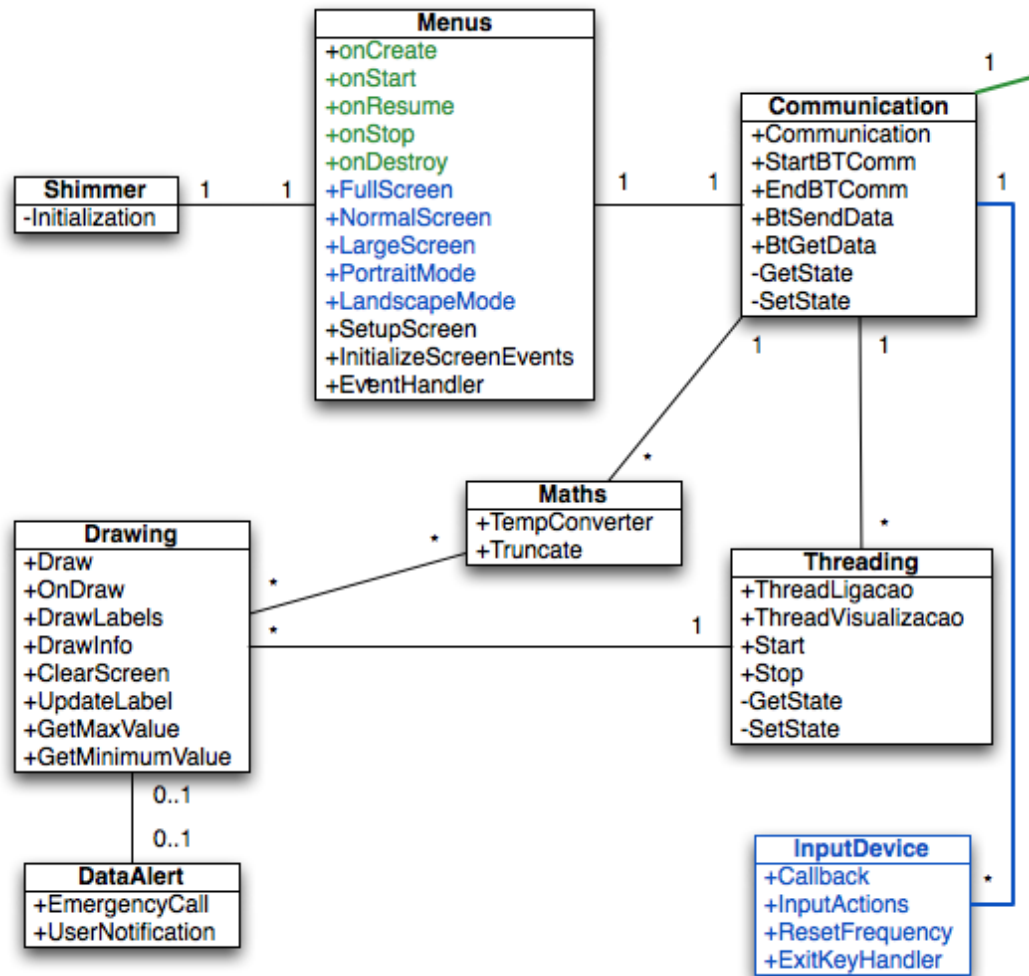


Figure 3.1: Class diagram.

As it was necessary to build custom classes for Bluetooth connectivity for the Android platform following on Figure 3.2 the inherent classes a presented.

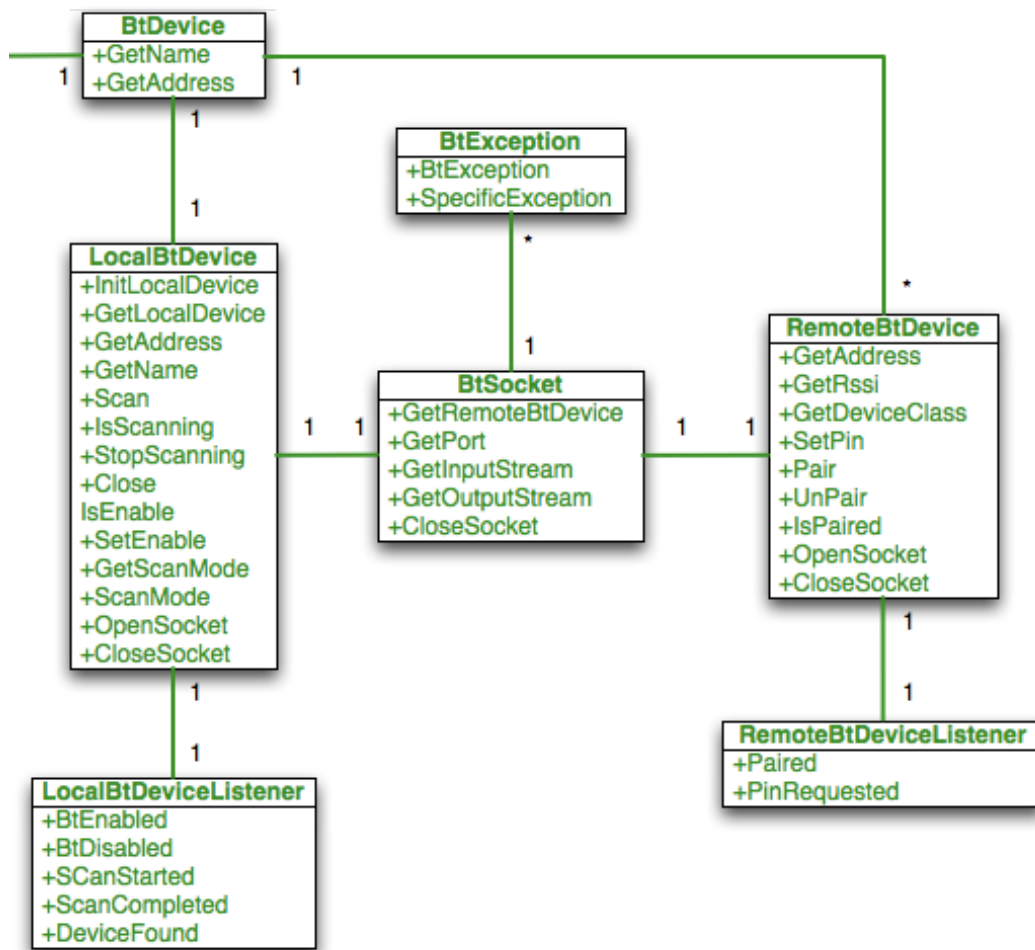


Figure 3.2: Android custom Bluetooth classes.

3.2.2 Activity Diagram

Activity diagrams are graphical representation of workflows of stepwise activities and actions. Each activity and action support choice, iteration and concurrency. Such diagrams can be used to describe a operation of a given application module and it shows the overall flow of control. Figure 3.3 presents the activity diagram of the platform.

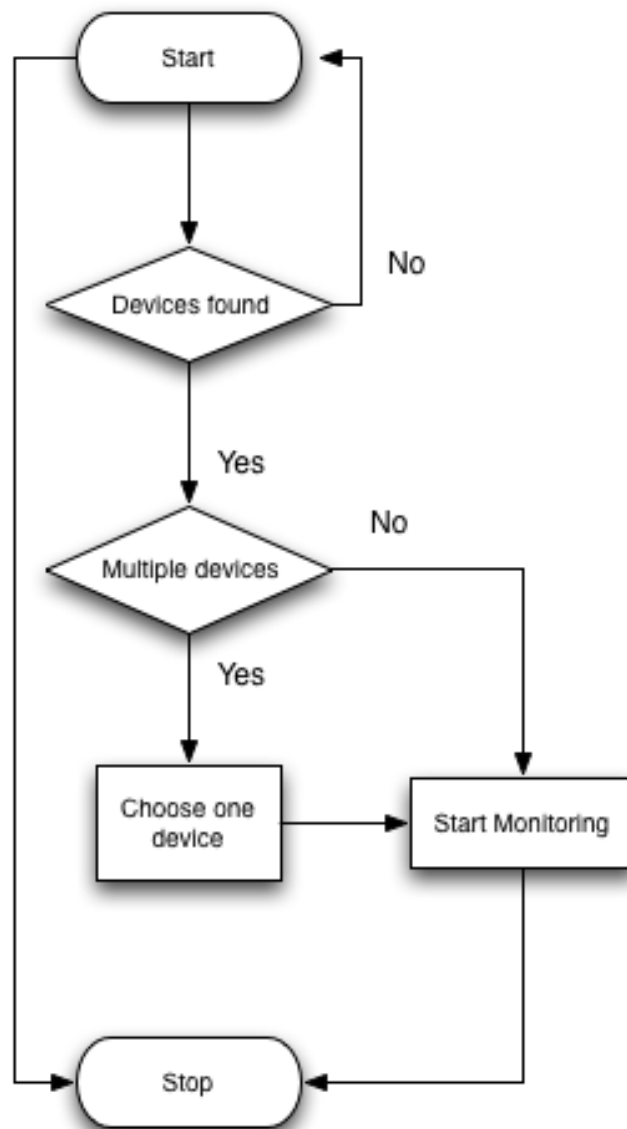


Figure 3.3: Activity diagram.

Since Android OS has a specific activity lifecycle it deals with every application through a unique system, so Figure 3.4 presents the Android OS activity diagram.

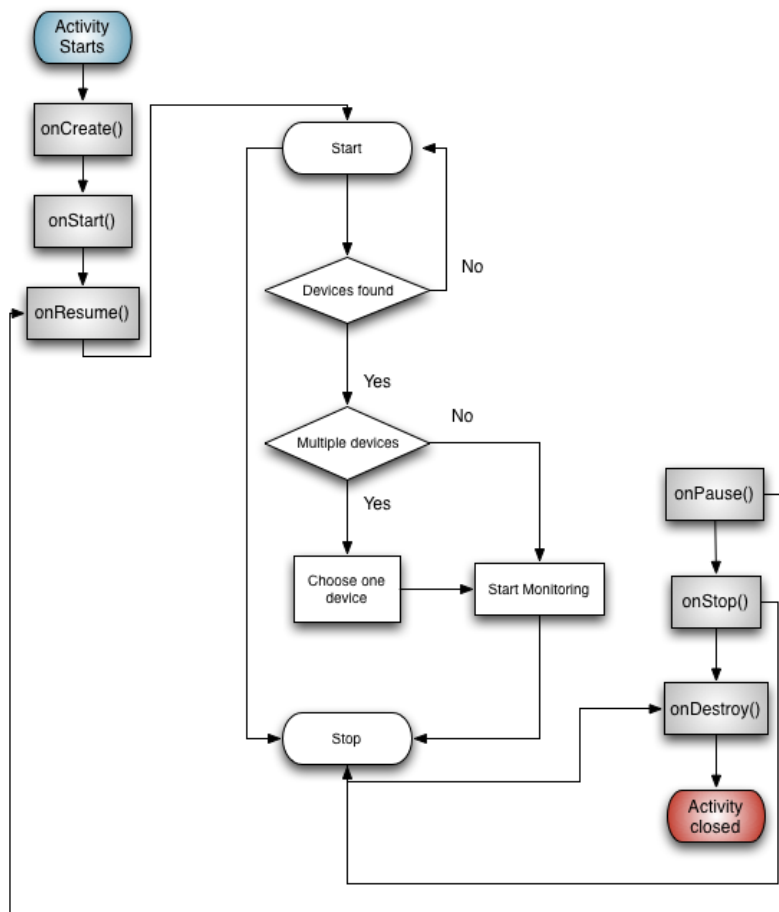


Figure 3.4: Android OS Activity diagram.

3.2.3 User Case

User case diagrams represent a graphical overview of the functionality provided by a system in terms of actors. Such diagram displays the actors goals and any dependencies between them. The user case diagram for this platform is presented at Figure 3.5.

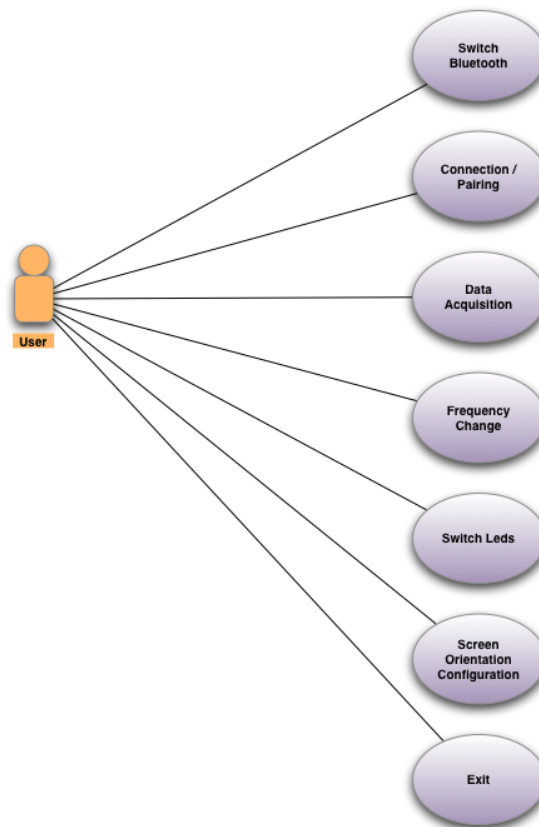


Figure 3.5: User case diagram.

3.2.4 Sequence Diagram

Sequence diagrams are diagrams that represent the process sequence, that is, messages will be passed between objects of the application. Such diagrams are useful because it simplifies in a logical sequence of behavior of objects, their classes and methods. It signs the procedure for a single use case and shows objects and messages passed between those use case objects.

Every sequence diagram is important but some has more relevance than others, therefore the more relevant are: Communication Start and Stop; Threading; Data acquisition, processing and visualization. Figure 3.6 presents the communication setup sequence diagram. The user has all the control over the communication process, so the beginning or the end of the communication is performed by the user. If a connection is already established between the mobile device and the SHIMMER the program will finish it, otherwise, a flag for communication start is sent to the SHIMMER device.

At the same time the thread *ThreadLigacao* is created in order to support all the communication.

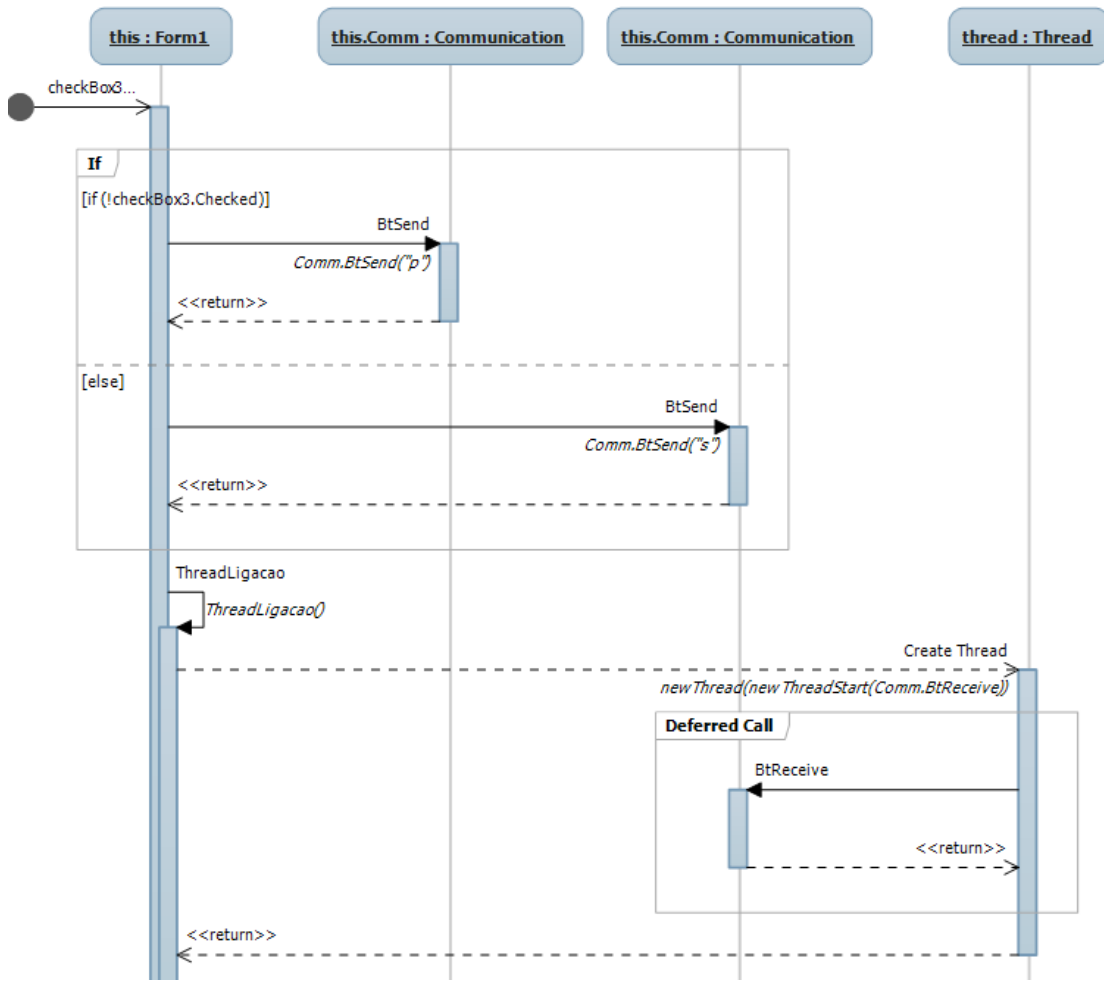


Figure 3.6: Sequence Diagram - Communication Setup.

In Figure 3.7 is presented the communication thread sequence diagram that supports all the communication and data exchange between the mobile device and the SHIMMER platform.

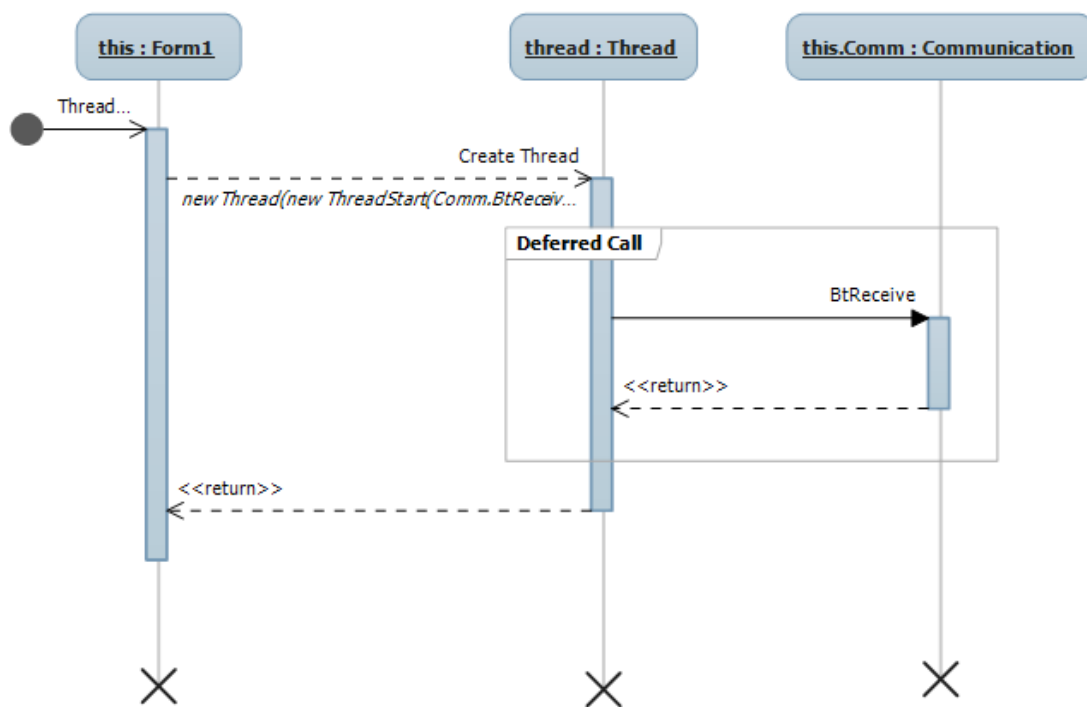


Figure 3.7: Sequence Diagram – Communication Thread.

An essential part of the SHIMMER API development is the data acquisition, processing, and visualization, so in Figure 3.8 is presented the sequence diagram that represents the processing and visualization process. Since the diagram is big, it has been divided in two parts for a better understanding. The first step is to convert the value received in bytes to the corresponding format, in this case a temperature value.

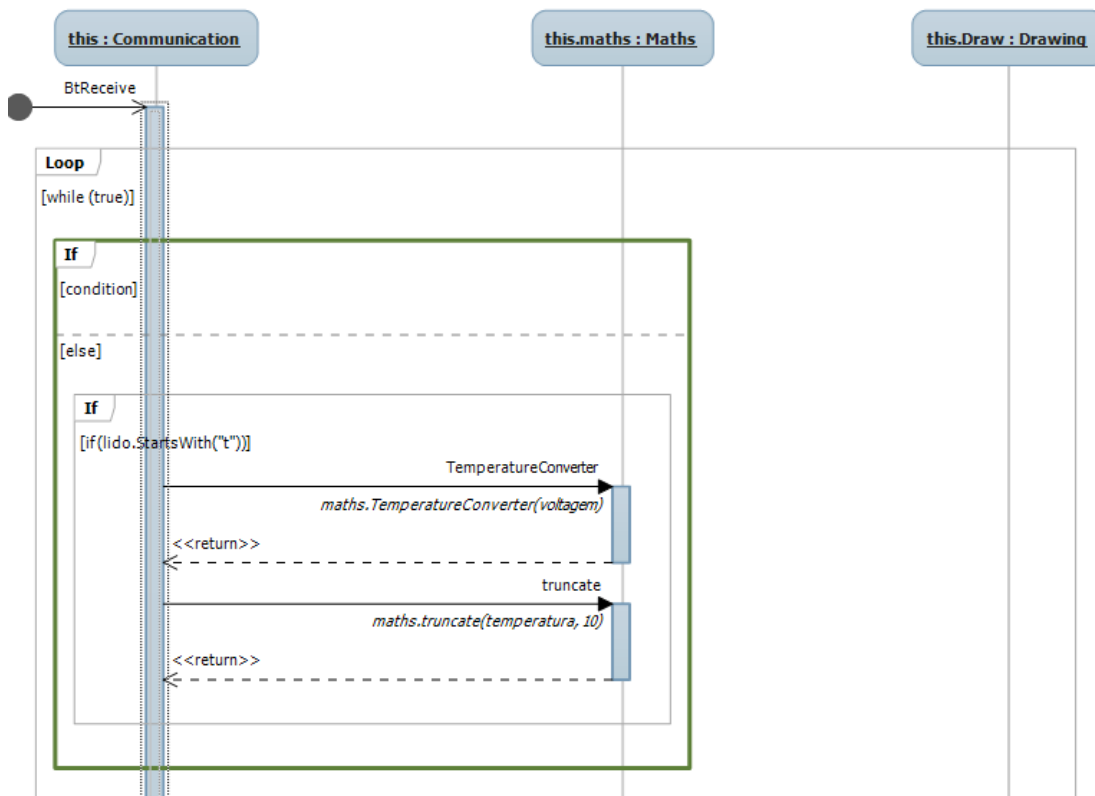


Figure 3.8: Sequence Diagram - Data processing.

Figure 3.9 presents the visualization part. Each time a conversion was made some mechanisms have to be made. Firstly the UI is changed in order to inform the user that the application still runs and that is waiting for another value. With the value in the correct format the maximum and minimum value are calculated. Through these calculations is possible to inform the user in real time about the inherent values.

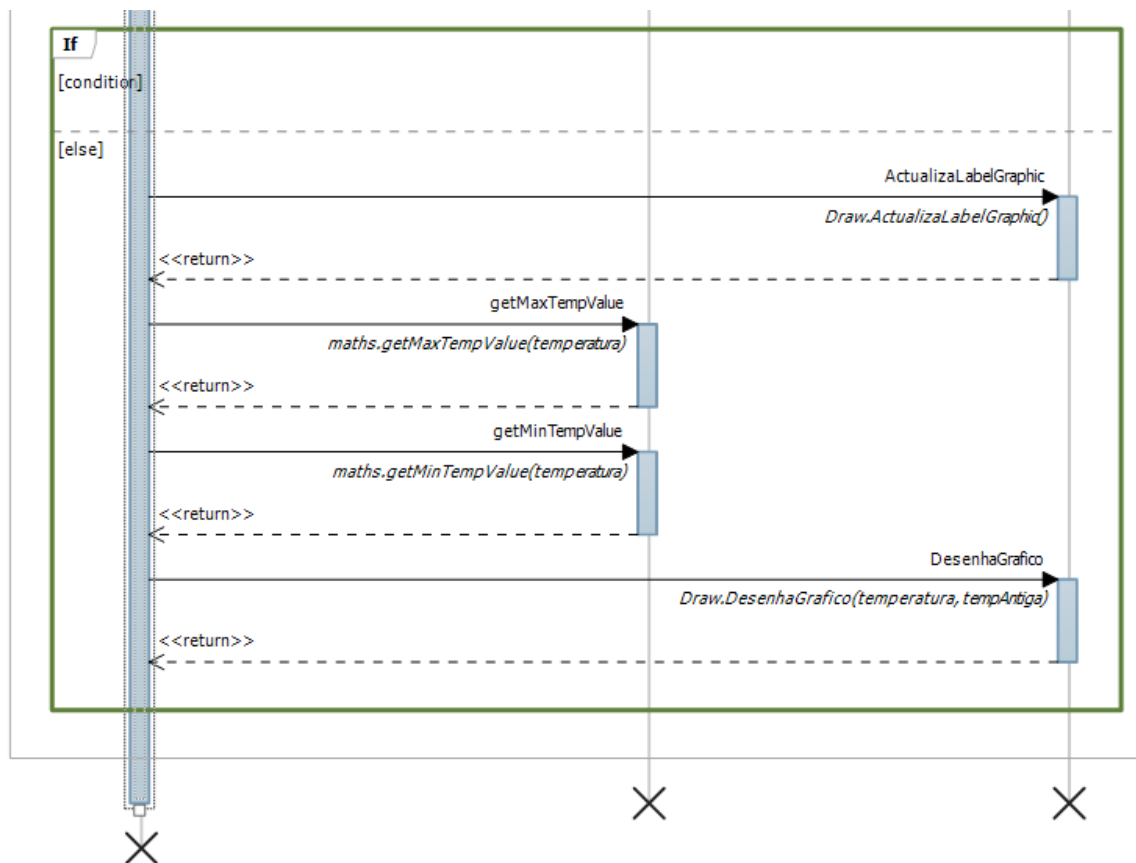


Figure 3.9: Sequence Diagram - Data visualization.

3.3 SHIMMER platform

This section presents the relevant features of the hardware platform used for this dissertation. This platform is called SHIMMER [51] and it acts as a base station unit for the BSN. Intel digital health group designed SHIMMER with Tiny Operating System (TinyOS) [52] support. It consists of the following components: MSP430 CPU running at 8MHz, bluetooth radio, 2.4GHz IEEE 802.15.4 Chipcon wireless transceiver, a tree axis accelerometer, a MicroSecure Digital (SD) slot for up to 2GBytes and an internal lithium-ion battery [53]. Through external boards it is possible to increase the SHIMMER features and functionalities. Despite all this characteristics and hardware, they are all compacted in a very small form factor (2.0 x 4.5 cm) [54]. Figure 3.10 presents the top view of the SHIMMER platform while figure 3.11 presents the bottom view with the components identification.

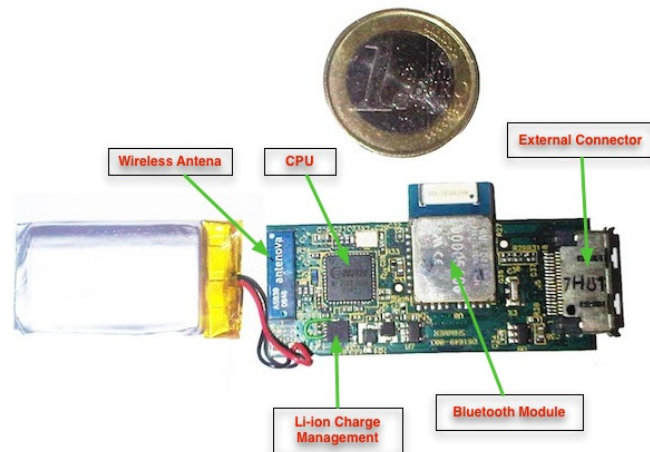


Figure 3.10: SHIMMER front view.

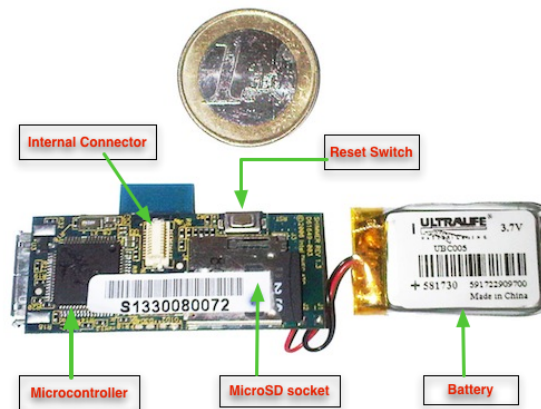


Figure 3.11: SHIMMER back view.

3.3.1 SHIMMER Software Development Kit

This section describes the development environment used to create the firmware to operate and control the SHIMMER platform [55]. Since the dissertation objective was to develop something new and innovator there was a need to develop a custom SHIMMER firmware in order to support mobile device connectivity and data transfer.

SHIMMER SDK included the following hardware: a SHIMMER platform; Dock station for battery charge and firmware deploy, 2GBytes MicroSD card and a Universal Serial Bus (USB) cable. The programming language used was Network Embedded Systems C (nesC), a programming language used to build applications for the TinyOS platform. To achieve this goal was used the SHIMMER developers Linux Virtual Machine (VM). This VM is a regular Linux system with additional configurations for TinyOS. As soon as we need to compile the firmware code the following command must be executed:

- make clean shimmer

If the code has no errors and it is ready for a real test it can be deployed to the SHIMMER platform using the following code:

- make shimmer install bsl,/dev/ttyUSBX

To upload succeed, the SHIMMER must be placed at the dock station and the USB connected to the computer. Figure 3.12 presents the SHIMMER SDK.



Figure 3.12: SHIMMER hardware and dock station.

Chapter 4

Mobile Framework

A software framework [56], in computer programming, is an abstraction in which a collection of classes and applications, libraries of SDKs and APIs help the different components to work together. Frameworks can be seen as software libraries in that they are reusable abstractions of code wrapped in a well defined API, yet they contain three key distinguishing features that separate them from standard libraries and they are: Inversion of control; Extensibility; Non-modifiable code; Inversion of control dictates that the overall program flow of control is dictated by the framework and not by the caller. The extensibility characteristic permits the user to extend it usually by selective overriding or specialized by user code providing specific functionality. The third characteristic concerns with the code itself, in other words, the framework code is not allowed to be modified. Although users can extend the framework and implement newer characteristics.

There are numerous mobile devices with different characteristics and OS. An ideal solution would be to have an application or framework, that could be simultaneously be implemented across larger number of devices as possible. However, and because the actual characteristics of each OS and the guidelines of this dissertation, was defined that each OS has his own implementation. Trough a specific approach and implementation for each system using the same functionality the implementation itself will gain performance and robustness due to the inherent characteristic of each OS and because it is fully build and directed to it.

Therefore the next sections will show the common characteristics between all the

API implementation for the different devices and OS.

4.1 SHIMMER custom firmware

The custom SHIMMER firmware has several specific characteristics and configurations, whose were created specifically for this dissertation. Next is explained the behavior and operation mode of the firmware. The SHIMMER will connect to the mobile device trough bluetooth and it always waits for commands over the connection. The commands are sent trough the mobile device, which in turn, receives data from the SHIMMER. Once a command is received and it is valid then it proceeds accordingly.

The firmware supports various commands such as: Start, Stop and Acquire. Each command results in different operation. The Start command indicates that SHIMMER must starts data collecting. If a bluetooth connection is available the data are simultaneously sent to the mobile device and stored at the SD card, otherwise it only stores them at the card for further analysis. That way, SHIMMER prevents any data loss associated to bluetooth connection and allows future analysis of data. All the data analysis is performed in real time. The Stop command, as the name suggests, stops the data collection.

Finally, the Acquire command preforms the transfer of all the storage data on the SD over bluetooth communication to the mobile device. Figure 4.1 presents the SHIMMER firmware diagram. This firmware provides all the above-mentioned features.

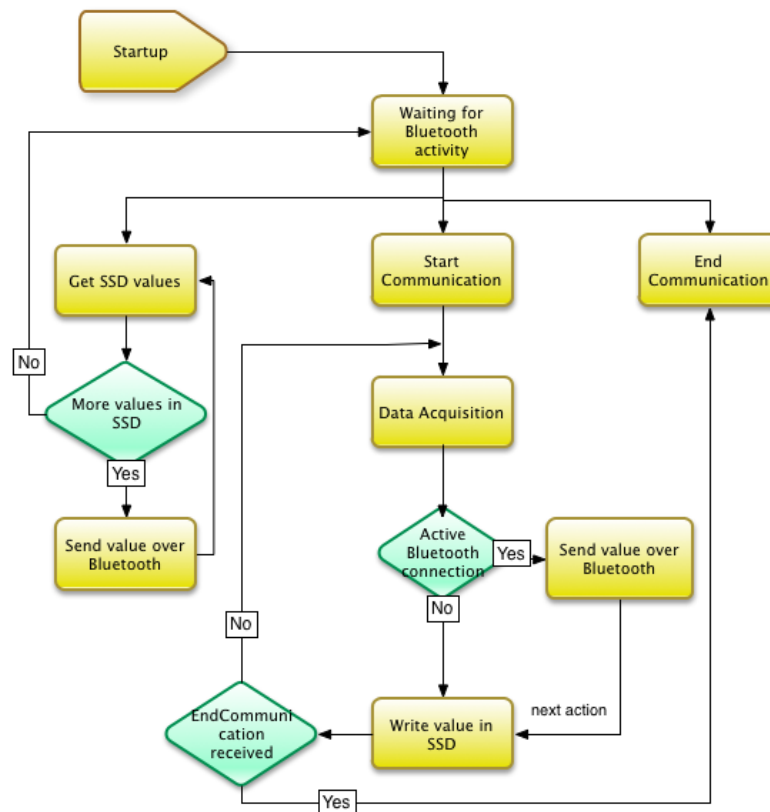


Figure 4.1: SHIMMER firmware diagram.

After the conclusion of the custom firmware, it has the ability to connect and transfer data with a mobile device. Therefore the next goal was to develop the inherent mobile devices API.

4.2 API custom architecture

The proposed architecture, allows extraction of system functionalities, where each layer is responsible for an individual task. Sensors are responsible for data collection, sensing health parameters of a given person. The sink node assumes the data collection and storage, protecting them from the external world and non-authorized accesses. The mobile device assumes data processing and presentation of all received data. Consequently, if the mobile device malfunctions, the connection is broken or is not present, data are not compromised because they will be storage in real time on the sink

node. In addition, the mobile device is independent from the involved BSN technologies because it may use a wired communication, a 2,4GHz wireless radio or any other. The only thing that really matters for a perfect data gathering, processing, and presentation is the type of connection between both mobile device and sink node. This proposed system architecture and its development, testing and validation are easier to accomplish because every module can be worked and tuned as a single part.

4.3 Bluetooth support and communication

Bluetooth is one of the most widely available wireless technologies currently available. Used by over 2 billion bluetooth-enabled devices, such like cell phones, mobile devices, laptops, gaming consoles and many other devices, it is the predominant wireless personal area networking technology. The mobile device connects to the sink node via Bluetooth connection, providing a mobile way for data gathering, processing, and visualization.

The sink and the BSN are protected from the outside world through a specific connection port and a unique password that is hard coded. This password is a prerequisite for device pairing and using it correctly will give access the bluetooth connection, data acquisition, analysis, and presentation. On the development processes, our main focus were on system robustness, convenience, and user-friendliness. The data gathered are critical to human life and behavior, therefore very sensitive too, so the application must be robust. The mobile tool must perform robust Bluetooth connection between mobile and the sink node, thus checking for errors on every single stream received with a very tight control over the communication. The application has two main functional blocks, the communication and presentation block. The first block assumes all the Bluetooth communication with the sink node, featuring connection establishment and management. This block has all the necessary configurations to successfully pair both mobile device and sink node. The second block deals with data analysis, processing, and presentation to the user. Every aspect regarding the data is performed here.

In the current implementation, data flows in streams with a specific format. Once again, the stream format was created and tuned to achieve better security parameters. The bluetooth connection has its advantages in term of user friendliness but have some faults in terms of security [57]. Since Bluetooth communication does not provide a fully

secure connection and data exchange, some additional methods were implemented to achieve more security, robustness and reliability.

Firstly is applied a Rivest, Shamir and Adleman (RSA) cryptography algorithm to the bytes acquired by the sensors. The RSA algorithm has over thirty years, and it is a algorithm known to be suitable for signing and encryption of data. RSA algorithm is widely used in common electronic protocols. The inherent security is directly related with a sufficiently long key used for encryption. After this initial step the data is fully encrypted. In order to increase robustness and security, another additional security mechanism was implemented. In cryptography, a Caesar cipher, is a widely known encryption technique and it is a substitution cipher. This substitution cipher is applied to all the message sent over Bluetooth. In order to improve security in the exchange of data between devices was decided to not send the keys or some information regarding the encryption over the communication channel. So all the keys and dictionaries were hard-coded on the mobile devices and the SHIMMER.

In a given scenario where the communication is submitted to a sniffer or any other attack the data is not compromised if detected. Since the detectors does not know the stream structure nor the encryption associated to them, the data acquired is encrypted and not readable. Only a authenticated and paired mobile device running the developed API can receive, process and present the data correctly.

By default, the sink node sends data with a frequency of one second, but this parameter could be changed in the application graphical interface. It can range from 1 second to 10 minutes, according to user selection.

Since the sink node of this prototype is very small, offering the possibility for being placed on small places such as patient belt or pocket, it offers a mobile and portable solution for real time data monitoring, processing, and visualization. While bluetooth communication is used in most devices, there was a setback in the development process. As Apple does not offer direct bluetooth API support for developers a exclusive method was performed.

4.4 Algorithms for data acquisition

The second functional block of the API is composed by the data presentation algorithm. Data, is an extremely important factor of the application, because it uses human vital

signs, so, the application must perform error tests constantly. It must be the most robust and precise as possible, so all the data received must be precisely analyzed in looking for possible errors. Each time the application receives data from the sensor node, it will perform specific tests in order to guarantee that user really see valid information and not false positives. The application suspects of all received data, since it is received through bluetooth and data may be corrupted or compromised. If some data was cataloged as corrupted or compromised the application will notify the user about it immediately. It is very important to keep record of all gathered data in order to find some kind of pattern, even if the pattern is from possible corrupted data.

In this approach, data flows in streams. Each stream has a specific data format. By default, the sink node captures and sends the streams with a frequency of 1 second. Since the first received data until the final value, it passes through several algorithms. The goal is to convert the received data, in bytes, to the inherent correct graphical presentation. This step has several algorithms linked together to reach the end result that is the correct point draw on the mobile device monitor. The first step is to strip the correct range of bytes from the stream received to post analysis. This step is essential to security improvement because the received stream have some associated flags. After that, and if the bytes number are correct, we must strip the correct value from the stream. After this, we have guarantees that we are working with the correct data and we can go to the next step. Now, the data conversion to an appropriate temperature value, in Celsius degrees, should be performs with another algorithm. All the transmitted data is organized in bytes, and then a transformation is needed. The algorithm must convert a 12-byte value, which corresponds to 4096 single values, in the equivalent temperature value. Afterwards, the temperature is striped and ready for display drawing. This step is very robust and has several related activities because all these activities are done in real time and they are used each time that new data is received. These activities will be reproduced on the display by informing the user of the real temperature, maximum and minimum values or even the data frequency.

The graphical representation of the signal is very important to assist and inform the patient in real time. In order to draw the correct points on the graphic, the calculation of the bottom line of the graphic, based on screen resolution and text occupancy was performed. A transformation function is used to map the 4096 values on a zero to one scale. The normalized value is multiplied by the bottom position of the chart, and a dot represents the current normalized value. This representation is totally dynamic

and will adjust itself to the situation presented adjusting the YY axis according to the temperature value. By default, the graphic will have a range of valid temperatures to display but it will automatically adjust to a new scenario if the data received demands. In a given scenario, if the range of temperatures goes from 35,2°C to 37,6°C and, if the temperature at some point increase to 37,6°C, the graphic automatically adjusts itself to represent all the data and change the range of temperature to 1°C above the maximum value. The new range will be 35,2°C to 38,6°C.

4.5 Classes and functionalities

The API development process has divided in two main phases: communication and data presentation. Each phase has its own set of classes that have individual and specific objectives. The data presentation block includes several classes where the most significant are the following: Shimmer; Menus; Communication; Maths; Drawing; DataAlert; InputDevice and Threading. Following a brief explanation of each class:

Shimmer class is responsible for all the program initializations and pre configurations, while the **Menus** class is responsible to handle all screen configurations and layouts.

The **Communication** class is responsible for all the pre and post configurations regarding all communications, like bluetooth address and port configuration. **Maths** class is used each time some mathematical operation is needed, mainly to process the incoming data and adjust the scale of graphical data representation.

The **Drawing** class is responsible for converting the received data, in bytes, to graphical presentation. **Threading** class is a key class because is used on every communication and supports all the data transfer between the mobile device and the sink node. The **InputDevices** class handles the input actions into mobile device display.

Finally, **DataAlert** class is the most technology advanced of all classes because of its inherent characteristics and goals. It is responsible for data analysis, data conversion, and can do an emergency call, if some threshold is reached. Our goal was to release an application, the most robust as possible, so all the data received must be precisely analyzed looking for errors. Every time the tool receive data from the sink it will make some tests in order to verify its authenticity, e. g. comparison of bytes read to bytes

that we should read, or order of the received bytes.

Another great feature of this class is the ability to apply artificial intelligence and turn it self sustained, helping a monitored person on a given situation (can be configured according to the case study). On this approach, we are monitoring human body temperature and if in a given execution point the temperature reaching a high value (configured threshold value), the application will make a phone call to an emergency phone number with a predetermined voice message.

In order to achieve bluetooth connectivity and communication in Android OS 1.5 the classes used are the following: `BtDevice` (default bluetooth configurations); `BtException` (exception class for bluetooth); `BtSocket` (sockets related class); `LocalBtDevice` (default configurations for the mobile device); `LocalBtDeviceListener` (event listener for local device); `RemoteBtDevice` (default configurations for remote device); and `RemoteBtDeviceListener`

4.6 Custom exceptions

Exception handling is a software mechanism to handle the occurrence of a exception, in other words it handles special conditions that change the standard flow of a program execution. Since the API developed has some particular characteristics and execution process some custom exceptions were created in order to sustain the normal execution flow.

To increase the robustness of the API several exceptions can be thrown in execution time. Each exception will give specific information of the error that has occurred. Since the communication is bluetooth based, the system wanted to give it the most secure, precise, and robust system as possible. Regarding the communication, exception 1 occurs when the mobile device is not paired with the sink node, because this is a necessary condition to use a bluetooth connection. Exception 2 appear when the device cannot find or pair with the sink node, while exception 3 happens when the sink node is already in acquisition mode, transmitting data to the mobile device and user tries to start it again. Exception 4, takes place when the mobile tool tries to receive data from the sink node and it is not on acquisition mode. Exception 5, occurs when communication is tried when the bluetooth port is closed, while exception 6, is thrown

when a communication is in progress and the mobile tool tries to close it. Exception 7, happens when bluetooth connection is interrupted, or sink node battery is low / exhausted. Finally, exception 8, is a special case since it has special responsibilities and actions. It occurs when the data received is or may be corrupted. Since all data gathered is scanned before presentation, it turns this exception a passive one. It will not stop the program but in its place, it will inform the user about some potential corrupted data, storing it for analysis and let the program keep running.

Trough this set of characteristics and exceptions the API is considered robust and reliable to be implemented, tested and used in order to provide the user a correct, precise and in real time information about his / her vital signs.

4.7 Execution process

The mobile API starts at the *Shimmer* class and initializes the default configurations (display orientation, application name, fonts settings, screen size detection), it goes to the *Menus* class and offer the user the first logistic choice (change default initialized values, communication configurations). The *Communication* class is responsible for all the pre and post configurations regarding all communications, like bluetooth address and port configuration. It will search for bluetooth enabled devices and if a capable device is found the user can connect to it through a bluetooth connection. That connection and the inherent transferred data is supported by a communication thread using the *Threading* class. After this initial step, the user has two main options to choose from: Start real time acquisition data or get the data stored on the SHIMMER SD. If the user selects the SD option, he will get all the values saved on it and they are graphically displayed on the mobile device if a real time monitoring was enabled. Otherwise, if the real data acquisition was chosen the acquisition, analysis and presentation of data in real time starts.

Drawing class allocates space for data that can be received from the SHIMMER. Data is received in a cyclic form: read the next stream; perform all the validation algorithms; present data on the display. Meanwhile, class *Threading* is instantiated every time a new sequence of streams starts or a new set of points are intended to draw on the screen, the frequency of streams is user adjustable. At each new stream sequence the API uses two classes for validation and action purposes. The *Maths*

class is responsible for all the data conversions, algorithms and validation. Finally, the *DataAlert* class scans all the data and acts accordingly the same, it has the capacity to alert the user if an abnormal value or detected or even contact some medical facility in an emergency situation.

Monitoring functionality can be started on any of compatible device, but only a single sink device can be monitored at a given time. The class *InputDevice* is a specific Python class and is used due to the fact that the Nokia phones used do not have touchscreen. Thus, it handles all the input actions on the physical keyboard of the mobile phone.

Chapter 5

Symbian OS Application

Python is a high level programming language that lets you work more quickly and integrate systems more effectively, whose design philosophy emphasizes code readability [58]. It is free to use, even for commercial products and runs on Windows, Linux, Mac OS X, and has been ported to the Java and .NET virtual machines. Python aims to combine a reliable language with very clear syntax and its standard library is large and comprehensive. Its use of indentation for block delimiters is unusual among popular programming languages.

Python supports multiple programming paradigms such as: object oriented; imperative programming, functional programming. It features a fully dynamic type system and automatic memory management [59]. Like other dynamic languages, Python is often used as a scripting language, but is also used in a wide range of non-scripting contexts.

Next is presented a list of side-by-side comparisons of features of Java and Python. In Java, all variable names and their types must be explicitly declared. Attempting to assign an object of the wrong type to a variable name triggers a type exception. So, Java is a statically typed language. Python for the other side is a Dynamic typed because the the programmer never declare anything. An assignment statement binds a name to an object, and the object can be of any type. If a name is assigned to an object of one type, it may later be assigned to an object of a different type. Java language is verbose: Abounding in words; using or containing more words than are necessary. On the other hand Python is Concise: write much functionalities in a few lines of code. Implies clean-cut brevity, attained by excision of the superfluous. Java is not a compact

language while Python is compact.

Looking at these comparisons, the conclusions dictate why Python can be written much more quickly, and maintained much more easily, than Java. The list is not exhaustive comparison, it is meant to be representative only.

In terms of programming language, Python was the primary choice for developing this API instead of pure J2ME because of its inherent characteristics [60]. Nokia maintain an interpreter for the Python programming language that is compatible with their 60 Series (S60) OS. The interpreter is signed by Nokia and therefore can be installed on any S60 compatible device. This is the base requirement to allow this API to run on a mobile device. It allows the creation of Python applications that do not need to be signed, as they are not directly interacting with the OS. Python applications have more control over the device that applications running in the J2ME. For instance, using the S60 Python application-programming interface API it is trivial for a script to do a phone call.

The biggest penalty for using Python applications on a S60 device is the relative awkwardness involved in launching the Python interpreter and then directing it to the script to be executed. However, the detailed call trace provided by the interpreter in the event of an application fault makes it a very useful platform for development.

Python is a dynamically typed object oriented interpreted language. Applications written in Python may access the functionality of the underlying platform through exposed APIs. For S60 applications there are APIs for graphics, telephony, socket communications, text to speech functionalities, audio recording and playback, etc. Being an interpreted language the data processing performance of Python is less than a native binary application and somewhat less than an application running within a virtual machine; however, the ease of development and detailed crash information make it an ideal choice for prototyping applications.

5.1 Symbian User interface

The user interface was designed to be easy to use and show the needed information in a good meaningful way. Figure 5.1 presents the initial screen displayed to the user.

Here the user can modify the default application settings (screen size, orientation) or can start communicating with the SHIMMER.

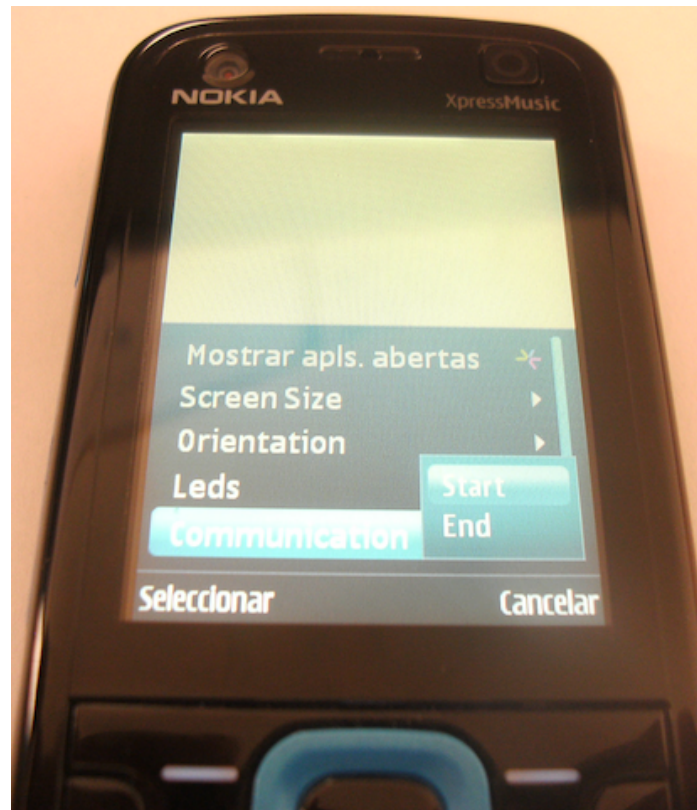


Figure 5.1: Symbian User Interface.

Figure 5.2 presents a screenshot of the instantaneous monitoring signal value. In terms of the mobile tool functionalities, the following two main areas are presented: a text information area and a graphical signal representation.

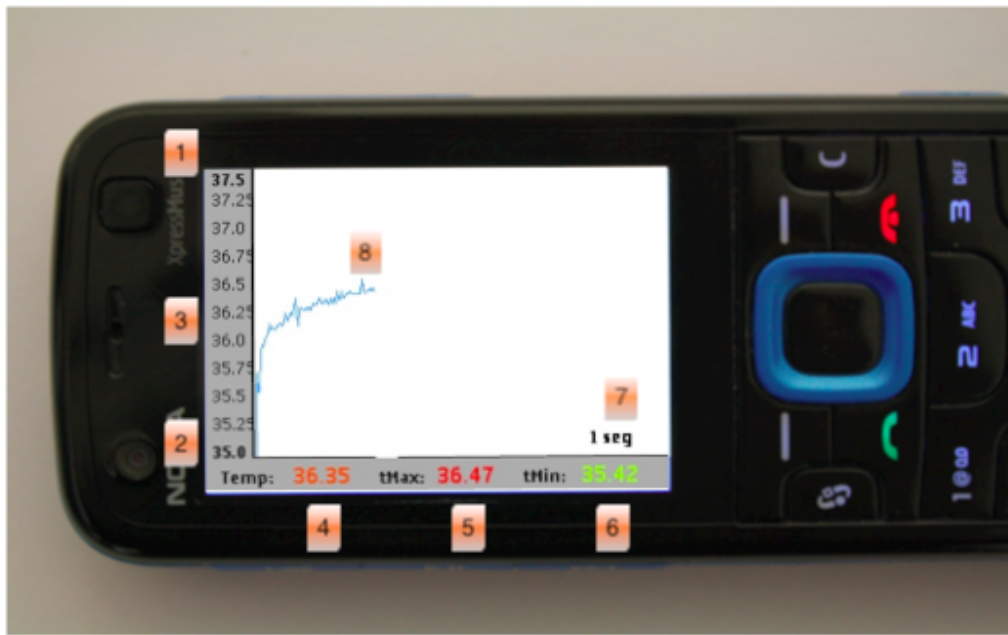


Figure 5.2: Symbian Data Presentation.

Following the figure 5.2, "1" represents the maximum absolute temperature value, while "2" presents the minimum absolute value. Label "3" presents the range values between the maximum and minimum values. Next, "4" depicts the instant temperature value. Number "5" shows the maximum relative value of the temperature that current session has acquired, while number "6" presents the minimum relative temperature value. The number shown in "7" informs about the current data frequency collection. The data can be acquired in several frequency values, ranged from 1 second to 10 minutes (offering the following ranges: 1 second, 30 seconds, 1 min., 5 min., and 10 min.). Finally, the area identified with "8" appears shows the graphical representation of data. When the signal reaches the end of the screen the application does not erase all the signal drawing, but it keeps all the collected data (historic) for further analysis. Such feature provides a better analysis for the user.

5.2 Deploy, Tests and Validation

The mobile tool was deployed, tested, and evaluated on several different devices, such as the Nokia 5320 Express Music (Java-enabled phone, S60 Symbian OS device version 9.3), the Nokia 5800 Express Music (Java-enabled phone, S60 Symbian OS device

version 9.4), the Nokia 6630 (Java-enabled phone, series 60 Symbian OS device version 8.0), and finally, the Samsung SGH-G810 (Java-enabled phone, series 60 3rd edition, Symbian OS device version 9.2). The application ran effortlessly on every one without any problems.

Table 5.1 summarizes the results of a performance comparison between the evaluated devices in four different approaches. The first column presents the startup time (in seconds) that the mobile device takes to launch the application. The second column has the connection time (in seconds) between the mobile device and the sink. The third column presents the time (in seconds) taken between the initial communication and data presentation. At this time, the tool starts all the initial data processing. These two times were measured based on an average of thirty experiments. Finally the last column indicates the percentage of corrupted or malformed data. Taking in the account that on a interval of three hundred seconds an average of six streams are not fully correct, let us assume that the overall results are very acceptable. In terms of smoothness the application seems very smoothness in any tested device. In a range of 1% to 100%, being the 1% the worst performance value and 100% the best one the application can be classified as 90%. These tests take several parameters in consideration, such as smoothness between menus and graphical interface, temperature signal update, label update, and graphic scaling. Since multithread is software implemented and not a device feature, it is available on all devices, providing the user the capability to make phone calls, text messages, web browsing, email or even play games, while the data is continually acquired.

Table 5.1: Symbian Benchmark Tests.

Device	Init. time	Connection time	Data presentation	Corrupted data
5320	1.3s	2.1s	1.5s	3.33%
5800	1.1s	2s	1s	3.67%
6630	1s	2s	1s	2.33%
SGH-G810	1.1s	2.2s	1.5s	2.67%

In terms of battery power consumption, the application consumes the same when compared to a scenario where a standard bluetooth connection is used (phone data transfer). However, the constant use of screen backlight can pose significant battery

drain (a scenario similar to the use of a screen saver).

The mobile application uses approximately 54KB of persistent storage space and 4KB of RAM. It is a small and fast application but a very robust one. It normally takes around 2 seconds to connect to the sink node and one more second to start gathering, analyzing, and presenting data from sink in a correct way.

The current application has the particularity of storing all the incoming data in a easy and common file format for posterior analysis and inspection. The file type chosen was the CSV due to its inherent characteristics. Through this file format it is possible to import it to an external program such as Microsoft Excel or other that support data analysis. The file architecture is composed by three columns, the first one represent the number of the current stream, the second represents the temperature value, finally the third represent the data and time of the current received value.

In order to guarantee that mobile tool performs exactly what is supposed to do, its results were validated against a regular desktop application and, as expected, the results were exactly the same, since the data source is the same. Then, it confirms that mobile application is running properly. The main advantage here is the portability offered by such tool and offers the patient an easy way to view the monitored signals anywhere she wants.

In order to develop this API implementation the PyS60 was used. It comes with all the interfaces, tools and resources needed to develop for Symbian devices. It is open source, under Apache 2 and Python licenses. Some of the other key components used were the following: TextMate (an text editor for Mac OS X); PyS60 (a framework that provides access to many of the phone uniquely functions);

Chapter 6

Windows Mobile Application

Windows Mobile is the Microsoft platform for mobile devices based on Windows CE version 5.0, and it is used in a wide variety of third party hardware, such as PDA and smart-phones. It is worth mentioning that Windows CE [61], is considered as a modular OS that serves as the foundation of several classes of devices, which can be referred to as embedded devices. It is supported by a wide variety of processors such as: Intel, MIPS, ARM, and Hitachi SH. It is optimized for devices which have minimal storage and a small scale factor, so that the kernel can be contained in less than a megabyte of memory. Devices are often configured without any disk storage, and may be configured as closed systems, with the OS burned on a flash Read Only Memory (ROM). Windows CE is compliant with the definition of a real time OS with a deterministic interrupt latency. Furthermore, it is a multitasking OS, where the fundamental unit of execution is the thread. Since the first edition in 1996 with the name of Pegasus, Windows CE has evolved in many different platforms, not only those belonging to the sphere of handheld devices.

The Windows CE kernel uses a paged virtual memory system to manage and allocate program memory. The virtual memory system provides contiguous blocks of memory, between 1 and 4 Kbytes, within 64 Kbytes regions, so that applications do not have to deal with memory allocation [62]. In any windows CE based device, the OS is stored entirely in the ROM memory, as well as the applications that come with the OS. The whole OS is mapped in a binary ROM image which is divided logically into two kinds of modules. The former case implies that the OS executes the ROM-based modules in place, in which case modules are also referred to as executable in place (XIP) modules. This is a technique which can be used to save space in Random Access Memory (RAM)

and to reduce the time needed to start an application. The latter case implies that the OS decompresses the module and pages it into RAM.

The RAM on a such devices is divided into two logically separated areas which are the object store and the program memory [63]. The object store resembles a permanent, virtual RAM disk. The data in the object store is retained when you suspend or perform a soft reset operation on the system. Normally, devices have a backup power supply for the RAM to preserve data if the main power supply is interrupted temporarily. When operations resume, the system looks for a previously created object store in RAM and uses it, if one is found. Devices that do not have battery-backed RAM can use a particular flag in the registry to preserve data during multiple boot processes. The program memory, instead consists of the remaining RAM. It stores heaps and stacks for the applications that are running on the system.

6.1 Windows Mobile application user interface

The Windows mobile interface provided by Microsoft is in most cases very simple and intuitive because people are used to Microsoft Windows OS and its similarities. So the user interface developed for this application follow those characteristics. The user interface was designed to be easy to use and show the needed information in a good meaningful way. Figure 6.1 presents the initial screen displayed to the user. Here the user can be informed by all the events processed by the mobile device because this screen acts as a debug showing information about connectivity, data processing and errors.

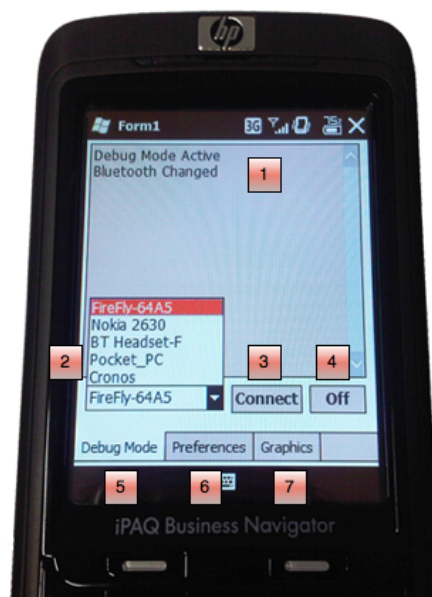


Figure 6.1: Windows Mobile debug window.

Following the Figure 6.1 , at the bottom of the screen three main tabs can be viewed (Debug mode, Preferences, Graphics). Each one represents a unique user interface with special and unique characteristics. On the center of the screen label "1", represents the text area where all the information is presented and each new information represents a new line entry on the text area. Such entry are presented in real time to the user and can be the widest such as remote device, connection and data information. Label "2", represents the list of remote Bluetooth devices found that are able to accept connection from the mobile device. Label "3", represents the connection button and it serves to start the direct communication with the selected bluetooth device from the label "2". Label "4" button will stop all the connectivity between the mobile device and the bluetooth device at a given moment. Label "5", represents the actual screen, a debug menu for all the text information for the user. Label "6" represents the preferences menu, while "7" the graphical data presentation.

In order to change the default application and connection configurations a preferences menu was created. This menu shown at Figure 6.2 provides the user the capacity to do the following: alternate the Bluetooth stack; connect and pair the local device; data acquisition mode; fsfrequency and Leds.



Figure 6.2: Windows Mobile preferences windows.

Following the Figure 6.2, "1" represents the local bluetooth stack. The user can change the bluetooth stack directly through this shortcut instead of the default windows mobile OS menu. Label "2" represents the connection and pairing activity. The user uses this menu to pair and connect both mobile device and the SHIMMER. Label "3" represents the data acquisition mode. The data transfer between both devices can only be active after this option is selected. The option shown in "4" informs about the current data frequency collection. The data can be acquired in several frequency values, ranged from 1 second to 10 minutes (offering the following ranges: 1 second, 30 seconds, 1 min., 5 min., and 10 min.). Finally at "5" the user can swap the communication leds presented at the SHIMMER device.

Figure 6.3 presents a screenshot of menu regarding the instantaneous monitoring signal value. In terms of the mobile tool functionalities, the following two main areas are presented: a text information area and a graphical signal representation. This screen is the principal one and is used to inform the user in a easy and clean way and in real time the graphical representation of the data gathered.

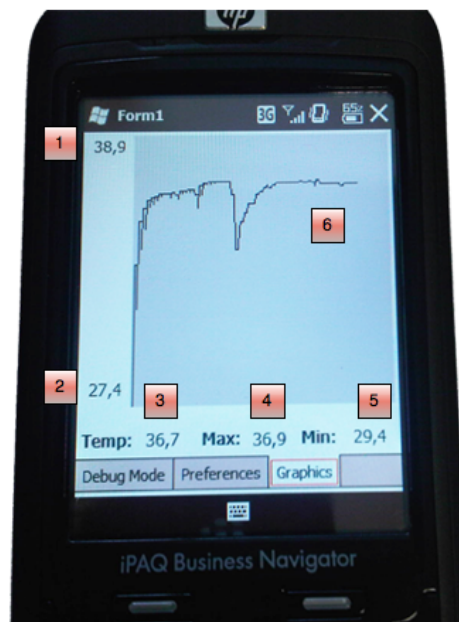


Figure 6.3: Windows Mobile real time graphical data presentation.

Following the figure 6.3, "1" represents the maximum absolute temperature value, while "2" presents the minimum absolute value. Next, "3" depicts the instant temperature value. Number "4" shows the maximum relative value of the temperature that current session has acquired, while number "5" presents the minimum relative temperature value. Finally, the area identified with "6" appears shows the graphical representation of data. When the signal reaches the end of the screen the application does not erase all the signal drawing, but it keeps all the collected data (historic) for further analysis. Such feature provides a better analysis for the user.

6.2 Deploy, tests and validation

The mobile tool was deployed, tested, and evaluated on several different devices, such as the HP iPAQ Business Navigator (Windows Mobile 6.5), the HTC HD (Windows Mobile 6) and the Asus M539W (Windows Mobile 6). The application ran effortlessly on every one without any problems.

Table 6.1 summarizes the results of a performance comparison between the evaluated devices in four different approaches. The first column presents the startup time (in seconds) that the mobile device takes to launch the application. The second column

has the connection time (in seconds) between the mobile device and the sink. The third column presents the time (in seconds) taken between the initial communication and data presentation. At this time, the tool starts all the initial data processing. These two times were measured based on an average of thirty experiments. Finally the last column indicates the percentage of corrupted or malformed data. Taking in the account that on a interval of three hundred seconds an average of six streams are not fully correct let us assume that the overall results are very acceptable.

Table 6.1: Windows Mobile Benchmark Tests.

Device	Init. time	Connection time	Data presentation	Corrupted data
HP iPAQ	6s	2.2s	1.5s	2.66%
HTC HD	3.2s	2s	1s	1.76%
Asus M539W	5.1s	2.1s	1.5s	2.33%

In terms of smoothness the application seems very smoothness at all tested devices. In a range of 1% to 100%, being the 1% the worst performance value and 100% the following results were achieved. The HP iPaq mobile device was the slowest track of the Asus M539W. Those devices had several difficulties not only on smoothness test but also on application initialization and data presentation. Were both classified as 75%. The HTC HD with his faster processor runs the application smoother and performed all the tests without major problems and has the classification of 90%. These tests take several parameters in consideration, such as smoothness between menus and graphical interface, temperature signal update, label update, and graphic scaling.

Since that recent mobile devices are primarily used to make phone calls, text messages, calendar events, emails and web browsing the system was designed to cooperate with them without disturb each others. When any of this functionalities are under use, the application is prepared to run on the background while a person uses the mobile device without issues. On all evaluated devices were performed several tests such as: phone calls; text message exchange, web browsing and, as expected, the data continuously flowed from the SHIMMER to the application without any issue.

Since multithread is software implemented and not a device feature, it is available on all devices, providing the user the capability to make phone calls, text messages,

web browsing, email or even play games, while the data is continually acquired.

In terms of battery power consumption, the mobile device consumes almost the same when compared to a scenario where a standard bluetooth connection is used. In several tests carried out the battery drained an average of 4% to 6% when using the application.

The application uses 19KB of persistent storage space, 320KB of cache memory (for preference settings), 1.87MB of RAM, and 4% of CPU usage (average value for thirty measurements in a time interval of 300 seconds).

The current application has the particularity of storing all the incoming data in a easy and common file format for posterior analysis and inspection. The file type chosen was the CSV due to its inherent characteristics. Through this file format it is possible to import it to an external program such as Microsoft Excel or other that support data analysis. The file architecture is composed by three columns, the first one represent the number of the current stream, the second represents the temperature value, finally the third represent the data and time of the current received value.

Once again, to guarantee that mobile tool performs exactly what is supposed to do, its results were validated against a regular desktop application and, as expected, the results were exactly the same, since the data source is the same. Then, it confirms that mobile application is running properly. The main advantage here is the portability offered by such tool and offers the patient an easy way to view the monitored signals anywhere she wants.

In order to develop this API implementation specific tools were used. The Windows Mobile SDK comes with all the interfaces, tools, and resources needed to develop windows mobile applications. Trough those specific tools Microsoft delivers the necessary frameworks, technologies and libraries for the correct mobile applications development.

Some of the other key components used were the following: Visual Studio 2008 (an IDE that manages the application projects and lets the developer edit, compile, run, and debug the code; Windows Mobile emulator (a Windows application that simulates

a windows mobile phone technology stack, allowing the developer to test applications locally on the computer).

6.3 Windows mobile emulator

Microsoft have a standalone version of the windows mobile emulator that is included with their development products. It is desktop software that imitates windows mobile 5.0 and 6 on the PC and permits the installation and uninstalls of software, themes or interfaces. This tool is ideal for developers who need to support or evaluate windows mobile devices applications. The emulator can be connected to external devices, support stress tests, among others.

Most windows mobile emulators are coming with a bluetooth connection support, although it has some limitations. It only can be used to connect to other computers, such as laptop or PC. Before making tests in real environment is a good option to always use the emulator for such testing. Since the emulator does not support direct communication with external devices with Radio Frequency Communication (RFCOMM) and Serial Port Profile (SPP) an auxiliary step was implemented.

In order to activate the Bluetooth communication from and to the windows mobile emulator some specific drivers were installed [64]. Those drivers activate the following components on the emulator: Bluetooth HCI Transport Driver; serial port driver to abstract the transport driver from knowing means of communication; Microsoft remote tools framework remote agent. After the successful installation the windows mobile emulator is ready for bluetooth device discovery and communication. The figure 6.4 represents a screenshot of the windows emulator running the windows mobile API.

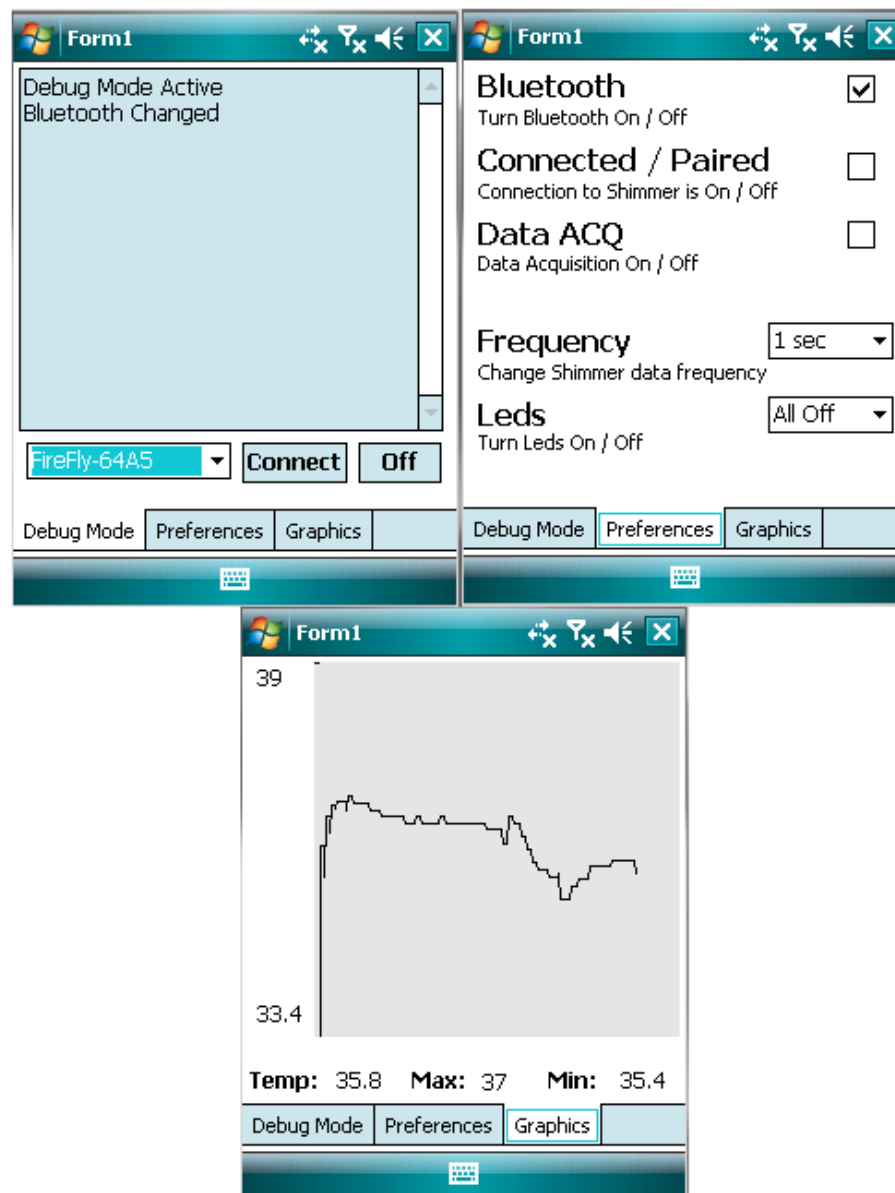


Figure 6.4: Windows mobile emulator with bluetooth support.

Following Figure 6.4 and as previously explained the API is composed by three screens. Is clearly evident that the emulator caught a bluetooth device (SHIMMER device) and is communicating with it and displaying data in real time.

Chapter 7

Android OS Application

Android OS is a handheld OS based on the Linux Kernel [65]. It was originally created by Android Inc., a company purchased by Google later on. It allows writing code in Java language and interacts with the device via Google specific Java libraries. Android OS delivers a complete set of software for mobile devices, namely, an operating system, middleware functionalities, and basic mobile applications, such as, text messaging or phone calls. The Android SDK provides the API necessary to begin creating applications on the Android platform.

The Android OS has four main paradigms [66]. The first one is related with the whole Android world and it must be Open Source. The second regards the creation of new applications and all of them must be created equal. The third paradigm is the applications boundaries. These applications must be rich and innovative supporting all kinds of information provided by Web services, other phones or even ubiquitous systems. The last one reflects the speed of the application development and deploying.

Based on Linux Kernel, Android was built from scratch in order to allow developers to create applications that take full advantage of all characteristics of the handset. For example, an application can call upon any core phone functionality such as making calls, text messaging or using the camera. Through this characteristic the developers can aim to better, cohesive and rich applications. The authors had used this characteristic to operate the telephony functionality and invoke a call in an emergency scenario. Using the Linux Kernel as its foundation, Android is capable of adapting to any new emergent technology as soon it goes out, plus it can have millions of people working to improve base functionalities.

Unlike all the predecessors mobile OS [67] (Symbian OS, RIM Blackberry OS, iPhone OS, Windows Mobile, Palm webOS, Maemo), Android does not differentiate between the phone core applications (default text messaging or phone calls) and new user developed applications. All of them can be built to have equal access to a given phone characteristics providing to the user the capability of using its phone to their own and unique interests and definitions. For example, the proposed tool is fully integrated in the OS as if it was a standard application.

Android provides the tools and API's to create innovative and useful applications using data from the phone device and all over the Internet. Android includes a set of tools that have been built from scratch alongside the platform in order to provide the developers the highest productivity and deep insight into their applications. The tool presented in this paper was created over the Eclipse plug-in provided by Android,. This plug-in enables an easier and faster way to create Android projects and also improves the communication and deploy of the application to the Android simulator or to a real device.

7.1 Android System Architecture

The Android architecture is complex and it is divided in the following four layers, from bottom up to top: Linux Kernel, Libraries, Application Framework, and Applications [68]. Each layer has its own responsibilities and characteristics. The upper layer presents all the applications that run on the phone device such as text message client, Internet browser, contacts, maps, calendar, and all the programs written by developers.

The second layer is the Application Framework. Android provides an open development platform, offering the ability to create extremely rich applications where developers can take advantage of the device hardware. Programmers have full access to the same framework APIs used by the core applications. Any application architecture is designed to simplify the reuse of components, it can offer its characteristics, and any other application may use them to its own goal. The Application Framework layer is responsible for these Android specific characteristics. It is compounded by several components, such as, Activity Manager, Window Manager, Content Provider, View System, Package Manager, Telephony Manager, Resource Manager, Location Manager, and Notification Manager.

Implicit in applications there are a set of services and systems, including some or all of the above-mentioned components [69]. An extensive set of Views can be used to create a given application, including text boxes, lists, buttons, images, and so on. An application that needs to access data from another application must have the Content Provider component active. The Content Provider stores and retrieves data, and makes it accessible to all applications. Android has several content providers for common data, such as, audio, video, and contacts, among others. Resource Manager provides access to external resources that are only used by the developer code and compiled into the application at the creation time. This data can be located outside the initial package and it can include strings, graphics, layout files, videos or any other external supported data type. The Notification Manager is responsible for enabling all applications to display custom information on status bar, such as, alerts and alarms. The Activity Manager handles the life cycle of applications and provides a common OS navigation through the application data flow. The Window Manager is responsible for all the windows layout and interaction between views. Telephony Manager, as the name suggests, administers all the call functions. Any application can access to contacts and make a call. Location Manager provides access to the system location services and APIs. These services allow applications to obtain periodic updates of any devices geographical location. Package Manager is in charged for retrieving several kinds of information related to the application packages that are currently installed on the device.

Libraries and Android Runtime compound the third layer. The libraries have several components, including Surface Manager, Media Framework, SQLite, OpenGL, FreeType, WebKit, SGL, SSL, and libc. All these components are the core mechanisms used to make all the Android OS and applications running properly. The Android Runtime has its core Libraries and the Dalvik Virtual Machine. The Dalvik Virtual Machine is a custom virtual machine which runs the Java platform on any Android supported mobile device. The virtual machine runs applications, which have been converted into a Dalvik executable format. The Dalvik machine is suitable for systems that are constrained in terms of memory and processor speed.

The lower layer is the Linux Kernel and it is the base for all Android specifications and characteristics. This layer behaves as a normal Linux Kernel with the difference that is tuned for mobile devices. The kernel has several modules and each one is adapted to a specific goal. The modules available at this layer are the following:

Flash Memory, Power Management, Display, Camera, Binder, Keypad, Wi-Fi, and Audio Drivers.

7.2 Activity Lifetime

Activity is a single functionality that a user can perform [70] and almost all activities interact with the user. In the system, activities are managed as a task stack. Every time a new activity is created, it is placed on the top of the stack and becomes the running activity. The previous activity remains below, in the stack, and it does not come to foreground again until the newest activity appears. The activity has four main states (active, paused, stopped, and dropped). If an activity is in screen foreground, it is active and if it lost focus but still visible, it is paused. If an activity is completely obscured by another activity, it is stopped, and, finally, if an activity is paused or stopped, it can be dropped from the system memory.

There are three main loops used for monitoring within an activity, the entire lifetime, the visible lifetime, and the foreground lifetime. Entire lifetime has two methods, the `onCreate(Bundle)` and `onDestroy()`. The `onCreate(Bundle)` is called when the activity is created for the first time, while the `onDestroy()` is the last call received by the activity when it is destroyed. The visible lifetime includes `onStart()` and `onStop()` methods. The `onStart()` method is called when the activity is becoming visible to the user, while the `onStop()` is called when the activity is no longer visible to the user. The foreground lifetime happens between a call to `onResume()` and `onPause()`. The `onResume()` method is called when the activity will start interacting with the user and the `onPause()` is called when the system is about to start resuming the previous activity.

7.3 Android application user interface

Since Android OS has several firmware and kernel versions, figure 7.1 presents default information about the mobile device used. Because they were developed several API the user interface similarity had to be considered in order to achieve full unification for both specification as user interface organization.



Figure 7.1: Android OS Version.

Following the figure, "1" represents the mobile device version, a HTC Magic, while "2" presents firmware version. Label "3" presents the kernel version compiled and used. Finally, "4" depicts the compilation number inherent to the kernel version.

The user interface, was designed to be easy to use and show the needed information in a good meaningful way. Because there were developed several API, the user interface similarity has to be considered in order to achieve full unification for both specification as user interface organization. Figure 7.2 presents the initial screen displayed to the user. This screen is used to display real time information to the user, such as remote device, connection and data information. If at a given time the device finds possible corrupted data that information is show at this screen.



Figure 7.2: Android application debug window.

Following the figure 7.2 , at the top of the screen three main menus can be viewed. Each one represents a unique user interface with special and unique characteristics. Label "1", represents the actual screen, a debug menu for all the text information for the user. Label "2" represents the preferences menu, while "3" the graphical data presentation. On the middle of the screen, "4" indicates where the text information is shown. On this screen some information is shown such as: local bluetooth initialization success; connection to remote device success; collection data started; information about the local Android OS unique stats.

In order to change the default application and connection configurations a preferences menu was created. This menu shown at Figure 7.3 provides the user the capacity to do the following: Alternate the bluetooth stack; Connect and pair the local device; Data acquisition mode; Frequency and Leds.



Figure 7.3: Android application preferences window.

Following the figure 7.3, "1" represents the local bluetooth stack. The user can change the bluetooth stack directly through this menu instead of the default Android OS menu. Label "2" represents the connection and paired activity. The user use this menu to pair and connect both mobile device and the SHIMMER. Label "3" represents the data acquisition mode. The data transfer between both devices can only be active after this option is selected. The option shown in "4" informs about the current data frequency collection. The data can be acquired in several frequency values, ranged from 1 second to 10 minutes (offering the following ranges: 1 second, 30 seconds, 1 min., 5 min., and 10 min.). Finally at "5" the user can swap the communication leds presented at the SHIMMER device.

Figure 7.4 presents a screenshot of the instantaneous monitoring signal value. In terms of the mobile tool functionalities, the following two main areas are presented: a text information area and a graphical signal representation. This screen is the principal one and is used to inform the user in a easy and clean way and in real time the graphical

representation of the data gathered.

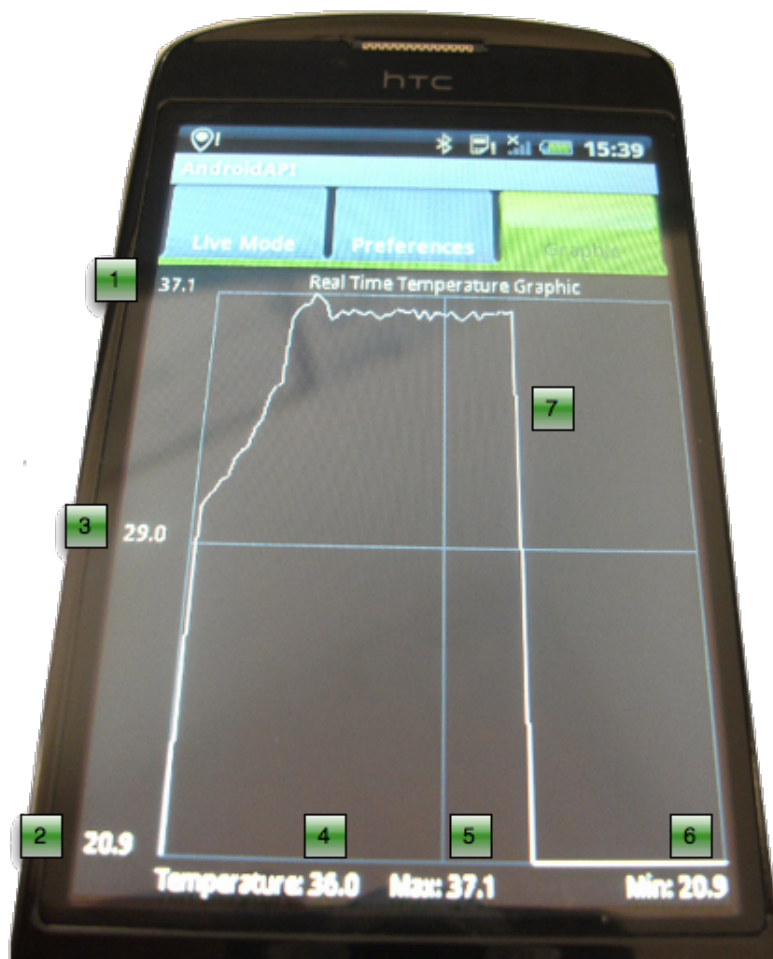


Figure 7.4: Android application real time graphical data presentation.

Following the figure 7.4, "1" represents the maximum absolute temperature value, while "2" presents the minimum absolute value. Label "3" presents the range values between the maximum and minimum values. Next, "4" depicts the instant temperature value. Number "5" shows the maximum relative value of the temperature that current session has acquired, while number "6" presents the minimum relative temperature value. Finally the area identified with "7" appears and shows the graphical representation of data. When the signal reaches the end of the screen, the application does not erase all the signal drawing, but it keeps all the collected data (historic) for further analysis. Such feature provides a better analysis for the user.

7.4 Deploy, tests and validation

The mobile tool was deployed, tested, and evaluated on the HTC Magic mobile device. This device has the Android OS version 1.5 installed by default. The application ran effortlessly on HTC Magic without any problems.

Table 7.1 summarizes the results of a performance tests evaluated on the HTC Magic device in four different approaches. The first column presents the startup time (in seconds) that the mobile device takes to launch the application. The second column has the connection time (in seconds) between the mobile device and the sink. The third column presents the time (in seconds) taken between the initial communication and data presentation. At this time, the tool starts all the initial data processing. These two times were measured based on an average of thirty experiments. Finally the last column indicates the percentage of corrupted or malformed data. Taking in the account that on a interval of three hundred seconds an average of six streams are not fully correct let us assume that the overall results are very acceptable.

Table 7.1: Android Benchmark Tests.

Device	Init. time	Connection time	Data presentation	Corrupted data
HTC Magic	1.2s	2s	1.2s	5.67%

In terms of smoothness the application seems very smoothness. In a range of 1% to 100%, being the 1% the worst performance value and 100% the best one the application can be classified as 90%. These tests take several parameters in consideration, such as smoothness between menus and graphical interface, temperature signal update, label update, and graphic scaling.

Since multithread is software implemented and not a device feature, it is available on all devices, providing the user the capability to make phone calls, text messages, web browsing, email or even play games, while the data is continually acquired.

In terms of battery power consumption, the application consumes the same when compared to a scenario where a standard bluetooth connection is used (phone data transfer). However, the constant use of screen backlight can pose significant battery

drain (a scenario similar to the use of a screen saver).

The current application has the particularity of storing all the incoming data in a easy and common file format for posterior analysis and inspection. The file type chosen was the CSV due to its inherent characteristics. Through this file format it is possible to import it to an external program such as Microsoft Excel or other that support data analysis. The file architecture is composed by three columns, the first one represent the number of the current stream, the second represents the temperature value, finally the third represent the data and time of the current received value.

Once again, to guarantee that mobile tool performs exactly what is supposed to do, its results were validated against a regular desktop application and, as expected, the results were exactly the same, since the data source is the same. Then, it confirms that mobile application is running properly. The main advantage here is the portability offered by such tool and offers the patient an easy way to view the monitored signals anywhere she wants.

In order to develop the Android OS API a specific SDK was used. The Android SDK comes with all of the interfaces, tools, and resources needed to develop Android applications. Due the fact that Android OS is UNIX based, many of the technologies that form the lower-levels of the OS are derived from open-source technologies. The interfaces for these technologies are therefore available in the standard library and interface directories.

Some of the other key components used were the following: Eclipse (an IDE that manages the application projects and lets the developer edit, compile, run, and debug the code; Android Plugin (a specific plugin for the eclipse IDE that simplifies the deployment process of the application to the mobile device); Android Emulator (a cross platform (Windows, Mac OS X, Linux) application that simulates the Android technology stack, allowing the developer to test Android applications locally on computer).

7.5 Custom firmware 1.5 Bluetooth API

Google introduced official support for bluetooth RFCOMM and SPP through the firmware version 2.0 [71]. Oldest versions such as 1.5 and 1.6 do not provide the users with the inherent technology. The available device for development was the HTC Magic loaded with the firmware version 1.5. Since neither HTC nor Google provides official updates for the version 2.0 the development process was limited to version 1.5, and with such scenario some difficulties were encountered.

In order to circumvent the development difficulties a custom bluetooth API aimed for firmware version 1.5 was built. Such API is limited to firmware version 1.5 since it uses unique methods inherent to that version. Although, it not tested or validated on another firmware version. The following classes were developed in order to develop a custom firmware for version 1.5.

- **BtDevice**: This class is the base class and is used to instantiate the bluetooth device with its default dongle information. It has all the necessary informations about the bluetooth profiles, classes and protocols.
- **LocalBtDevice**: This class is responsible to instantiate the bluetooth device with its name, address, class, state. It is compounded by the necessary methods to get that information.
- **LocalBtDeviceListener**: This class is used as a listener for the local bluetooth device. It has the responsibility to inform when the device do the following: Scan mode; Bluetooth enabled or disabled; Device found;
- **BtException**: This class is compounded by all the exceptions that can be thrown at any given time related with bluetooth activity.
- **BtSocket**: This class is responsible for the data transfer between devices. It has all the methods related with the input and output streams and bluetooth socket creation.
- **RemoteBtDevice**: This class is responsible for acquiring the following information regarding the remote device: Device Address; Device class; Device name; Pairing; Set Pin;

- **RemoteBtDeviceListener**: This class is used as a listener for the remote blue-tooth device. It has the responsibility to inform when the device do the following: Paired; Pin requests;

Trough this custom API the Android version 1.5 has the RFCOMM and SPP enabled and can connect to remote devices using those protocols and ports. Since SHIMMER uses them a successful data transfer and communication between both devices was fully achieved.

Several tests were developed to test the implementation of the API such as: Device scanning; Device pairing; Device communication. Such tests have probed robust enough in order to assert that the custom API is robust, reliable and trustable to the objectives proposed on this dissertation.

Chapter 8

iPhone OS Application

The iPhone, iPod touch and iPad is a line of multimedia enabled mobile devices designed and marketed by Apple. iPhone originally released in 2007 and functions as a camera phone, a media player (equivalent to a video iPod), and an Internet active client (email, web browsing, bluetooth and Wi-Fi connectivity). The user interface is based on a multi-touch screen, including a virtual keyboard rather than a physical one. The iPhone come with several default applications but third-party applications are available from the App Store (virtual store provided by Apple), which launched in mid 2008 and now has well over 230,000 apps. These apps have diverse functionalities, including games, reference, GPS navigation, social networking, and advertising for television shows, films, and celebrities.

iPhone OS is the OS at the heart of iPhone, iPod touch and iPad devices. Although it shares a common heritage and many underlying technologies with Mac OS X, iPhone OS was designed to meet the needs of a mobile environment, where user needs are slightly different then a normal computer. This operating system manages the device hardware and also provides the basic technologies needed to implement native applications on the phone. Depending on whether it is installed on an iPhone, iPod touch or iPad, the operating system also ships with several system applications, such as Phone, Mail, and Safari, that provide standard system services for the user.

The Apple iPhone is a powerful mobile platform with a ARM 11 620 MHz processor and 128 MB DRAM. It has a screen size of 320 x 480 pixels, 3.5 inches and it comes equipped with a built-in microphone, speakers, and 3-axis gyroscopic sensors. The iPhone also has an audio line-out to allow sound to flow through an external device.

8.1 iPhone OS architecture

In iPhone OS, the underlying system architecture, and many of the technologies, are similar to those found in Mac OS X. The kernel in iPhone OS is based on a variant of the same basic Mach kernel that is found in Mac OS X. On top of this kernel are the layers of services that are used to implement applications on the platform. Figure 8.1 shows a high-level overview of these layers.

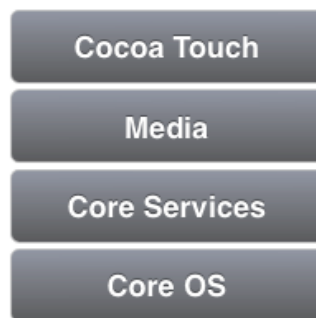


Figure 8.1: iPhone OS Architecture.

The iPhone OS architecture is divided in four layers and each one represent specific technology that is interconnected. Such architecture offers the developer several choices when comes to implementing their code.

The first layer is the *Core OS* and consists of the following components: OS X kernel; Mach 3; BSD; Sockets; Security; Power Management; Keychain; Certificates; File System and Bonjour. The second layer is the *Core Services* and consists of the following components: Collections; Address Book; Networking; File Access; SQLite; Core Location; Net Services; Threading; Preferences and URL utilities.

The *Core OS* and *Core Services* layers contain the fundamental interfaces for iPhone OS, including those used for accessing files, low-level data types, Bonjour services, network sockets, and so on. These interfaces are mostly C-based and include technologies such as Core Foundation, CFNetwork, SQLite, and access to POSIX threads and UNIX sockets among others.

As we move into the upper layers, are found more advanced technologies that use a mixture of C-based and Objective-C based interfaces. For example, the *Media* layer contains the fundamental technologies used to support 2D and 3D drawing, audio, and video. This layer includes the C-based technologies OpenGL ES, Quartz, and

Core Audio. It also contains Core Animation, which is an advanced Objective-C based animation engine. The third layer is the *Media* layer and is composed by the following components: Core Audio; OpenAL; Audio Mixing; Audio Recording; Video Playback; JPG, PNG, JPG File formats; PDF; Quartz (2D); Core Animation and OpenGL ES.

In the *Cocoa Touch* layer, most of the technologies use Objective-C. The frameworks at these layers provide the fundamental infrastructure used by developers on their applications. For example, the Foundation framework provides object-oriented support for collections, file management, network operations, and more. The UIKit framework provides the visual infrastructure for your application, including classes for windows, views, controls, and the controllers that manage those objects. Other frameworks at this level give you access to the users contact and photo information and to the accelerometers and other hardware features of the device.

The fourth layer is the *Cocoa Touch*. The Cocoa Touch layer is subdivided in two middle layers called *UIKIT* and *Foundation*. The UIKIT is composed by the: User Interface elements (Multi-Touch Events and Multi-Touch Controls), Application Runtime, Event Handling and Hardware API (Accelerometer). The Foundation layer is composed by the Utility and Collection classes (Localization and Alerts), Object Wrappers for system services and a subset of foundation in Cocoa.

8.2 iPhone application user interface

The user interface was designed to be easy to use and show the needed information in a good meaningful way. Because they were developed several API the user interface similarity had to be considered in order to achieve full unification for both specification as user interface organization. Figure 8.2 presents the initial screen displayed to the user. This screen is used to display real time information such as bluetooth initialization, loading data, processing and data information. If at a given time the device finds possible corrupted data that information is show in red at this screen.

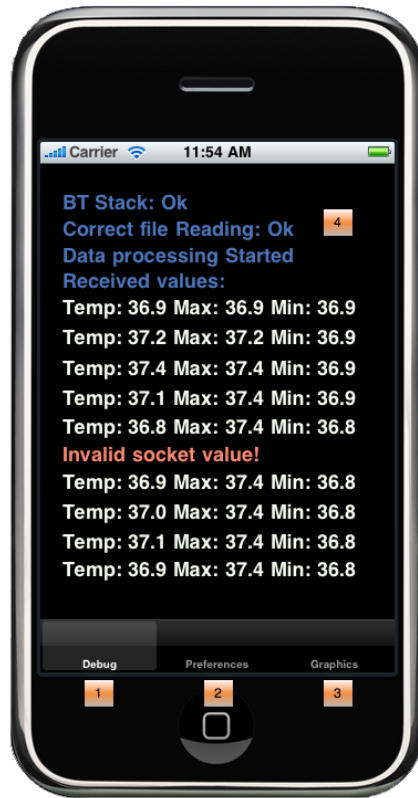


Figure 8.2: iPhone application debug window.

Following the figure 8.2 , at the bottom of the screen three main tabs can be viewed (Debug, Preferences and Graphics). Each one represents a unique user interface with special and unique characteristics. Label "1", represents the actual screen, a debug menu for all the text information for the user. On this image is shown some information regarding the initialization process, the successful load file procedure, and the actual temperature values. Label "2" represents the preferences menu, while "3" the graphical data presentation. On the center of the screen label "4", represents the text area where all the information is presented. Each new information implies a new line on the text area and it is presented in real time to the user.

In order to change the default application configurations a preferences menu was created. This menu shown at figure 8.3 provides the user the capacity to do the following: Alternate the bluetooth stack; Load a new CSV file; Start the graphical presentation; Change the frequency values.

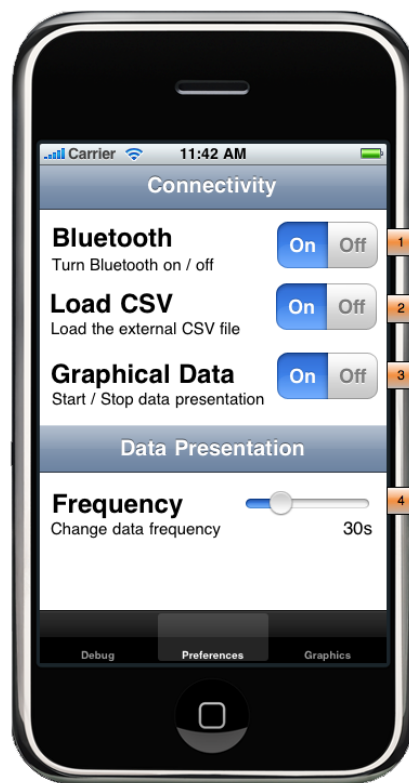


Figure 8.3: iPhone application preferences window.

Following the Figure 8.3, "1" represents the local Bluetooth stack. The user can change the bluetooth stack directly through this shortcut instead of the default windows mobile OS menu. Label "2" represents the menu for load the default CSV file for posterior analysis and graphical presentation. It can only load a file at a time and at the current implementation the file must be placed at a specific directory at the iphone file system. Label "3" represents the graphical data presentation mode. Finally at "4" informs about the current data frequency collection. The data can be acquired in several frequency values, ranged from 1 second to 10 minutes (offering the following ranges: 1 second, 30 seconds, 1 min., 5 min., and 10 min.). Since the CSV file has the information regarding the timestamp of the stream it is possible to calculate the frequency and change it in real time if the user wants it.

Figure 8.4 presents a screenshot of menu regarding the instantaneous monitoring signal value. In terms of the mobile tool functionalities, the following two main areas are presented: a text information area and a graphical signal representation. This

screen is the principal one and is used to inform the user in a easy and clean way and in real time the graphical representation of the data gathered.

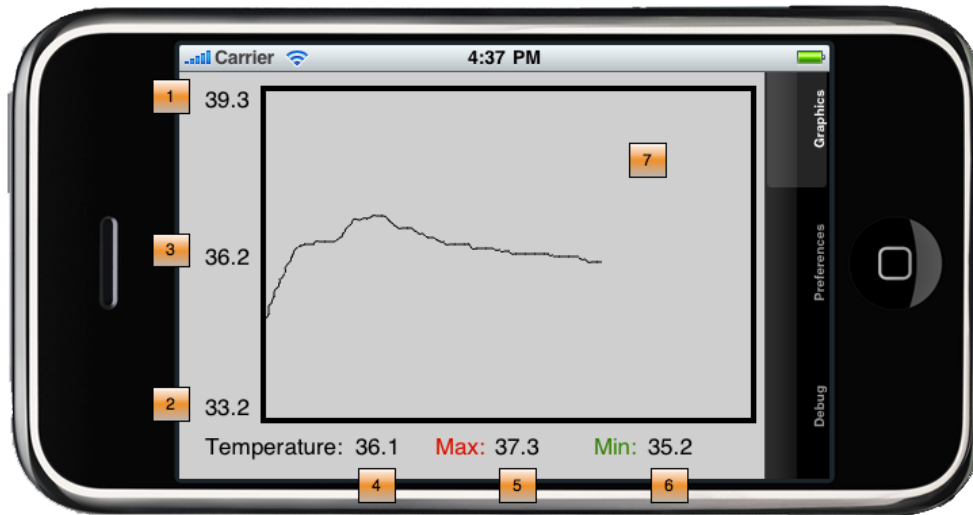


Figure 8.4: iPhone application real time graphical data presentation.

Following the Figure 8.4, "1" represents the maximum absolute temperature value, while "2" presents the minimum absolute value and the label "3" presents the average absolute value. Next, "4" depicts the instant temperature value. Number "5" shows the maximum relative value of the temperature that current session has acquired, while number "6" presents the minimum relative temperature value. Finally, the area identified with "7" appears shows the graphical representation of data. When the signal reaches the end of the screen the application does not erase all the signal drawing, but it keeps all the collected data (historic) for further analysis. Such feature provides a better analysis for the user.

8.3 Deploy, tests and Validation

Since the iPhone OS does not official support bluetooth for RFCComm and SPP [REF] a mechanism has been created in order to follow the standards of the Symbian, Windows Mobile and Android API [72]. The mobile tool was deployed, tested, and evaluated on the Apple iPhone 3G 8GB. The application ran effortlessly on without any problems.

Table 8.1 summarizes the results of a performance evaluated on the iPhone device in

three different approaches. The first column presents the startup time (in seconds) that the iPhone takes to launch the application. The second column has the opening time (in seconds) that takes the iPhone to import the CSV file to the application and become ready for data analysis, processing and presentation. Finally the third column presents the time (in seconds) taken between the CSV data import until data presentation. At this time, the tool starts all the initial data processing. These two times were measured based on an average of thirty experiments.

Table 8.1: iPhone Benchmark Tests.

Device	Initialization time	File Opening time	Data presentation
iPhone 3G	1.2s	2.1s	1.5s

This table does not display the percentage of possibly corrupt data because that information is directly linked to the file that was imported. That is, if you import a file that has cataloged ten corrupt values that will also be passed to this version because it has been previously detected. In order to achieve an applications unification, this version will inform the user if any data is considered corrupt.

In terms of smoothness, the application seems very smoothness. In a range of 1% to 100%, being the 1% the worst performance value and 100% the best one the application can be classified as 95%. These tests take several parameters in consideration, such as smoothness between menus and graphical interface, temperature signal update, label update, and graphic scaling.

Since multithread is software implemented and not a device feature, it is available on all devices, providing the user the capability to make phone calls, text messages, web browsing, email or even play games, while the data is continually acquired.

In terms of battery power consumption, the application consumes the same when compared to a scenario where a standard bluetooth connection is used (phone data transfer). However, the constant use of screen backlight can pose significant battery drain (a scenario similar to the use of a screen saver).

In order to develop this API implementation a specific SDK was used. The iPhone

SDK comes with all of the interfaces, tools, and resources needed to develop iPhone applications. Apple delivers most of its system interfaces in special packages called frameworks. In addition to frameworks, Apple also delivers some technologies in the form of standard shared libraries. Because iPhone OS is based on UNIX, many of the technologies that form the lower-levels of the OS are derived from open-source technologies. The interfaces for these technologies are therefore available in the standard library and interface directories.

Some of the other key components used were the following: Xcode (an IDE that manages the application projects and lets the developer edit, compile, run, and debug the code; Interface Builder (a tool used to assemble the user interface visually. The interface objects created are then saved to a special resource file format and loaded into the application at runtime); Instruments (a runtime performance analysis and debugging tool); iPhone Simulator (a Mac OS X application that simulates the iPhone technology stack, allowing the developer to test iPhone applications locally on Intel based Macintosh computer).

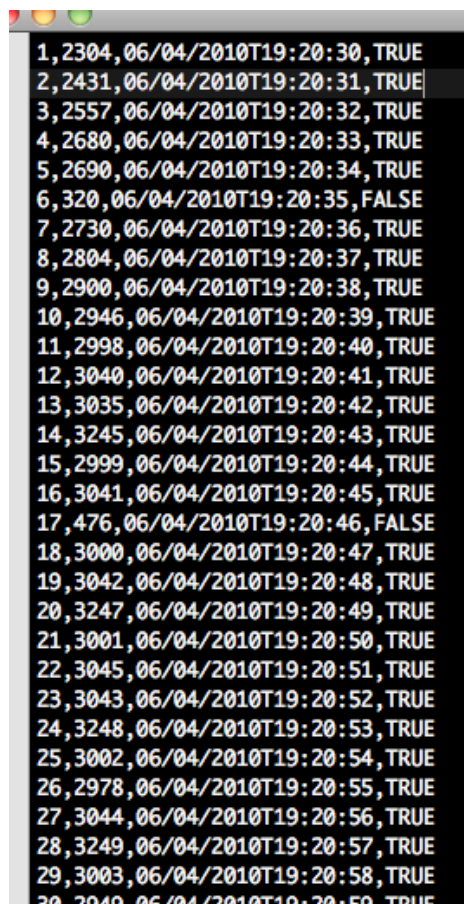
Although the SDK provides the software needed to write applications, Xcode and Instruments also let you interact directly with an attached device to run and debug your code on the target hardware. Development on an actual device requires signing up for Apple paid iPhone Developer Program and configuring a device for development purposes [73].

8.4 iPhone CSV auxiliary file format

Since iPhone has the bluetooth limitation it will use the CSV stored on another application to present a graphical version of data similar to what happens on the other applications. Through this file format it is possible to import it to an external program such as Microsoft Excel or other that support data analysis. The file architecture is composed by four columns: First column represent the number of the current stream; Second column represents the acquired value in a 12bytes representation; Third column represent the data and time of the current received value; Finally the fourth column contains a flag representative of a corrupted or malformed data.

The iPhone application has the particularity to import a CSV file and convert it to graphical display. Since each CSV has its own number of streams and a given frequency associated, the iPhone has the capacity to interpret it correctly and display the information smoothly without any major problem. If at a given time the frequency changes (due to date and time differences) the application automatically informs the user that the frequency changed. Therefore the user can understand if he / she is analyzing data with 1 second or 10 minute frequency.

Figure 8.5 presents a snapshot of a given CSV file stored on an Symbian, Windows Mobile or Android application. This snapshot represents thirty seconds of collected data. It is easily visible that the two malformed or corrupted data are located at positions six and seventeen. Once again the application suspects all data and inspects it with various algorithms.



```
1,2304,06/04/2010T19:20:30,TRUE
2,2431,06/04/2010T19:20:31,TRUE
3,2557,06/04/2010T19:20:32,TRUE
4,2680,06/04/2010T19:20:33,TRUE
5,2690,06/04/2010T19:20:34,TRUE
6,320,06/04/2010T19:20:35,FALSE
7,2730,06/04/2010T19:20:36,TRUE
8,2804,06/04/2010T19:20:37,TRUE
9,2900,06/04/2010T19:20:38,TRUE
10,2946,06/04/2010T19:20:39,TRUE
11,2998,06/04/2010T19:20:40,TRUE
12,3040,06/04/2010T19:20:41,TRUE
13,3035,06/04/2010T19:20:42,TRUE
14,3245,06/04/2010T19:20:43,TRUE
15,2999,06/04/2010T19:20:44,TRUE
16,3041,06/04/2010T19:20:45,TRUE
17,476,06/04/2010T19:20:46,FALSE
18,3000,06/04/2010T19:20:47,TRUE
19,3042,06/04/2010T19:20:48,TRUE
20,3247,06/04/2010T19:20:49,TRUE
21,3001,06/04/2010T19:20:50,TRUE
22,3045,06/04/2010T19:20:51,TRUE
23,3043,06/04/2010T19:20:52,TRUE
24,3248,06/04/2010T19:20:53,TRUE
25,3002,06/04/2010T19:20:54,TRUE
26,2978,06/04/2010T19:20:55,TRUE
27,3044,06/04/2010T19:20:56,TRUE
28,3249,06/04/2010T19:20:57,TRUE
29,3003,06/04/2010T19:20:58,TRUE
30,2949,06/04/2010T19:20:59,TRUE
```

Figure 8.5: CSV file format.

Chapter 9

Conclusions and Future Work

This chapter presents a synthesis of the main achievements and points to several directions for future work. The main objective of this dissertation was to develop a Mobile platform-independent solution for body sensor network interfaces. This was carried out with the construction of four mobile API. This multi mobile platform provides users a way to monitor their data from from his / her BSN. The data processing and presentation is done on real time and the the mobile device of the user is used to display all the information. Therefore all the dissertation objectives were successfully accomplished.

After introducing and delimiting the theme of the dissertation, describing the objectives, and showing its main contributions, in chapter two, the related work in mobile computing using WSN and BSN, together with some insights some mobile OS, was presented. There was highlighted the headed value of the proposed solution.

Chapter three presented the requirements of analysis taken before the mobile platform development process. First, is detailed the essentials requirements for all the development steps. Next, the class, activity and user case diagrams are presented. Next the SHIMMER platform was presented along its SDK.

In chapter four the mobile framework characteristics that are common to all platforms implementations was exploited. The SHIMMER firmware and data flow control is presented. There the bluetooth communication was chosen as the best way to get the data measures from the BSN to the mobile platform. All the algorithms for data acquisition developed to achieve better robustness and security was explained. Next, all the common classes and exceptions created for the implementations were presented

and explained.

Chapter five presented the Symbian OS API implementation and its development details were presented. The evolution of this implementation from a text mode interaction to a GUI was detailed explained. Finally all the deploy devices, tests and inherent validation were explained.

Chapter six presented the Windows Mobile OS API implementation and its development details were presented. The evolution of this implementation from a text mode interaction to a GUI was detailed explained. Next, all the deploy devices, tests and inherent validation are explained. Finally, the Windows mobile emulator and its functionalities were exploited.

Chapter seven presented the Android OS API implementation and its development details were presented. The Android system architecture and the custom activity lifetime is exploited. Next, the evolution of this implementation from a text mode interaction to a GUI was detailed explained. Next, all the deploy devices, tests and inherent validation are explained. Finally, the custom firmware developed for version 1.5 in order to support bluetooth communication was detailed.

Chapter eight presented the iPhone OS API implementation and its development details were presented. The iPhone system architecture is exploited. Next, all the deploy devices, tests and inherent validation are explained. Finally, the CSV file architecture and use was exposed.

A big concern during the system construction was the creation of an application that could run on more limited (processor and memory) devices, in order to amplify the range and easy to use of it. On evaluated devices, the less powered was the Nokia 5320 and the application did not step back in comparison with the other mobile devices.

Taking into account that mobile devices are used to make phone calls, send and receive text, and multimedia messages, among other functionalities, the system was designed to cooperate with them without disturb each others. Then, when other functionalities are under use, the system is prepared to run on the background while a person uses the mobile device without problems. On all evaluated devices (above listed) tested with phone calls and SMS exchange, as expected, the data continuously flowed from the sink to the application without losing performance and smoothly.

In terms of battery power consumption, every version of the application consumes

the same when compared to a scenario where a standard bluetooth connection is used (phone data transfer). However, the constant use of screen backlight can pose significant battery drain (a scenario similar to the use of a screen saver).

After several applications have been developed and conducted several tests we can now draw some interesting conclusions. In terms of initialization time the faster API is the Symbian version followed closely by the Android and iPhone implementation. On this test the Windows version is the slowest and takes an average of 5 times longer to start compared to the Symbian version. In terms of data presentation time all the versions are on a equal baseline. In terms of corrupted data the results vary slightly but the overall results are pretty decent. All the tests were made based on 300 streams, and the worst result indicated that only 17 streams that corresponds to 5.67% were considered corrupted or malformed. Since the world economy is decreasing and the money available for investment is less the cheaper mobile device to acquire is a Symbian OS model, followed by a Windows Mobile device.

To conclude this document, it remains to suggest future research directions that result from this work:

- Increase the number of mobile platforms where the platform can be executed. To achieve this a new way of development should be used. Instead of using the core development languages of each mobile platform and OS a new custom SDK should be created. This is, develop a custom SDK where the development language adopted flexible and that the resulting binary or byte code can be deployed on a large range of multi platform devices.
- Power limitations still one of the weaknesses of systems that use BSN. So the use of energy scavenging techniques could be considered to improve the standalone characteristics aimed for this system.

Thought this application the authors really hope that it can contribute to the increased use of BSN in a more practical, dynamic and with greater portability, but without losing the requirement that is intrinsic to the types of data that are being treated.

References

- [1] M. Franceschinis, M.A. Spirito, R. Tomasi, G. Ossini, and M. Pidala. Using wsn technology for industrial monitoring: A real case. In *Sensor Technologies and Applications, 2008. SENSORCOMM '08. Second International Conference on*, pages 282 –287, 25-31 2008.
- [2] Z. Soferman, D. Blythe, and N.W. John. Advanced graphics behind medical virtual reality: evolution of algorithms, hardware, and software interfaces. *Proceedings of the IEEE*, 86(3):531 –554, mar 1998.
- [3] Peter R. Orszag. Growth in health care costs. Presentation to US Congress, 2008.
- [4] Un population division tables. Online; accessed in July 2010. http://www.un.org/esa/population/publications/popfacts/popfacts_2010-4.pdf.
- [5] M. S. Marzouk. Aging, age-specific health care costs and the future health care burden. 1997.
- [6] P.S. Pandian, K.P. Safeer, D.T. Shakunthala, P. Gopal, and V.C. Padaki. Internet protocol based store and forward wireless telemedicine system for vsat and wireless local area network. In *Signal Processing, Communications and Networking, 2007. ICSCN '07. International Conference on*, pages 54 –58, 22-24 2007.
- [7] Orlando R. E. Pereira, Paulo A. C. S. Neves, and Joel J. P. C. Rodrigues. Mobile solution for three-tier biofeedback data acquisition and processing. In *GLOBECOM*, pages 56–60, 2008.
- [8] Orlando R. E. Pereira, Joao M. L. P. Caldeira, and Joel J. P. C. Rodrigues. A symbian- based mobile solution for intra-body temperature monitoring. In *12th*

- International Conference on E-Health Networking, Applications and Services, (IEEE Healthcom 2010)*, July 01-03 2010.
- [9] Orlando R. E. Pereira, Joao M. L. P. Caldeira, Lei Shu, and Joel J. P. C. Rodrigues. A mobile core-body temperature monitoring system on android. In *5th International Conference on Body Sensor Networks (BodyNets-10)*, September 10-12 2010.
- [10] U. Brandes, J. Lerner, and T.A.B. Snijders. Networks evolving step by step: Statistical analysis of dyadic event data. In *Social Network Analysis and Mining, 2009. ASONAM '09. International Conference on Advances in*, pages 200 –205, july 2009.
- [11] K.R. Fowler. The future of sensors and sensor networks survey results projecting the next 5 years. In *Sensors Applications Symposium, 2009. SAS 2009. IEEE*, pages 1 –6, feb. 2009.
- [12] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38:393–422, 2002.
- [13] Rahul Sarpeshkar. Invited talk: Ultra low power electronics for medicine. *Wearable and Implantable Body Sensor Networks, International Workshop on*, 0:37, 2006.
- [14] Doina Bucur and Jakob E. Bardram. Resource discovery in activity-based sensor networks. *MONET*, 12(2-3):129–142, 2007.
- [15] Adam T. Barth, Mark A. Hanson, Harry C. Powell, Jr., Dincer Unluer, Stephen G. Wilson, and John Lach. Body-coupled communication for body sensor networks. In *BodyNets '08: Proceedings of the ICST 3rd international conference on Body area networks*, pages 1–4, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
- [16] J. A. Stankovic, Q. Cao, T. Doan, L. Fang, Z. He, R. Kiran, S. Lin, S. Son, R. Stoleru, and A. Wood. Wireless sensor networks for in-home healthcare: Potential and challenges. In *High Confidence Medical Device Software and Systems (HCMDSS) Workshop, Philadelphia, PA*. Citeseer, 2005.
- [17] Shih-Lun Chen, Ho-Yin Lee, Chiung-An Chen, Hong-Yi Huang, and Ching-Hsing Luo. Wireless body sensor network with adaptive low-power design for biometrics and healthcare applications. *Systems Journal, IEEE*, 3(4):398 –409, dec. 2009.

- [18] J. Kemp, E.I. Gaura, J. Brusey, and C.D. Thake. Using body sensor networks for increased safety in bomb disposal missions. In *Sensor Networks, Ubiquitous and Trustworthy Computing, 2008. SUTC '08. IEEE International Conference on*, pages 81 –89, june 2008.
- [19] R. Naima and J. Canny. The berkeley tricorder: Ambulatory health monitoring. In *Wearable and Implantable Body Sensor Networks, 2009. BSN 2009. Sixth International Workshop on*, pages 53 –58, june 2009.
- [20] Shu-Di Bao, C.C.Y. Poon, Yuan-Ting Zhang, and Lian-Feng Shen. Using the timing information of heartbeats as an entity identifier to secure body sensor network. *Information Technology in Biomedicine, IEEE Transactions on*, 12(6):772 –779, nov. 2008.
- [21] O.R.E. Pereira, P.A.C.S. Neves, and J.J.P.C. Rodrigues. Mobile solution for three-tier biofeedback data acquisition and processing. In *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, pages 1 –5, 30 2008-dec. 4 2008.
- [22] M.J. Chaudhry, S. Murawwat, F. Saleemi, S. Tariq, M. Saleemi, and F.J. Chaudhry. Power optimized secure bluetooth communication. In *Multitopic Conference, 2008. INMIC 2008. IEEE International*, pages 182 –188, dec. 2008.
- [23] Kim Doang Nguyen, I-Ming Chen, Zhiqiang Luo, Song Huat Yeo, and H.B.-L. Duh. A body sensor network for tracking and monitoring of functional arm motion. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 3862 –3867, oct. 2009.
- [24] Lin Zhong, Mike Sinclair, and Ray Bittner. A phone-centered body sensor network platform: Cost, energy efficiency user interface. *Wearable and Implantable Body Sensor Networks, International Workshop on*, 0:179–182, 2006.
- [25] U. Maurer, A. Rowe, A. Smailagic, and D.P. Siewiorek. ewatch: a wearable sensor and notification platform. In *Wearable and Implantable Body Sensor Networks, 2006. BSN 2006. International Workshop on*, pages 4 pp. –145, april 2006.
- [26] A. A. Reeves, J. W. P. Ng, S. J. Brown, and N. M. Barnes. Remotely supporting care provision for older adults. In *BSN '06: Proceedings of the International*

- Workshop on Wearable and Implantable Body Sensor Networks*, pages 117–122, Washington, DC, USA, 2006. IEEE Computer Society.
- [27] K.W. Goh, J. Lavanya, Y. Kim, E.K. Tan, and C.B. Soh. A pda-based ecg beat detector for home cardiac care. In *Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 27th Annual International Conference of the*, pages 375–378, jan. 2005.
- [28] G. Virone, A. Wood, L. Selavo, Q. Cao, L. Fang, T. Doan, Z. He, R. Stoleru, S. Lin, and J. A. Stankovic. A an advanced wireless sensor network for health monitoring.
- [29] Shih-Lun Chen, Ho-Yin Lee, Chiung-An Chen, Chin-Chun Lin, and Ching-Hsing Luo. A wireless body sensor network system for healthcare monitoring application. In *Biomedical Circuits and Systems Conference, 2007. BIOCAS 2007. IEEE*, pages 243–246, nov. 2007.
- [30] S. Dagtas, Y. Natchetoi, and H. Wu. An integrated wireless sensing and mobile processing architecture for assisted living and healthcare applications. In *HealthNet '07: Proceedings of the 1st ACM SIGMOBILE international workshop on Systems and networking support for healthcare and assisted living environments*, pages 70–72, New York, NY, USA, 2007. ACM.
- [31] Darko Kirovski, Nuria Oliver, Mike Sinclair, and Desney Tan. Health-os:: a position paper. In *HealthNet '07: Proceedings of the 1st ACM SIGMOBILE international workshop on Systems and networking support for healthcare and assisted living environments*, pages 76–78, New York, NY, USA, 2007. ACM.
- [32] S. Kroc and V. Delic. Personal wireless sensor network for mobile health care monitoring. In *Telecommunications in Modern Satellite, Cable and Broadcasting Service, 2003. TELSIKS 2003. 6th International Conference on*, volume 2, pages 471–474 vol.2, oct. 2003.
- [33] Shu-Di Bao, Yuan-Ting Zhang, and Lian-Feng Shen. Physiological signal based entity authentication for body area sensor networks and mobile healthcare systems. In *Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 27th Annual International Conference of the*, pages 2455–2458, 2005.
- [34] E. Jovanov, A. O'Donnell Lords, D. Raskovic, P.G. Cox, R. Adhami, and F. Andrasik. Stress monitoring using a distributed wireless intelligent sensor system.

- Engineering in Medicine and Biology Magazine, IEEE*, 22(3):49 – 55, may-june 2003.
- [35] Dae-Seok Lee, Young-Dong Lee, Wan-Young Chung, and R. Myllyla. Vital sign monitoring system with life emergency event detection using wireless sensor network. In *Sensors, 2006. 5th IEEE Conference on*, pages 518 –521, oct. 2006.
- [36] H. Carroll, J.K. Flanagan, and Satish Baniya. A trace-driven simulator for palm os devices. In *Performance Analysis of Systems and Software, 2005. ISPASS 2005. IEEE International Symposium on*, pages 157 –166, 20-22 2005.
- [37] Google Mobile OS. Android.com android at google i/o. Online; accessed in July 2010. <http://www.android.com/>.
- [38] Samsung Mobile OS.
- [39] Apple Inc. iphone os 4 preview. Online; accessed in July 2010. <http://www.apple.com/iphone/preview-iphone-os/>.
- [40] Limo foundation. Online; accessed in July 2010. <http://www.limofoundation.org/>.
- [41] Home of the maemo community. Online; accessed in July 2010. <http://maemo.org/>.
- [42] Meego open source linux project. Online; accessed in July 2010. <http://meego.com/>.
- [43] Moblin developer community. Online; accessed in July 2010. <http://moblin.org/>.
- [44] Openmoko - open. mobile. free. Online; accessed in July 2010. http://wiki.openmoko.org/wiki/Main_Page.
- [45] Palm OS. Palm developer center. Online; accessed in July 2010. <http://developer.palm.com/>.
- [46] Nokia Mobile Phone. Python for s60. Online; accessed in July 2010. <http://wiki.opensource.nokia.com/projects/PyS60>.

- [47] Blackberry OS. Blackberry developer zone. Online; accessed in July 2010. <http://na.blackberry.com/eng/developers/>.
- [48] Symbian OS. The symbian foundation community. Online; accessed in July 2010. <http://www.symbian.org/>.
- [49] Windows Phone. Mobile phones - windows mobile. Online; accessed in July 2010. <http://www.microsoft.com/Windowsmobile/>.
- [50] the free encyclopedia Wikipedia. Mobile operating systems. Online; accessed in July 2010. http://en.wikipedia.org/wiki/Mobile_operating_system.
- [51] D. Musiani, K. Lin, and T.S. Rosing. Active sensing platform for wireless structural health monitoring. In *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*, pages 390 –399, 25-27 2007.
- [52] Wong Küng-Ing. A light-weighted, low-cost and wireless ecg monitor design based on tinyos operating system. In *Information Technology Applications in Biomedicine, 2007. ITAB 2007. 6th International Special Topic Conference on*, pages 165 –168, 8-11 2007.
- [53] K.J. O'Donovan, B.R. Greene, D. McGrath, R. O'Neill, A. Burns, and B. Caulfield. Shimmer: A new tool for temporal gait analysis. In *Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE*, pages 3826 –3829, 3-6 2009.
- [54] E. O'Connell, S. O'Keefe, T. Newe, M. Healy, E. Lewis, and W.B. Lyons. Development of a prototyping platform for the integration of multiple fiber optic sensing devices to a shimmer x2122; system for in-situ maritime monitoring. In *Sensors, 2009 IEEE*, pages 1800 –1803, 25-28 2009.
- [55] Shimmer research webpage. Online; accessed in July 2010. http://shimmer-research.com/wordpress/?page_id=20.
- [56] Po-Chang Liu, Ming-Jyh Tsai, and Deng-Jyi Chen. A generic software framework for the software system architecture design and implementation of handset devices. In *Computer and Information Science, 2005. Fourth Annual ACIS International Conference on*, pages 169 – 174, 2005.

- [57] J.D. Padgette. Bluetooth security in the dod. In *Military Communications Conference, 2009. MILCOM 2009. IEEE*, pages 1 –6, 18-21 2009.
- [58] G. Lindstrom. Programming with python. *IT Professional*, 7(5):10 – 16, sept.-oct. 2005.
- [59] T.E. Oliphant. Python for scientific computing. *Computing in Science Engineering*, 9(3):10 –20, may-june 2007.
- [60] L. Prechelt. An empirical comparison of seven programming languages. *Computer*, 33(10):23 –29, oct 2000.
- [61] D. Moth A. Wigley and P. Foot. Microsoft mobile development handbook. *Microsoft Press, first edition, 2007*.
- [62] A. Savoldi, P. Gubian, and I. Echizen. A comparison between windows mobile and symbian s60 embedded forensics. In *Intelligent Information Hiding and Multimedia Signal Processing, 2009. IIH-MSP '09. Fifth International Conference on*, pages 546 –550, 12-14 2009.
- [63] B. Scorgie, P. Veeraraghavan, and S. Ghosh. Early virus detection for windows mobile. In *Communications (MICC), 2009 IEEE 9th Malaysia International Conference on*, pages 295 –300, 15-17 2009.
- [64] dm.Klionsky. Bluetooth for microsoft device emulator. Online; accessed in July 2010. <http://code.google.com/p/bthmsdevemul/>.
- [65] S.R. Schach, B. Jin, D.R. Wright, G.Z. Heller, and A.J. Offutt. Maintainability of the linux kernel. *Software, IEE Proceedings -*, 149(1):18 –23, feb 2002.
- [66] Google Android Official. "android introduction". Online; accessed in July 2010. <http://www.android.com/about/>.
- [67] Sharon P. Hall and Eric Anderson. Operating systems for mobile computing. *J. Comput. Small Coll.*, 25(2):64–71, 2009.
- [68] Rahul Shah. Android its anatomy and features. Online; accessed in July 2010. <http://www.ciol.com>.

- [69] Dan R. Herrick. Google this!: using google apps for collaboration and productivity. In *SIGUCCS '09: Proceedings of the ACM SIGUCCS fall conference on User services conference*, pages 55–64, New York, NY, USA, 2009. ACM.
- [70] Google Android Developers. Application fundamentals. Online; accessed in July 2010. <http://developer.android.com/guide/topics/fundamentals.html>.
- [71] Bluetooth Android Developers. Android official bluetooth support. Online; accessed in July 2010. <http://developer.android.com/sdk/android-2.0-highlights.html>.
- [72] iPhone OS Development. iphone os reference library. Online; accessed in July 2010. <http://developer.apple.com/iphone/library/navigation/index.html#section=Topics&topic=Networking%20%26amp%3B%20Internet>.
- [73] Apple Developer. iphone development program. Online; accessed in July 2010. <http://developer.apple.com/iphone/program/>.