SENSORS ON MOBILE DEVICES FOR AAL

João Manuel Cardoso Madureira

Mestrado em Engenharia Electrotécnica e de Computadores
Área de Especialização de Telecomunicações
Departamento de Engenharia Electrotécnica
Instituto Superior de Engenharia do Porto
2013

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Disciplina de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Electrotécnica e de Computadores

Candidato: João Manuel Cardoso Madureira, Nº 1060392, 1060392@isep.ipp.pt
Orientação científica: Prof.ª Doutora Paula Maria Marques Moura Gomes Viana, pmv@isep.ipp.pt
Empresa: Fraunhofer Portugal
Supervisão: Eng. Diogo Dias Júnior, diogo.junior@fraunhofer.pt

Mestrado em Engenharia Electrotécnica e de Computadores
Área de Especialização de Telecomunicações
Departamento de Engenharia Electrotécnica
Instituto Superior de Engenharia do Porto
20 de novembro de 2013

Dedicated to my parents...

Acknowledgements

Firstly, I would like to thank my research supervisor, MSc. Diogo Júnior, for giving me this great opportunity of working and learning in Fraunhofer Portugal. For his availability, help, teachings and encouragement throughout this work. I would also like to extend my thanks to other Fraunhofer researchers that were always very helpful and encouraging.

Secondly, I would like to express my gratitude to Prof. Dr. Paula Viana for her guidance, patience, assistance and for accepting to be my supervisor, a big thank you for her.

I would also like to thank my girlfriend Marta for her support, patience, care and for always be by my side.

Finally, and most important, I would like to express my deep gratitude to my parents, for their support, encouragement and guidance throughout this journey. I could have not succeeded without them.

Resumo

Com o envelhecimento da população, as preocupações com a garantia do seu bem-estar aumentam criando a necessidade de desenvolver ferramentas que permitam monitorizar em permanência este sector da população. A utilização de smartphones pelos mais velhos pode ser crucial no seu bem-estar e na sua autonomia contribuindo para a recolha de informação importante já que estes estão muitas vezes equipados com sensores que podem dar indicações preciosas ao cuidador sobre o estado atual do paciente.

Os sensores podem fornecer dados sobre a atividade física do paciente, bem como detetar quedas ou calcular a sua posição, com a ajuda do acelerómetro, do giroscópio e do sensor de campo magnético. No entanto, funcionalidades como essas requerem, obrigatoriamente, uma frequência de amostragem mínima por parte dos sensores que permita a implementação de algoritmos, que determinarão esses parâmetros da forma mais exata possível.

Dado que nem sempre os pacientes se fazem acompanhar do seu smartphone quando estão na sua residência, a criação de ambientes de AAL (*Ambient Assisted Living*) com recurso a dispositivos externos que podem ser "vestidos" pelos pacientes pode também ser uma solução adequada. Estes contêm normalmente os mesmos sensores que os smartphones e comunicam com estes através de tecnologias sem fios, como é o caso do *Bluetooth Low Energy*.

Neste trabalho, avaliou-se a possibilidade de alteração da frequência dos sensores em diferentes sistemas operativos, tendo sido efectuadas modificações nas instalações por defeito de alguns sistemas operativos abertos. Com o objectivo de permitir a criação de uma solução de AAL com recurso a um dispositivo externo implementaram-se serviços e perfis num dispositivo externo, o SensorTag.

Palavras-Chave

Cuidador, paciente, AAL, sensores, Bluetooth Low Energy, SensorTag, smartphone, sistema operativo.

Abstract

With the ageing of the population and the concerns about the people's well-being increasing, new tools need to be developed in order to allow the permanent monitoring of this demographic group. The use of smartphones by elders can be crucial in their well-being and autonomy since they are a lot of times equipped with sensors that can provide caregivers with precious indications about the caretaker current status.

Sensors can provide data about the patient's physical activity, as well as detecting falls or calculate its position, with the help of the accelerometer, the gyroscope or the magnetic field sensor. But functionalities as these require, mandatorily, a minimum sampling frequency from the sensors that will allow the implementation of algorithms that will determine those parameters in the most exact way possible.

Having in mind that the caretakers often are not accompanied by their smartphone when they are at their home places, the creation of AAL environments with the help of some other external devices that can be "worn" by the patients might also become an adequate solution. These devices normally contain the same sensors as the smartphones and communicate with them through wireless technologies, like Bluetooth Low Energy.

This work evaluates the possibility of sensor frequency modification in several Operating Systems, and some modifications in the source code of some open source systems were performed. With the objective of allowing the creation of an AAL solution using an external device, profiles and services were implemented on an external device, the SensorTag.

Keywords

Caregiver, caretaker, AAL, sensors, Bluetooth Low Energy, SensorTag, smartphone, Operating System.

Table of Contents

Index of Figures

Index of Tables

Acronyms

AAL – Ambient Assisted Living

ADC – Analog-to-Digital Converter

AFH – Adaptive Frequency Hopping

AICOS – Assistive Information and Communication Solutions

AOSP – Android Open Source Project

API – Application Programming Interface

BLE – Bluetooth Low Energy

CPU – Central Processing Unit

CSS – Cascading Style Sheets

DMP – Digital Motion Processor

FFOS – Firefox Operating System

GPS – Global Positioning System

GPU – Graphics processing Unit

HAL – Hardware Abstraction Layer

HTML – HyperText Markup Language

I^2C – Inter-Integrated Circuit

ISEP – Instituto Superior de Engenharia do Porto

JSON – JavaScript Object Notation

LED – Light-Emitting Diode

MAC – Media Access Control

MEMS – Micro-Electro-Mechanical Systems

OS – Operating System

PDA – Personal Digital Assistance

PDU – Protocol Data Unit

QML – Qt Meta Language

RAM – Random Access Memory

RF – Radio Frequency

ROM – Read-Only Memory

SMS – Short Message Service

SPI – Serial Peripheral Interface

USB – Universal Serial Bus

VM – Virtual Machine

XML – eXtensible Markup Language

1. INTRODUCTION

The work presented in this thesis is the result of an internship made at Fraunhofer Portugal, conducted under the scope of the Master in Electrical and Computer Engineering - Telecommunications, in the Instituto Superior de Engenharia do Porto (ISEP).

1.1. CONTEXT

With the continuous ageing of the population, it is fundamental to be able to give people the possibility of living an autonomous life, for the longest time possible, and new mobile solutions are appearing each day that can help users do that. Fraunhofer AICOS, among other things, develops mobile solutions that can help needed patients. Applications like activity monitoring, fall detection and indoor location can be used to achieve that goal, and are continuously being improved by the company's scientists.

Sensors are fundamental parts of those applications. Accelerometers, gyroscopes or magnetometers can give pretty accurate data about the positioning, orientation and other parameters of the devices they are attached. The interaction between sensors and smartphones is being improved with the help of Mobile Operating Systems. They provide APIs that can be easily used to make robust and powerful applications based on sensor data. But as the users do not always carry their smartphones with them, other devices can be used to perform the same exact tasks, using some way of communication, like Bluetooth Low Energy.

1.2. OBJECTIVES

The main objective of this work is to investigate sensor access in the upcoming Open Source Mobile Operating Systems, as well as in the dominant OS on the world nowadays, Android. Possible improvements in this task may allow the use of improved algorithms in the fields of fall detection and activity monitoring and they should be implemented whenever possible. Two approaches should be implemented: use built-in sensors in mobile devices like tablets and smartphones, and the use of external sensors which can be connected to mobile devices through a wireless link.

So, the main objectives are:

- Research about the upcoming mobile operating systems and their sensor access;
- Change the source code relative to sensor access and improve the sensor sampling rates;
- Develop new profiles and services on an external device in order to be possible to adapt it to AAL environments, having in mind the knowledge gathered previously about sensor access.
- Accomplish a proof of concept, allowing easy later implementations to mobile devices by the company scientists.

In order to achieve those objectives, the approach was to first perform a thorough research about the most promising Mobile Operating Systems on the market. Then, a research on how to build and install modified versions of those Operating Systems had to be made, along with the possibility to do that. The study and firmware modification of external devices was the final step, with its concept tested with the help of Android Operating System.

1.3. DISSERTATION STRUCTURE

Apart from this Introduction (Chapter 1), this dissertation consists of five chapters. In Chapter 2 a review of the related AAL products is made, some concepts about assisted living are explained and a review about the sensors and communication technologies used in AAL are also analyzed. In Chapter 3, Mobile Operating Systems are investigated, their

main features are studied, including the available development environments or the system's architecture. Chapter 4 describes the necessary steps taken in order to optimize sensor access in the studied operating systems and results are presented. The adaptation of an external device to implement assisted living tasks is described in Chapter 5 and the major achievements of this project and some future work are pointed in Chapter 6.

2. AAL PRODUCTS AND RELATED TECHNOLOGIES

Developed societies are faced nowadays with the population's aging problem, the world is aging at a very fast rate and this tendency will be accompanied by the increase of people with physical limitations. In 2050, about 16% of the world's population will be over 65, opposing the 7% nowadays [1]. So, new health solutions and new approaches regarding the traditional health care system are unavoidable. Older people will want to be able to stay in their homes, being assisted in their diary activities and live independently for longer, while still feeling secure, protected and supported. This can be made using sensors and this technology has proven recently to be a very effective way to improve people's life. They can provide data that can be used to evaluate the user's current activity, location, giving the possibility of alerting a person or an entity if something is wrong.

Over the last few years, the use of sensors in mobile devices have been increasing significantly and they have now an important role in the people's way of living. They give the users a lot of new features that are being incorporated every day on an endless number of different areas. Applications that respond to gestures and movement are now possible, like a racing car game that do the steering by rotating the device, for example. Sensors allow smartphones to be "really smart" under determined circumstances and applications

that use them are evolving quickly. Success of smartphones is leading to an increasing amount of *Micro Electro Mechanical Systems* (MEMS) and sensors in this area, providing new services and improving hardware performance. Sensors are vital for the user-smartphone interaction nowadays and it will continue to be that way in the future.

Examples of AAL actuation places are the home where the user lives, the town, the store, but also the user's body itself. The user can be monitored using the smartphone sensors, or may be able to wear devices that contain external sensors, like a pendant that can detect falls [2], as well.

The main advantage of those devices reside in the fact that this allows the patient not to be directly monitored by a caregiver (a relative, a doctor, etc.), but instead, the patient can be remotely monitored by independent devices that can alert caregivers when something goes wrong. The caregivers are able to fully live their lives normally, taking actions accordingly when something isn't right.

A Smart System can be described as an integrated system that can diagnose and describe a situation. It can achieve functionalities and is able to communicate with other Smart Systems and other environments, performing multiple tasks and assist the users in different activities in their daily lives [3]. In the AAL case, the most common Smart Systems are the ones who can send information to someone or something, based in sensor data over a period of time. These environments include sensors and motion detectors on doors, for example, and they can detect if the door was opened and which one, creating the concept of Smart Homes. The advantage of this approach is that it avoids the problem of misplacing the sensors or to damage the wearable devices. Smart Home technology including ambient sensors has been a fundamental piece of rehabilitation and monitoring related applications. AAL is the application that has the focus on this dissertation and in this case refers to smart systems that can be a part of health assistance in the individual's living environment [4] [14].

The remote monitoring of a patient status and self-awareness of his condition by the patient are the most often pursued applications for AAL technologies. Systems where wearable and ambient sensors are combined represent the most accurate prototypes nowadays using this kind of technology.

In the initial stage of this master thesis, an analysis of the current state of the art and related work was executed. The aim of this research was to learn more about the way sensors are used in *Ambient Assisted Living* (AAL), showing the advantages and disadvantages of having embedded sensors in mobile devices, opposing to having external devices that can be connected to them.

2.1. PRODUCTS FOR AAL

SenseWear Armband, for example, is a wearable device that has multiple sensors that allow the device to monitor patient's motion, steps taken, galvanic skin response that measures the electrical conductivity of the skin, which changes in response to sweat, among other features [7]. Its appearance can be seen in figure 1.

Figure 1 - SenseWear Armband

This device can store data for 28 days, which means that it can save the month's patient history, giving doctors and caregivers a very accurate estimate about how the patient is doing.

Other companies like *emcare* [8] offer other solutions like **Panic Alert**. This device sends an alert to a care center when a button is pressed. If the patients aren't feeling very well or if an intruder wants to enter their property, they can send and alert and will receive a call from them asking if something is wrong. If nobody answers the call, the care center will then notify one of the priority contacts assigned by the users, letting him know about the situation. This is not a wearable device, instead, it will be installed next to the bed or at any place that could be of easy reach by the patient. This device is supposed to have up to five

years in battery life, depending on activations. The communication with the care center is made using Wi-Fi, so an additional device is needed to send the notification.

On-Wrist Fall Alert is another product developed by *emcare*. This device has been developed to be permanently used around the wrist and uses an accelerometer to determine if a fall has occurred. If it detects a fall, this device will start to vibrate a few seconds later, to let the user know that a fall has been detected and if the user moves his arm, the alert will be cancelled.

Figure 2 - On-Wrist Fall Alert

Pendant Fall Alert is another kind of product from the same company and it was made to be used as a pendant, around the neck. Like On-Wrist Fall Alert, it has an accelerometer that can detect falls and, like Panic Alert, has a button that can be pressed to raise an alarm. It joins both "worlds". In case of an alert, those two devices will alert the care center [8].

Other applications use the smartphone sensors to do tasks like activity monitoring. One of those applications is the **Mover** application developed at Fraunhofer AICOS. Mover reads the data from the phone's accelerometer through the day, giving an estimate about the user's daily activity. This application can also make the user aware if he is moving more or moving less than usual, this way he can choose to be more active or maybe rest a little, depending on this information. Mover also has a fall detection algorithm, which triggers an alarm in case of falling, and may alert one of the emergency numbers chosen via SMS and/or e-mail. Then, the caregiver can perform the actions necessary for the situation.

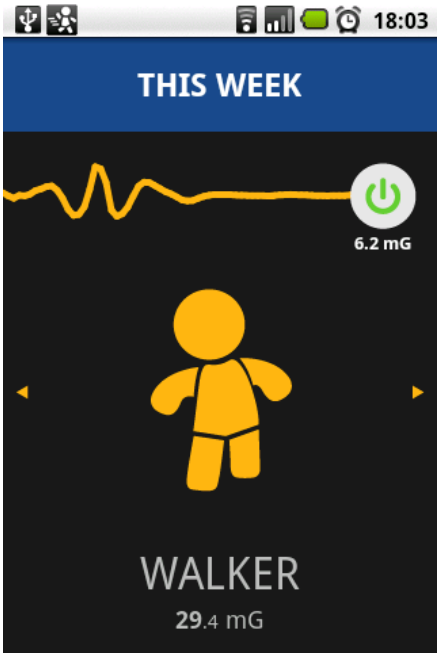

Figure 3 - Mover application

In figure 3 it is possible to see the appearance of the Mover application, in this case, the patient is shown as a "walker" which means that this week his activity has been average [18].

2.2. SENSORS

Most of mobile devices nowadays have built-in sensors that measure orientation, accelerations and rotation rates, as well as environmental conditions. These sensors can provide raw data with high precision and accuracy or can provide modified data using sensor fusion. If the user wants to monitor changes in the environment, to monitor user motion, among other features, he will have to use the sensors existent in devices.

It is possible to define three types of sensors that can be used to create useful applications: Motion, Environmental and Position sensors.

Motion sensors measure acceleration forces and rotational forces along three axes. Accelerometers and gyroscopes are in this category.

Environmental sensors measure environmental parameters, such as temperature, pressure and light, between others. Barometers, photometers and thermometers are environmental sensors.

Position sensors give the physical position of the device. This category includes magnetometers and orientation sensors [27].

The sensors that are the focus of this dissertation are the accelerometer, the magnetometer (magnetic field sensor) and the gyroscope. These three sensors can give data used to estimate the user's current indoor position, fall detection, among other related activities. But other sensors like the proximity sensor, the light sensor and the barometer are also used in many smartphone applications.

2.2.1 ACCELEROMETER

Accelerometers are capable of detecting accelerations and the G-force associated with the movement. These sensors are nothing more than a MEMS based circuit that measures the forces of acceleration caused due to gravity of movement or tilting action. They measure the acceleration of the device to which it is attached. Recent accelerometers usually have three axes (X, Y and Z) which allows to correctly determine the acceleration and if the device is with the screen turned up or down, for example [6] [11].

There are a lot of different applications that use these sensors like fall detection or gesture recognition. In smartphones, they are used mostly for switching between landscape and portrait modes and this sensor is also used as a tilt sensor for tagging the orientation to photos that are taken with the built-in camera of the phone [19].

2.2.2 GYROSCOPE

These sensors don't measure an external reference like accelerometers or magnetometers, they measure their own rotation. They are based on the principles of angular momentum. A rotating object has this angular momentum that will tend to rotate the object continuously. So, when the spinning wheel is rotating, it will resist to any change in its axis of rotation. If an external force is applied to change it, an opposite reaction that pushes in the opposite direction will be created. Roll, the rotation around X axis, pitch, around Z axis and yaw, rotation around Y axis, will be automatically detected by the device's gyroscope. Unlike an accelerometer, it can detect if a device made a full spin or is experiencing inertial change, and not only the translation of the direction [11] [15].

A mechanical gyroscope contains a spinning wheel that is mounted inside two metallic rings orthogonal to each other. The spinning wheel will continue to spin on its axis, regarding of the alignment of the outer rings [81].

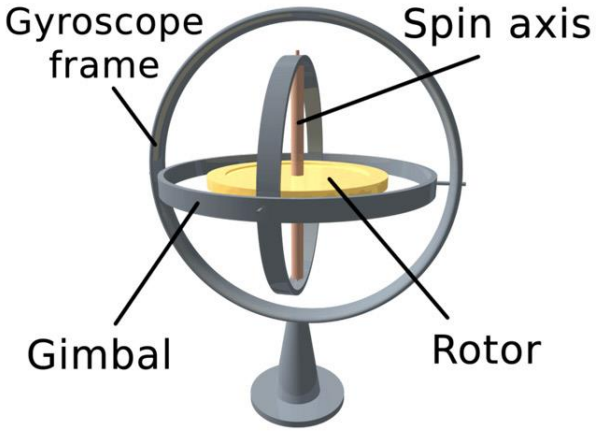

Figure 4 - Mechanical gyroscope

This sensor is largely used for motion sensing capabilities to create games that rely on motion control, instead of on-screen controls. When combining a gyroscope and an accelerometer, a 6-axis motion sensing is available, that could detect precise motion by simply moving the phone naturally. This way the smartphone is much more effective and has motion sensing capabilities comparable to a game controller like the Wii-mote.

The majority of the nowadays smartphone games use a gyroscope to control them. For example a game where a plane is to be controlled, can rely on the gyroscope to make the plane go left or right, up or down and this is made much more accurately than if it was done by just an accelerometer. In fall detection, the use of gyroscopes can improve the fall detection accuracy, because smartphones can then detect not only the acceleration (with an accelerometer) of the device, but also its angular velocity, and, combining those two data types, it is possible to detect false positives more accurately like *Ojetola et al*. shown in their study, *Fall Detection with Wearable Sensors* [16].

Gyroscopes are complex in implementation and it involves high precision in design and electronics. Usually on mobile devices these sensors are the ones with the higher power consumption, so applications that use this sensor will drain the phone's battery faster [6] [11].

2.2.3 MAGNETOMETER

Unlike regular compasses, a digital compass does not use magnets to get attracted to Earth's magnetic field, because this would lead to damaging the smartphone's capabilities. There is not a spinning needle. Most electronic compasses work by magneto-inductive technology, reading differences in the Earth's magnetic field. In order to get clear readings, if the compass is near some kind of device or near a large mass of metal, like a car, it needs to subtract that reading from the reading of Earth's magnetic field, to get a valid reading [6] [11].

Magnetometers that exist on smartphones can behave as metal detectors and as compasses, but the innovative application with theses sensors is, undoubtedly, the indoor navigation. Researchers from the University of Oulu in Finland created an *Indoor Positioning System* (IPS) that uses the Earth's natural magnetic field to estimate position, **IndoorAtlas** [49]. In their study, they demonstrate that every square inch of Earth emits a magnetic field, and if these magnetic fields could be mapped, a magnetic field sensor can detect those magnetic fields, making a map of the surroundings, and indoor positioning is possible. The main advantage of this approach is the fact that no extra hardware is needed, no Wi-Fi or Bluetooth base stations, just the smartphone and its magnetic field sensor is enough. The disadvantage of products like this is that magnetic fields change. As the molten iron in the Earth changes, so do magnetic fields. These shifts are not very fast, but even so, this will translate on an error if the user does not update the maps regularly. Another big disadvantage is that objects like cars, motors, any object that will have magnetic fields associated to them, will change the surrounding magnetic field and will cause errors if they are not contemplated in the original map [50].

It is useful to notice that magnetic north and the geographic north are not the same. The angle between them is called magnetic declination. In Portugal, for example, the declination is something like 4 degrees West, but this value will be different depending on the location [51]. So, people who want to use applications that will rely on the magnetic north will have to bear this in mind.

2.2.4 PROXIMITY

A proximity sensor is a device that detects objects when they get within a certain distance from the sensor. If the object gets at that distance, the sensor usually turns on a circuit to

perform some basic function like turn on or turn off the screen light. They are largely used in industry today, in distinct areas. The main operating mode is to send a wave of some form and monitor reflections. If a reflection is sensed, the sensor confirms that there is an object nearby. There is four main types of proximity sensors: infrared, acoustic, inductive and capacitive.

Inductive sensors sense objects with the generation of magnetic fields. A coil is charged with electrical current, and this current is measured by a circuit. If metallic parts get close enough to the coil, the current will increase significantly and the proximity switch will open or close accordingly. These sensors have the disadvantage of only detecting metallic objects.

Infrared sensors send out beams of invisible infrared light, and then a photodetector located on the sensor detects any reflections of this light. As these switches are just a light source and a photodiode, they are susceptible to false readings due to background light.

Acoustic proximity sensors work similarly to infrared switches but use sound instead of light. Acoustic sensors measure the time that it takes for sound pulses to "echo" and this time is used to calculate the distance, just like a sonar.

Finally, **capacitive** switches sense objects by detecting changes in capacitance around them. An oscillator is connected to a metal plate and when the plate gets near an object, this radio frequency changes, and the frequency detector sends a signal. These switches are much more sensible sensing objects that conduct electricity than the ones which do not.

Smartphones are equipped with infrared sensors. The most common application for this sensor in smartphones is to disable accidental touch events while in a call. This avoids the generation of unwanted touch events by the ear [12] [52] [82].

2.2.5 LIGHT

Ambient light sensors are largely used nowadays in mobile devices and allow to optimize device displays to the environment or to gather ambient light parameters. The devices brightness optimization is made using the values from these sensors. If they detect that the lightning condition is too brighter or too dark, different brightness values will be given to the device's display allowing to see well both in the dark and in the day light. Other issues like battery consumption are also affected as the device consumes less power when the

display is dim. This auto adjustment of display brightness reduces the strain on the user's eyes and protects the screen pixels.

Most of the light sensors used today, use the principle of superposition to calculate the ambient light brightness. For this, they use two or more different types of photodiodes, each sensitive to a different portion of the light spectrum. When these devices are exposed to light they create a current. The brighter the light, the higher the current. This is the main principle of these sensors. Combining these photodiode outputs mathematically, each one with a suitably adjusted gain, the sensor can obtain a really accurate measurement of environment brightness for the light sources commonly available.

Usually, these sensors are not used when a quick response is needed. However, there is a problem inherent to these sensors that has been debated lately. The sensor generally is not really measuring the light level of the environment, because usually the user is facing the sensor and his head blocks the light. So, the brightness level of the device and the eyestrain can be really affected, reducing significantly the utility of this auto adjustment application while the user is using the smartphone [13] [19].

2.2.6 BAROMETER

This instrument is used in meteorology to measure atmospheric pressure. Essentially measures the weight of the air. It is possible to measure the pressure exerted by the atmosphere by using mercury, water or air in a way that is possible to forecast short term changes in the weather.

Digital barometers are really tiny and they often can be found in some dedicated GPS units and watches. They replace all the mechanical parts of a regular barometer with a simple pressure sensing transducer, which measure pressure in Pascal. They need to be calibrated in order to deliver accurate readings and that usually is done by giving them a correct pressure reading regarding the elevation.

Barometers were included recently in smartphones and the reason for that inclusion was the altitude value. It is possible to determine changes in altitude based on barometer readings, because the atmospheric pressure is directed related to the altitude. So, barometers are mostly used to aid the GPS sensor in correcting altitude measurements.

These sensors can help a device to provide a more accurate weather forecast. Atmospheric pressure when combined to general forecast for the air can warn the users to upcoming rains and such, a little bit in advance. This could also help many people, especially older ones, and people with some health conditions that are very sensitive to changes in the atmospheric pressure. They could be noticed faster about those changes and act accordingly by taking certain pills or to lay down for a while, just to give some examples. Altitude is much more accurate than the altitude measured by GPS systems, so it is possible to know in which floor the user is more accurately, giving indoor navigation and location service a new approach [6] [19].

2.2.7 SENSOR SAMPLING RATE

One of the most important features about a sensor is its sampling frequency or sampling rate. With a higher sampling rate it is possible to obtain more values during a period of time, and this can make application interaction more precise and accurate.

Output data rate on a sensor defines the rate at which the data is sampled and the maximum frequency a sensor can achieve [11].

Choosing a sensor sampling rate will always have to take in consideration the use of the application or system. An application that uses the values from the light sensor will, usually, use a sampling rate a lot smaller than an application for activity monitoring that uses the accelerometer. Ambient light values are likely to change a lot less than the accelerometer values when someone is walking, for example, and there is no need for a large number of data, that will consume more battery. So, a user needs to choose the minimum sampling rate (fewer samples per second) that still meets the requirements for the application, because this way power consumption will be reduced.

In order to maximize an application that uses sensor data, selecting a sampling rate may be crucial and has to follow some parameters:

- Sample as fast as possible to obtain the greatest accuracy;
- Slow enough that the noise doesn’t dominate the input signal;
- Slow as possible not to overload the application’s processor;
- Fast enough to guarantee adequate response time.

These parameters may change according to which sensor has been in use, because some sensors are more affected by drift and noise than others [10].

Achieving higher frequencies may be important to applications that require a fast response time like a mouse that is controlled by accelerometers and gyroscopes, to help a Parkinson patient to be able to read a text on a smartphone using sensor data to counter his trembling, among others. These applications need to gather sensor data as soon as possible to be able to take the according responses in time.

The smartphone sensors have the obvious advantage of being embedded on a smartphone. There is no need for additional hardware and the majority of the recent smartphones have the sensors that can do that.

2.3. COMMUNICATION TECHNOLOGIES

There are some wireless technologies that can be used to make the connection between mobile devices, like a smartphone and a wearable device. Technologies like Wi-Fi or Bluetooth are commonly used nowadays for these kinds of connections, but there are other less known technologies that can play an important role in the future such as Bluetooth Low Energy or Zigbee.

2.3.1 WI-FI

Wi-Fi [37] allows the exchange of data at high-speed. It is mainly used in devices that exchange data over a computer network, such as the internet. Any device can connect to the internet using an *Access Point* (AP), so this is an extra hardware needed when compared to Bluetooth [23].

This technology is a group of specifications for Wireless Local Area Networks (WLAN), based on the IEEE 802.11 [35]. Wi-Fi is the most common wireless technology nowadays and can be found almost everywhere, from hotels to airports, schools and many other because to be able to use it, the user only needs to have a laptop, a smartphone or other compatible device.

SSID (Service Set Identifier)

In order to make a Wi-Fi network, it is necessary that the *stations* (STA) to connect to devices that can provide the access, the AP. When one or more STAs connect to the AP, a

network is created and is called *Basic Service Set* (BSS). When more than one BSS are created in the area, it is crucial that they receive a different SSID, which is inserted in each packet of the network. SSID is the name of each wireless network. Figure 5 shows an AP that can be used by many different STAs forming a BSS [23] [36].

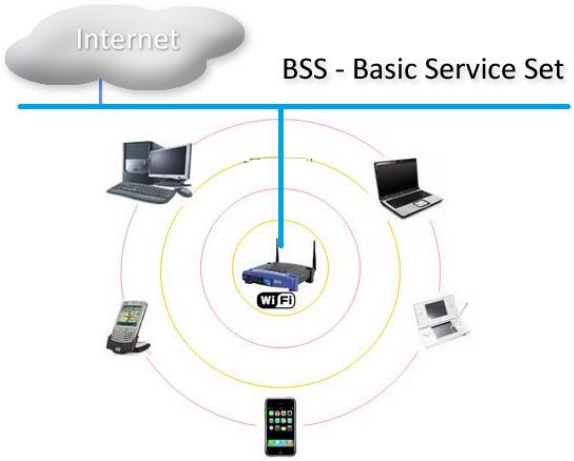

Figure 5 - Wi-Fi network

Security

In wired technologies, such as Ethernet [38], it is easier to control network access because a physical connection is needed. In Wi-Fi this is not necessary, anyone with a compatible technology can connect to the network. So, in order to block unwanted user access, security schemes were created.

Wired Equivalent Privacy (WEP) appeared with the original Wi-Fi and is basically an authentication mechanism. Each device need to provide a key that is used to cypher the network's data. These keys can either be 64 or 128 bit long. But the WEP uses keys that may be easily discovered with the application of some techniques. So another solutions were investigated.

Wired Protected Access (WPA) also is based on authentication and network cyphering, but the way it is done is safer and trusted. The base of this kind of security is a protocol, *Temporal Key Integrity Protocol* (TKIP). A 128 bit key is used by the network devices and combined with the MAC Address (a distinct hexadecimal code for every device on the network) of each STA. As this address is different for each device, a unique sequence is generated. The key is periodically switched, unlike WEP [23] [36].

But more recently, other even more secure scheme has appeared. WPA2 uses a secure pattern named *Advanced Encryption Standard* (AES), but has the disadvantage of requiring a lot of processing. This security scheme is recommendable for high security networks [36].

Lately, a new concept was introduced, called Wi-Fi Direct [39], where devices can connect to each other without the need of an access point. This technology totally changes the way *adhoc* connections work. With this new protocol, any device can become an access point, allowing other Wi-Fi compatible devices to establish a connection. It is possible to say that Wi-Fi Direct works in a similar way as Bluetooth, but with higher rates, it can achieve 250 Mbps. The negative point is that unlike Bluetooth, this technology will require more power, causing mobile devices to drain the battery quicker. This makes this technology not so attractive for devices that do not need a high transmission rate. Wi-Fi Direct is already implemented in some of the mobile Operating Systems, like Android [40].

It allows direct data transfer between devices with minimal setup using *Wifi Protected Setup* (WPS) to establish a connection. It requires that the user knows the device name that he wants to connect to. The range of a Wi-Fi Direct connection is 100 meters [22].

2.3.2 ZIGBEE

Zigbee is used to create *Personal Area Networks* (PAN) built from small, low power digital radios. Devices often transmit data over long distances by passing data through intermediate nodes to reach the distant ones, creating a mesh network, with no centralized control or high power transmitter or receiver able to reach all the network's devices [23].

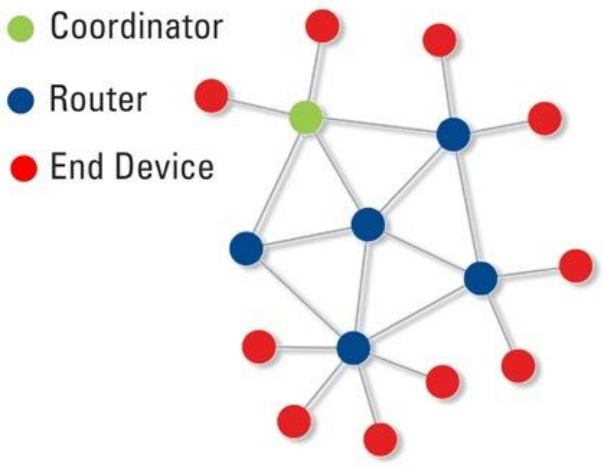

Figure 6 - Zigbee network

In Zigbee, as it is possible to see in figure 6, there has to be three different kinds of nodes. Coordinators that are the master devices and coordinate all the network, Routers that route the information which is sent by the end devices and the End Devices that usually contain the information about the environment. To be a fully mesh network, a system has to be peer-to-peer, which means that all the devices can connect with each other directly sending broadcast messages. This does not occur in Zigbee, an end device has to connect to a router or a coordinator and then talk to other end device [23] [83].

With not high data rates, 250 kbit/s maximum, is best suited for periodic or intermittent data or to receive a single signal from a sensor or input device. Light switches, remote controls and other industrial or consumer equipment that requires short-range wireless data transfer at relatively low data rates.

This technology [42] was established in 2002 by a group of 16 companies and it introduces mesh networking to the low power wireless spaces and its primary targets are home automation and remote controls. Due to Zigbee's complexity this technology may be flawed in environments where devices need to be able to operate for extensive periods from a limited power source or in environments that may need relatively high data transfer rates [25].

It enhances the IEEE 802.15.4 standard [43] by adding some security and network layers and an application framework. The Zigbee Alliance developed standards that can be used to create interoperable solutions and for custom applications where interoperability is not a requirement, manufacturers can create their own manufacturer specific standards.

Zigbee technology can achieve some distance (100 m) in communication and its channels are similar to the channels in BLE, but as Zigbee is not a frequency hopping technology, its planning has to be careful, to ensure that there are not much interferers where the communication is happening [25]. It operates in the 2.4 GHz frequency and over 16 channels and used the AES-128 bit security scheme also used in Wi-Fi networks [23].

Although at the moment this technology is not integrated out-of-the-box in smartphones, it is expected that it will be incorporated soon, and some major manufacturers already discussed and shown interest in this kind of technology. With this, users would not need separate applications for each Zigbee equipped piece of hardware as control could be consolidated on the smartphone device.

2.3.3 BLUETOOTH

Bluetooth [44] is a wireless technology standard for exchanging data over short distances by many types of devices. It is a relatively low power technology, ant its data transfer is made through RF, allowing the device to detect each other if they are inside the proximity range.

When two or more devices communicate in a Bluetooth connection, they constitute a *piconet*. In this technology, the device that starts the communication is the Master and the other ones are Slaves. The master controls the data transmission of the network and the device synchronism. Each piconet can support up to 8 devices, but it is possible to increase this number by overlapping piconets. These networks are called *scatternets*. In a scatternet, a slave can be in more than one piconet, but a master cannot. This behavior is represented in figure 7 [45] [84].

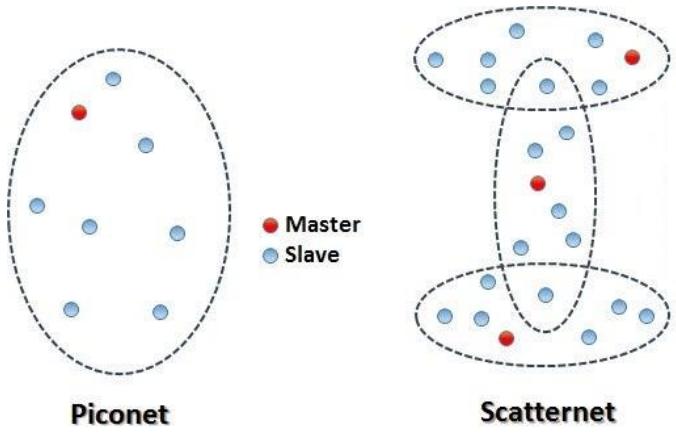

Figure 7 - Bluetooth network

The key features of Bluetooth are low cost, low power and robustness. When two devices connect to each other, this process is called pairing. The structure and global acceptance of the technology means that any enabled device can connect to other enabled device, everywhere in the world. A fundamental strength of this technology is the possibility of handling voice and data simultaneously, which provides some useful applications like hands-free headsets for voice calls or synchronization between smartphones and computers, for example.

Bluetooth operates under the unlicensed *industrial, scientific and medical* (ISM) band at 2.4 to 2.485 GHz, using a spread spectrum, frequency hopping and full-duplex signal. The 2.4 GHz ISM band is available in most countries, making this technology global.

Interference between connections is avoided using *Adaptive Frequency Hopping* (AFH). This technology allows to detect other devices in the spectrum and avoids using the frequencies they are using. This adaptive hopping is done among 79 frequencies at 1MHz intervals, which give a high degree of interference immunity and a more efficient data transmission within the spectrum [23] [47] [21].

Bluetooth is a technology that is in constant evolution, making the applications and the versions to be continuously changing through time. Different needs and different states in evolution require different approaches:

Bluetooth 1.0

The first Bluetooth, manufacturers often found difficulties in its interoperability and implementation.721 Kb/s is the usual rate for this version.

Bluetooth 1.1

Much alike the previous version in its transmission rates, some problems were solved and the *Received Signal Strength Indicator* (RSSI), the system that measures the received power strength was implemented.

Bluetooth 1.2

Faster connections and better interference immunity are the principal characteristics of this version. Transmission rate was also increased.

Bluetooth 2.0 + EDR

In this version the power consumption was optimized and the transmission rate was increased to 2.1 Mb/s. with the Enhanced Data Rate (EDR) feature. It is optional and may not be used in Bluetooth 2.0 if it is not needed and the rate is 721 Kb/s in this case.

Bluetooth 2.1 + EDR

More secure connections are the main characteristic of this version where transmission rates are the same as Bluetooth 2.0. A lower power consumption was also achieved for this version.

Bluetooth 3.0 + HR

Devices using this technology can achieve 24 Mb/s of transmission speed. These speeds can only be reached on devices compatible with the *High Speed* (HS) characteristic. Despite the evolution, this version is still compatible with the previous versions [84].

Bluetooth 4.0 (Low Energy)

The main advantage of this version is the power consumption. This version is the wireless technology that was used in this project and its features are explained in section [2.3.4] [20].

2.3.4 BLUETOOTH LOW ENERGY

Bluetooth Low Energy v.4.0 (BLE) is a technology whose utilization has been increasing in the last few years, mostly in fields like health or sports. A lot of new applications are possible with the inclusion of ultra-low power wireless chipsets. Most of the nowadays applications in those fields use significant amount of power and they usually need extra hardware to establish communications. This is the one big advantage of technologies like BLE.

Bluetooth Smart or Bluetooth Low Energy [46] is an emerging technology developed by the Bluetooth *Special Interest Group* (SIG) for short-range communications. Unlike previous Bluetooth technologies, BLE has been designed as a low-power solution control and this is the most important feature in this technology. BLE implements an entirely new protocol stack, new profiles and applications. Its main goal is to be able to have a long battery life using only a coin-cell.

As the other Bluetooth versions, this technology operates in the 2.4GHz ISM band with 40 channels spaced 2MHz apart and it is capable of transmitting at a rate of 1Mbit/s (absolute maximum, usually it is “only” 100 Kb/s) using GFSK modulation. Like the Classic Bluetooth it uses frequency hopping, but at a slower rate. Only 3 of the 40 channels are used to advertise which allow device discovery. These 3 advertising channels are located in different parts of the spectrum to provide immunity against the interference caused by Wi-Fi. Once the device is connected it uses the remaining 37 channels for data transmission [20].

Given that various technologies use the license-free 2.4 GHz band, an overcrowded radio environment may be found, decreasing the performance of the technologies due to the need of retransmissions. The robustness and reliability of BLE is made through Adaptive Frequency Hopping (AFH) as Golmie *et al.* demonstrate on their study – *Bluetooth Adaptive Frequency Hopping and Scheduling* [47].

BLE Protocol Stack

Like Classic Bluetooth, the protocol stack for this technology is composed by two main parts: the *Controller* and the *Host*. The Controller handles the lower layers of the stack responsible for capturing packets and the radio. Typically is implemented as a small *System-on-Chip* (SOC) with an integrated radio. The Host is responsible of handling the upper layers of the stack, the *Logical Link Control and Adaptation Protocol* (L2CAP), *Attribute Protocol* (ATT), the *Generic Attribute Profile* (GATT), the *Security Manager Protocol* (SMP) and the *Generic Access Profile* (GAP). Communication between the Controller and the Host is standardized as the *Host Controller Interface* (HCI). The non-core profiles, i.e., application layer functionalities, not defined by the Bluetooth specification, can be implemented on top of the Host [85].

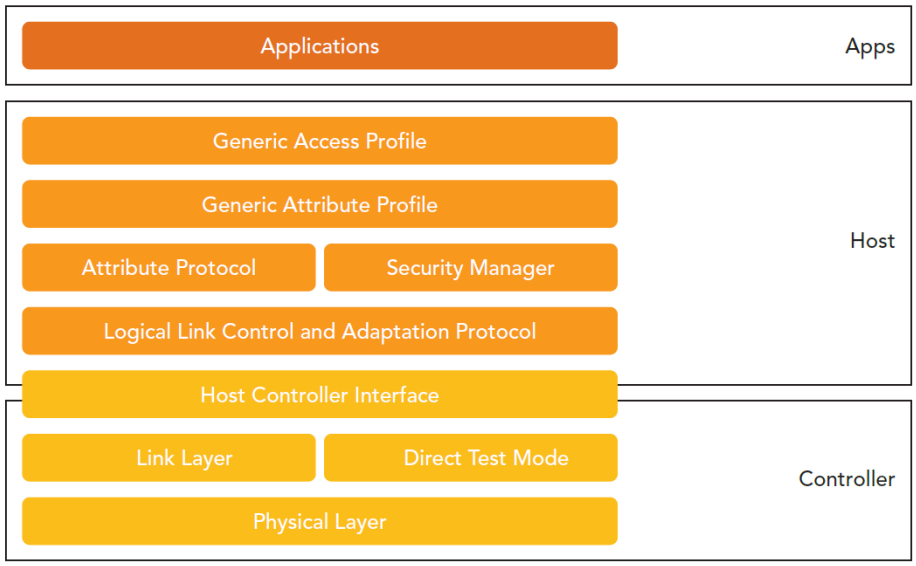

Figure 8 - BLE protocol stack

As is possible to see in figure 8, some of the BLE Controller features are inherited from the classic Bluetooth Controller, but they are incompatible. A device that only implements BLE (single-mode device) cannot communicate with a device that only implements Classic

Bluetooth. Many devices can implement both technologies and they are called dual-mode devices.

L2CAP - The communication between this layer and the Controller is made via the HCI. The main function of this layer is to provide data services to the upper layers, and to segment the packets into fragments for the controller. It is also responsible of reassembling packets from the Controller before they are submitted to the upper level layers.

GAP - The Generic Access Profile defines generic procedures and specifies device roles for service and device discovery, connection and security management. The GAP defines four roles with some requirements on the Controller: Broadcaster, Observer, Peripheral and Central. A Broadcaster only broadcasts data (through the advertising channels) and does not support connections with other devices and the Observer is its complementary, i.e., receives data transmitted by the Broadcaster. The Central role is the one designed for a device that is in charge of initiating and managing multiple connections, hence Peripherals are for single devices which use a single connection with a central device. A device can support several roles, but only one can be adopted at a given time. Additional profiles can be built on top of GAP. A new profile that specifies how applications can interoperate is called application profile.

ATT - It defines the communication between two devices acting as a server and a client, on top of a dedicated L2CAP channel. The server maintains a set if attributes. An attribute is a data structure that stores information that will be used by the GATT, the protocol that operates on top of the ATT. GATT determines the role (client or server) and this is independent of the slave and master roles. The access to the server's attributes is requested by the client, and this triggers response messages from the server. The server can send to clients two types of unsolicited messages also: notifications and indications. The difference is that notifications are unconfirmed and indications require confirmation by the client. Clients can write attribute values, by sending the appropriate commands.

GATT - This is a framework that uses the ATT for service discovery and exchange of characteristic attributes from one device to the next. Its interface with the application layer is made through application profiles. Characteristics are sets of data that include values and properties. All the data related to services and characteristics are stored in attributes. Generally there is an attribute table for each service that is going to be used by the profile.

For example a gyroscope sensor service may contain a gyroscope characteristic that uses an attribute for describing the sensor, other attribute for storing the data relative to the sensor and descriptors that specify the measurement units.

In BLE there are two types of packets: *Data* and *Advertise*, each with different lengths. Data packets can be as small as 80 bits or as long as 376 bits. This means that in a transmission of data packets, time can range between 80 microseconds, to 0.3 milliseconds. On the other hand, advertising packets' PDUs contain a 16 bit header and up to 31 bytes of data. Advertising packets can be sent within a range that goes from 20 milliseconds to 10 seconds. This specifies the interval between two consecutive advertising packets. The scanner can connect to an advertiser by finding these packets. The scanner has to be configured with a scan interval and a scan window and, after the establishment of the connection, the scanner provides the advertiser with two important parameters. Connection interval, that specifies the start time of connection events – the exchange sequence of data packets – and slave latency that specifies the amount of connection intervals a slave can ignore without losing the connection. This optimizes power consumption [20] [23] [26] [75].

Profiles

In order to use Bluetooth technology, a device has to be able to interpret Bluetooth profiles. Devices must be compatible with the subset of these profiles necessary to use the desired services. So, profiles are definitions of possible applications and specify general actions that devices can use or do to communicate with other Bluetooth devices. Profiles reside on top of the Bluetooth Core Specification.

A wide range of profiles describing many kinds of applications or use cases are available. They are designed for specific functionalities, for example the Find Me Profile that uses the Immediate Alert Service to perform an alert, there is a Heart Rate Profile that enables a Heart Rate Sensor to obtain data and then expose it to the Heart Rate Service, a Proximity Profile that enables proximity monitoring between two devices by using the Link Loss Service, the Immediate Alert Service and the Tx Power Service, among many others. They allow developers to create applications aimed specifically to the features each profile implements using predefined attribute/value pairs of information that can be found in each

profile. Profiles provide standards which manufacturers follow and this allows the interoperability among devices that use Bluetooth.

Bluetooth SIG defined many profiles for Low Energy. The GATT profile [48] is a general Bluetooth specification in order to send and receive short pieces of data – attributes – over a Low Energy connection. Currently, all LE application profiles are based on GATT, but many individual devices can implement more than one profile, obviously. A device can implement a Proximity Profile and a Gyroscope Profile, for example [17] [75].

A Comparison with Bluetooth Classic

Bluetooth technology was originally designed for continuous, streaming data applications and successfully eliminated wires in areas such as medical and industrial applications, but also in consumer applications. Classic Bluetooth will continue to provide robust wireless connections, but if the goal is power saving, Bluetooth Low Energy is the technology to use. Whereas Bluetooth Classic was developed with faster data rates in mind, BLE aims to be able to achieve lower power consumptions by the devices. It is not designed to stream large amounts of data, instead it is designed to periodically send short bursts of data [20] [21].

BLE reduces significantly the Classic Bluetooth's peak, average and idle mode power consumptions, with energy efficiencies that can be several times higher than the Classic Bluetooth. These low consumptions enable BLE chipsets to work with a small battery for a year or more in some cases. The simple Link Layer of BLE allows this saving in energy consumption by making quick connections, BLE chips spend most of the time asleep, only waking up to send data, and since this process only takes a few milliseconds, energy will be saved. Classic Bluetooth requires something like 100 milliseconds [20] .

BLE devices can be one of two types: single mode devices that can only support Bluetooth Low Energy, and dual mode devices, that are fully compatible with previous versions of Bluetooth. Dual mode devices can perform high data rate streaming, but cannot benefit from the low consumption features that can be provided only by using BLE low data rate modes. BLE can't replace completely previous Bluetooth versions, but can allow new use cases and applications [21] [26].

There are a lot of Classic Bluetooth features adopted by BLE. AFH, as well as part of the L2CAP are used by BLE, and it also implements the same link security with simple pairing modes, encryption and secure authentication. This inheritance allows BLE to be secure, robust and reliable in tough environments [85].

Table 1 resumes Bluetooth Classic and BLE characteristics.

Table 1 - Bluetooth Classic vs BLE

Specification	**Classic Bluetooth**	**Bluetooth Low Energy**
Frequency	2400 to 2483.5 MHz	
Range	100 m	50 m
Number of channels	79	40
Data Rate	1 – 3 Mbps	100 Kbs
Nodes	8	Unlimited
Voice Capable	Yes	No
Total time to send data	100 ms	3 ms

Besides those characteristics, both technologies support service discovery and have the profile concept implemented. Their use cases are basically the same, with BLE more focused on health and sports environment, because of the lower power consumption devices. From table 1 it is possible to conclude that BLE has its limitations also. One of them is the data transfer rates than can be several times less than the Bluetooth Classic, so for that reason streaming Bluetooth Low Energy connections will lose some of its low consumption capabilities as the utilization approaches the continuous transmission. So, in other words, the choice between classic and low energy has to be made depending on the use case that is going to be given to the final application using one of these technologies [20].

3. MOBILE OPERATING SYSTEMS

The way sensors are exploited on a smartphone is directly related with the phone's operating system. A mobile operating system is an OS built exclusively for a mobile device like smartphones, PDAs, tablets or other digital mobile devices. It is responsible for identifying and defining mobile functions and device features, including dialing, application synchronization or text messaging, between others. We can say that a mobile OS is similar to a standard OS, like Windows, Linux and Mac, but with other primary concerns, like the wireless variations of local broadband connections, mobile multimedia, the hardware management and the optimization of the application software's efficacy in the device. In order to adapt to different mobile environments, mobile operating systems combine the features and the operating system of a standard personal computer, with a touchscreen, Bluetooth, GPS, dialer and other device features. It runs focusing its resources to emphasize communication, such as *Random Access Memory* (RAM), storing and CPU speed [5].

Unlike personal computers, it is usually not possible to install a different operating system on a smartphone. If the user doesn't like the one that came preloaded, the best option is to buy a new one. So the choice of an Operating System is much more critical and will reflect

the user's lifestyle because it will determinate the choice of apps and phone functionality. Nowadays, from the operating systems already settled in the market, it is possible to distinguish two types: the open source mobile operating systems, led by Android, and the proprietary operating systems like Apple's iOS, Windows Phone or Blackberry OS.

The main difference between these two types of mobile operating systems is the user customizability. It is possible to change almost everything in Android for example, but in iOS, the user has a very limited change capability. The kernel, user interface and some standard applications are available to anyone who wants to use them in Android's case, but only the kernel is available in iOS.

In order to be able to test the fully capabilities of the sensors (in this particular case we are more interested on the sampling frequency) on a mobile operating system, the OS needs to be open source. This means we need access to the source code and to have total freedom over the device and over the software. The open source mobile operating systems already on the market are Android and Firefox OS. There are some upcoming open source systems like Ubuntu Touch and Tizen that are expected to be on the market relatively soon.

3.1. ANDROID

This mobile operating system has been dominating the world smartphone market for the last years and it seems that this domination will continue at least for the next few.

Android is an open source, Linux-based, operating system designed primarily for touchscreen devices like smartphones, tablets and TVs. The main purpose of Android is to create an open source platform available for carriers, manufacturers and developers, in order to create a successful developing "community" that improves the mobile experience for end users [53].

In 2005, Google bought the OS and the first Android-powered phone was sold in October 2008. The code is released under the Apache license [54]. This license allows device manufacturers, wireless carriers and developers to modify the open source code and to distribute it. This fact is determinant for the number of existing applications and custom ROMs available. Android is the software of choice for technology companies that want a low-cost and customizable, lightweight operating system for their devices. Despite being

primarily designed for smartphones and tablets, many other applications on televisions, digital cameras, among other electronic devices, had been seen using Android [55].

Figure 9 - Android's home screen

The User Interface (UI) in Android is based on direct manipulation and it is very intuitive. The touch inputs are somewhat similar to real-world touches, which give users the feeling of real gesture-based actions. Sensors like accelerometers or gyroscopes are used by some of the core applications to respond to user actions, for example changing the orientation between portrait and landscape, or simulating the steering wheel in a race car game. It is possible to personalize home screen icons and applications, add widgets, among other features. The home screen is highly customizable allowing the users to change for a different operating system look if they want it to. As Android is largely used around the world, there is an "infinite" number of applications developed that can be acquired in Google Play and on other 3rd party app stores [56].

Google is giving updates to Android every six to nine months, with most of the devices receiving them over the air. This gives users a new feel every now and then, keeping users interested and waiting for the new update. But unless the user has one of the Nexus devices, these updates can be slower and often arrive months later from the date of the official release. This occurs mostly because of the extensive variation in hardware of Android devices.

If the developer community continues to work together in the next years, this platform will surely continue to evolve.

3.1.1 ARCHITECTURE

Android is divided into five core layers (figure 10). It is important to understand how Android works at a high level if the user/developer wants to port Android into a new hardware. For this to succeed it is also important to understand how the hardware drivers and the *Hardware Abstraction Layer* (HAL) interact with each other and with the many layers of the Android operating system.

The HAL serves as a standard interface that allows the Android system to communicate with the device driver layer without knowing about the lower-level implementations of the drivers and hardware. The correspondent HAL must be implemented for a specific piece of hardware, which means that the contract defined in each hardware specific HAL interface must be respected, to correctly interact with the hardware. These implementations are usually built into shared library modules (`.so files`).

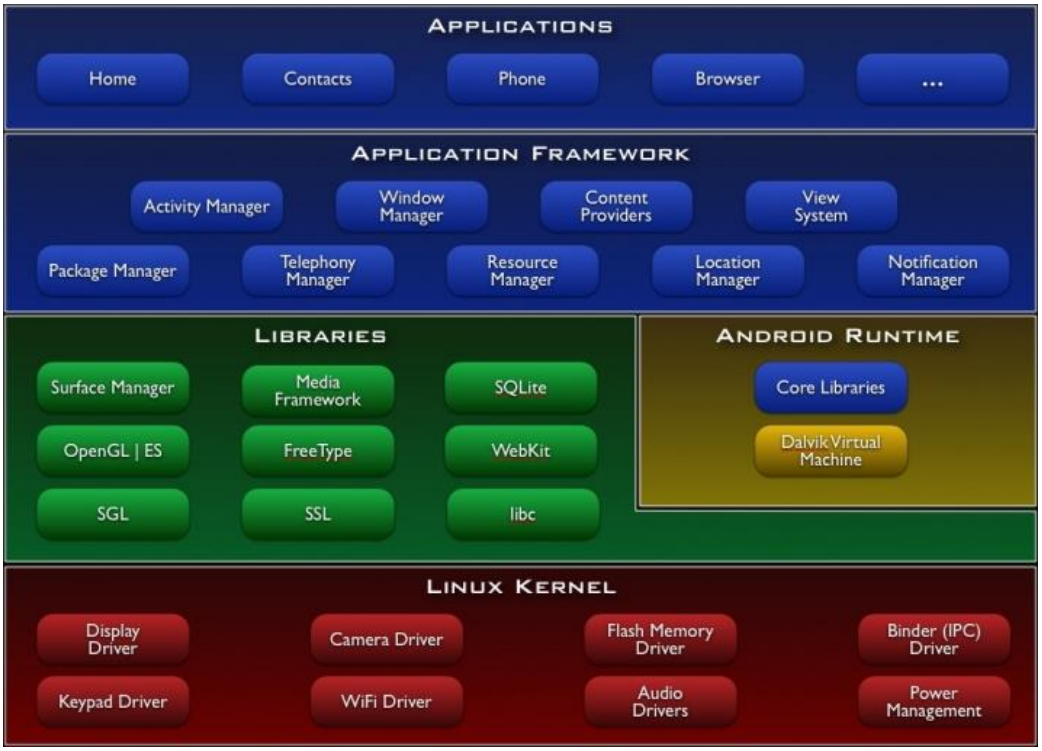

Figure 10 - Android architecture

Through application framework, Android offers developers the possibility of building extremely rich and innovative applications. They are able to access almost anything on the

device like the hardware, location information, set alarms or run background services, among many other things. One of the main Android advantages is that it does not differentiate between core applications and 3rd party applications, which means that any developer can build an application that can access all the phone's capabilities and services. Developers have full access to the same framework APIs used by core applications.

Underlying to all the applications is a set of system services that control the behavior of each application. Content providers enable applications to access data from other applications, or to share their own data, for example, while *Resource Manager* provides access to non-code resources like graphics, strings and layout files. The managing of the application lifecycles is made by *Activity Manager*.

Android also includes a C/C++ set of libraries that allow developers to expose the capabilities of the various components, through the Android application framework. Libraries like *SQLite* that is a powerful database engine available to all the applications and *Surface Manager* that manages the access to the display subsystem and composites 2D and 3D graphic layers from multiple applications, are just two of the core libraries that can be used to create powerful applications.

Android Runtime provides a set of core libraries of the Java programming language that allows to access most of the devices functionalities in this language. Every Android application runs its own separate process, with its own instance of the *Dalvik Virtual Machine*. This virtual machine has been written so that multiple instances of the VM run simultaneously and efficiently. The Dalvik VM executes files in executable format (`.dex`) that is optimized for minimal memory footprint. The VM is register-based and runs Java classes that have been transformed into the `.dex` format by the "dx" tool. The virtual machine relies heavily on the Linux kernel for features such as threading and low-level memory management. It is not possible to run Java class files on Android, they need to be converted to Dalvik bytecode format first.

Finally the Linux kernel relies on a 2.6 kernel for the device's core services like security, network stack and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack. It includes some additional features such as *wakelocks* or memory management to better fit in mobile environments [57].

3.1.2 DEVELOPMENT ENVIRONMENT

There are a lot of resources and libraries that can be used, giving developers the opportunity of making useful and practical applications.

Applications are usually developed in Java language using Android *Software Development Kit* (SDK). This SDK includes a set of tools like a debugger, software libraries, a handset emulator based on QEMU, documentation, sample code and tutorials. To start developing applications it's recommended the Eclipse *Integrated Development Environment* (IDE) using *Android Development Tools* (ADT) plugin [58].

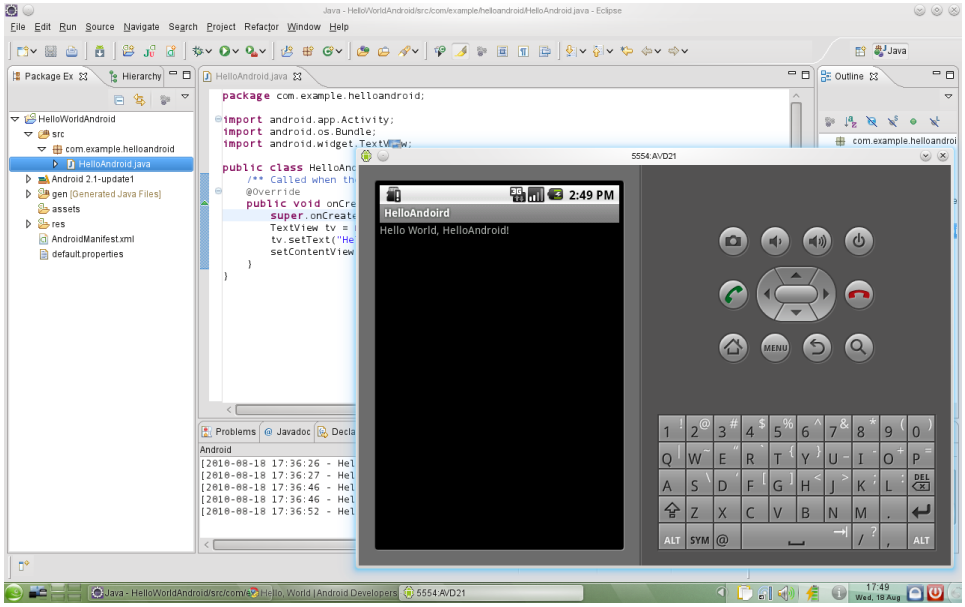

Figure 11 - Eclipse IDE

ADT is a set of components which extend the Eclipse IDE with Android development capabilities. Contains all required functionalities to create, compile, debug and deploy Android applications from the IDE. It provides specialized editors for resource files such as layout files. These editors allow switching between the XML representation of the file and a user interface via tabs at the bottom of the editor.

Applications can be tested on a real device or using an emulator. To use the emulator it is necessary to create *Android Virtual Devices* (AVD). To connect to either a real device or a virtual device, the SDK contains *Android Debug Bridge* (ADB).

If the developer wants to create an application or extension in C/C++ he can use *Native Development Kit* (NDK). This is a set of tools that allow the implementation of parts of the

application using native-code languages such as C or C++. For certain types of apps this could be crucial and helpful to reuse some of the existing native code libraries, but the majority of the applications do not need the Android NDK.

Applications written in native code may sometimes result in a performance improvement because does not use the Dalvik Virtual Machine, but most of the times will just increase the complexity of the application with no clear benefits. In general, developers should only use NDK when it is essential for the application and not just because they prefer to write in C or C++. Maybe applications such as signal processing or physics simulation should use the NDK, because they are self-contained, CPU intensive applications that don't allocate much memory, so they might be the best candidates to this technology. First, developers should check in the framework APIs provided by Android if there is a way to do the application without using the NDK, and use it only when it is not possible [59].

3.1.3 TESTING ANDROID

It is easy to test and put the hands on a device running Android, because there is so much handsets in the world running this Operating System. But it is also possible to test the OS on an emulator or building a specific version for a specific device. This can be made with the *Android Open Source Project* (AOSP) [41].

It is possible to build an Android version and the kernel for a specific device, as well. Operating systems that can build a specific ROM are Ubuntu and Mac OS. For Gingerbread (2.3x) or newer, a 64-bit environment is needed. A powerful desktop or laptop is recommended because the building process consumes a lot of RAM memory and will need at least 30 GB for a single build. These builds can be made for a variety of devices, including Pandaboard, Samsung Galaxy Nexus, Google Nexus 10, the emulator and other target devices. They usually tend to behave very similar to the final Android product released to the market for a specific version. Some features might not work so well because of proprietary drivers and the OS might look different than some of the final products because of the front end interfaces developed by the manufacturers.

Custom ROMs can be installed, giving users a new experience. They are very popular, because they modify the system searching for memory and performance improvements. To install these ROMs it is necessary root access to the devices. CyanogenMod [24] is probably the most popular custom ROM in the world, but it is possible to find thousands of

them around the internet as the Android's source code is open and anyone can change it. A more "free" user experience is given to the user with this ROM, because it overcomes some of the chains of the Google's Android. Using a custom ROM usually results in more frequent updates that could fix bugs and introduce new features, because developers behind it don't have the same bureaucracy and procedures as the manufacturer and carriers have. Custom ROMs are usually faster, more efficient and use less memory, because developers tend to rip out the "garbage" such as carrier installed apps and they optimize the kernel, allowing, for example to extend battery life. Most of them allow to install applications directly on the SD Card, so if the device has run out of space, there is still this very useful option. On stock ROMs (version of the operating system that comes, by default, with the phone) this is not possible. Installing these custom software have downsides too, something could go wrong and this maybe can cause the loss of warranty, so, in the end, the user has to choose whether he wants to install something like this or not.

3.1.4 SENSOR FRAMEWORK

In Android, it is possible to access sensors available on the device by using Android sensor framework. This framework is a part of `android.hardware` package and provides several classes and interfaces that help developers perform a variety of sensor tasks. Tasks like defining the rate at which the data is acquired and register or unregister sensor event listeners that monitor sensor changes, are possible using this framework.

Sensors can be hardware based or software based. Hardware-based sensors are physical components built-in the devices and measure specific properties like acceleration, temperature or angular change. Software-based are not physical components and derive their data from one or more hardware-based sensors. They are often called virtual sensors. Linear acceleration or orientation sensors are just two examples of software-based sensors [59].

Table 2 describes the most important classes of the Android sensor framework:

Table 2 - Classes of Android Sensor Framework

Class	**Function**
`Sensor`	Creates an instance of a specific sensor.
`Sensor Manager`	Creates an instance of the sensor service; Provides methods for listing and accessing sensors; Sets data acquisition rates and calibration parameters.
`Sensor Event`	Gives information about an event that occurred because of changes in sensor data; Includes raw sensor data, timestamps and the type of the sensor that generated the event.
`Sensor Event Listener`	Creates two callback methods that are capable of receiving event notifications when sensor data changes.

A lot of public methods can also be used by Android applications in order to get sensor capabilities. Resolution, power consumption and manufacturer, are just some examples of information that developers can get by using those public methods.

Table 3 resumes the most important methods.

Table 3 – Android Sensor Public methods

Method	**Returns**
`getName ()`	Name string of the sensor
`getPower ()`	The power in mA used by this sensor
`getResolution ()`	Resolution of the sensor in the sensor's unit
`getVendor ()`	Manufacturer string of the sensor
`getMaximumRange()`	Maximum range of the sensor in the sensor's unit
`getMinDelay ()`	The minimum delay between two sensor events in microsecond

The data delay (sampling rate) controls the interval at which sensor events are sent to the application via `onSensorChanged ()` callback method. This delay is very important and has to be defined accordingly to the application requirements. By default there is four

predefined delays (`GAME`, `UI`, `NORMAL` and `FASTEST`). The `SENSOR_DELAY_FASTEST` is the fastest rate at which sensor events are sent to the application. In the new APIs (level 11 and newer) it is also possible to define manually the desired delay, in microseconds. This defined delay is just a suggested value. The system can modify this delay. A smaller delay imposes a bigger load on the processor and uses more power, so developers must choose the larger delay that still meets the application requirements. Once the delay is set it is not possible to change it. The only way to do that is by unregister and then re-register the sensor listener [59] [57] [19].

Needless to say, the applications made to use sensors cannot be tested in the emulator. The emulator cannot emulate sensors. Physical devices are the only option to do that.

Sensor data flow in Android can be seen in figure 12.

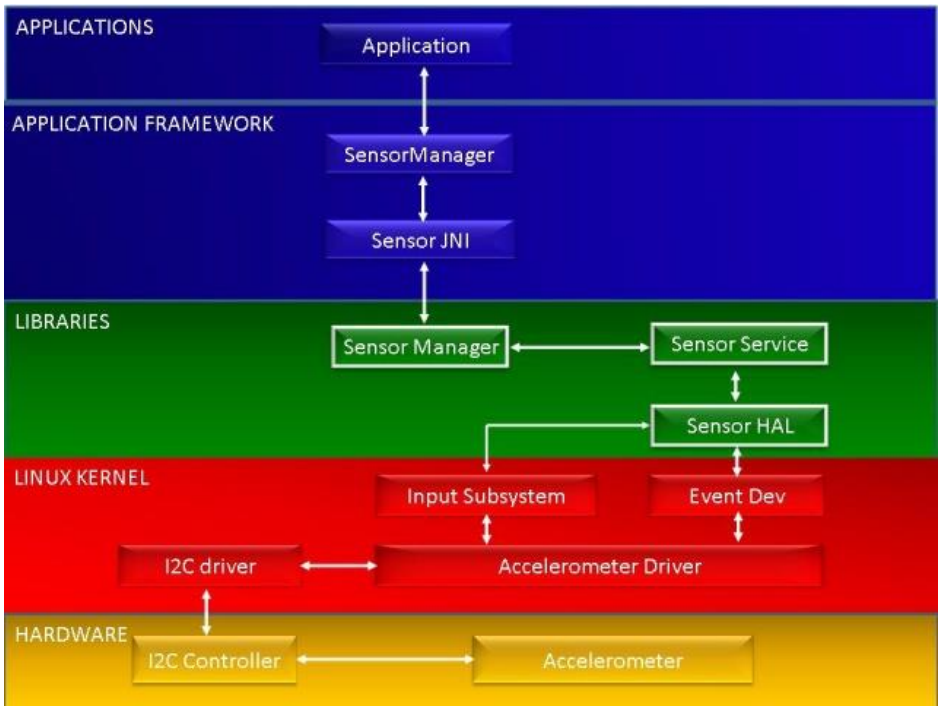

Figure 12 - Android sensor data flow

The sensor represented in the figure 12 is the accelerometer, but the flow is almost the same for other sensors. The applications use the sensor application framework to get the sensor data. After the instance of the accelerometer, in this case, being created, the application communicates with the C++ layer through sensor *Java Native Interface (JNI)*. The middle layer basically consists of Sensor Manager, Sensor Service and Sensor HAL. These libraries define some methods and functions that will correlate both parameters from

the application and data from the lower layers through kernel. This kernel includes the sensor driver and through a computer bus, such as I^2C, can communicate directly with the hardware sensor [28].

The main difference in sensor implementation is the type of event generation and data gathering. While proximity sensor, for example, is interrupt-based, almost all other sensors are poll-based. Interrupt-based sensors only give events when the data changes. If the sensor data keeps the same value for a long time, an event will never be sensed on that amount of time. The values on those sensors are generally not required to be read quickly. The device will behave perfectly even when no values are returned. Those sensors are not usually used for games and other applications that require a lot of values during a short interval of time. They are used mostly for screen luminosity control or for device information.

Poll-based sensors are polled for at regular intervals of time, selected by the delay parameter used while registering a Sensor Event Listener. Sensors like accelerometers, gyroscopes or magnetic field sensors belong to this category. They provide quick information that will allow applications like games, to behave accordingly to the user's gestures, for example [19].

3.2. FIREFOX OS

Firefox OS (project name: *Boot to Gecko* (B2G)) is an open source operating system, in development by Mozilla Corporation. It uses a Linux kernel and boots into a Gecko-based runtime engine, which lets users run applications developed entirely using HTML5, JavaScript, and other Open Web Application APIs. Mozilla aims to fully support HTML5, using "open Web" technologies, rather than the platform-specifics native APIs. The main idea is to have all user-accessible software running on the phone be a Web application that uses advanced HTML5 and device APIs to access the phone's hardware directly using JavaScript. That way it is possible to remove some of the steps of software-hardware communication and the overall process will be quicker [61]. The smartphones with Firefox OS will be an environment similar to the one it is possible to see in figure 13.

Figure 13 - Firefox OS appearance

HTML5 has the ability to communicate directly with the hardware. For anyone who wants to use the camera, for example, the OS just opens a website that uses HTML5 to access the camera. This means that every website can use the camera feature of B2G [86]. On Facebook, for example, it's possible to share a photo directly, instead of take, store and then share as usually happens. Giving access to hardware layer allows to remove a lot of steps. Another advantage to Mozilla is that there are a lot more web developers than app developers, which means that a lot more applications can be developed and shared, and, with Firefox OS (FFOS) being totally open for new app development, this could be a massive change in the smartphone world.

It can be said that the main goal for FFOS is not to compete with the high-end smartphones, but to offer entry-to-mid-level smartphones at feature phone prices. It aims the called feature phones, but they want to turn them into more smartphone-like with their applications. It will be strictly low-end, at least for the first few years. Mozilla's goal is not to compete with Android and iOS, but to help people migrate from feature phones to smartphones. To run FFOS an 800 MHz processor and 256 MB of RAM are required, which is remarkably lower than some of the requirements of other Operating Systems [61].

3.2.1 ARCHITECTURE

This Operating System's architecture is composed of three main components: *Gonk*, *Gecko* and *Gaia*.

Gonk is the lower-level operating system of B2G and consists on a Linux kernel and a userspace Hardware Abstraction Layer. The kernel and a lot of userspace libraries are common to open-source projects like *Linux, Libsub, Bluez,* etc. Some other parts of HAL are shared with Android project like GPS, camera, between others. It is possible to say that Gonk is an extremely simple Linux distribution.

Gecko is a free and open source layout engine used by many applications developed by Mozilla, like Firefox browser, and other open source software projects. It's the application runtime of B2G. At a high level, Gecko implements the open standards for HTML5, CSS and JavaScript and makes those interfaces run well on the Operating Systems that it supports. Roughly, it is the Firefox browser with the new Web APIs and without the user interface.

Finally **Gaia** is the User Interface of B2G. Everything that is drawn after the startup of B2G is related to Gaia. It's written entirely in HTML5, CSS and JavaScript. Its only interface to the underlying operating system is through Open Web APIs, which are implemented by Gecko [60].

In figure 14 is it possible to clearly distinguish these three fundamental layers of Firefox OS.

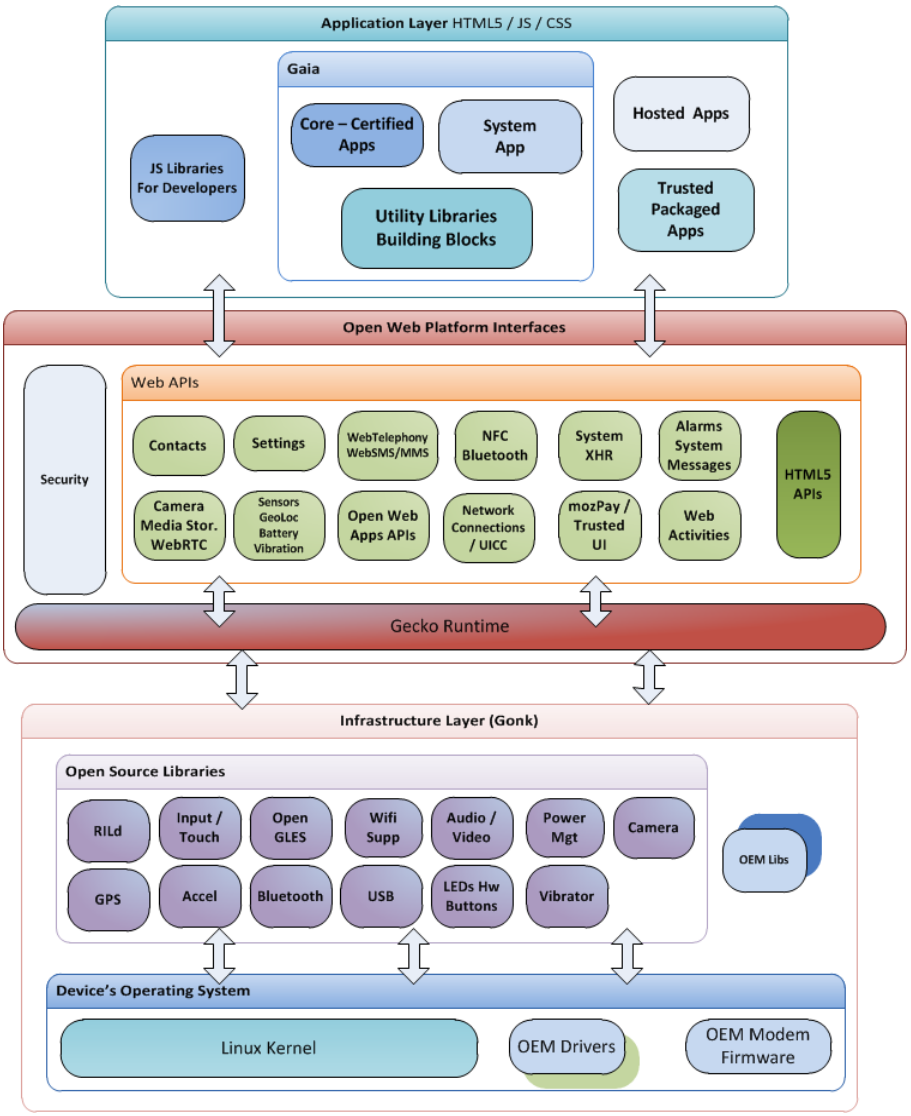

Figure 14 - Firefox OS architecture

At the moment of this dissertation Firefox OS were still a pre-release product, so the architecture described here is not necessarily final and some things could be changed afterwards.

3.2.2 DEVELOPMENT ENVIRONMENT

Finally, with Firefox OS, web developers get exactly what they expected for a long time: a mobile environment that can be totally created only with HTML5, CSS and JavaScript. Mozilla is working to create applications that are backed entirely by the open Web. Because of this fact there is no specific IDE for this Operating System. Almost every IDE can be used to develop HTML5 applications. Web is the platform to use. Open web apps are considered to be a great opportunity for all developers who want to port their applications into a large number of devices, not being held to a specific vendor platform

(such as Android or iOS). The main difference between a web application and a website is that the application can be installed and integrated to the device, and not being just a "bookmark", means that a web application becomes part of the device's system. *Open Web Apps* (OWA) are intended to be, in the future, working in all browsers, operating systems and devices. There are parts of OWA technology right now, that are standardized (HTML5, CSS, JavaScript, etc.), while other parts are still not yet standardized, and the Mozilla implementation is specific for Firefox OS and to other Mozilla technologies. They clearly identify those not standardized parts, and they want to make a total standardized environment in the future.

Mozilla has been built an app system that lets users buy an application once and run it on all their HTML5 devices. This is made through a receipt that is acquired when the user purchases an application. The receipt is a JSON Web Token with metadata that links to the Marketplace's public key and it verification service URL, and it is a verifiable proof or purchase, so the application is intended to be portable across any device that supports Open Web Apps system.

There are two types of applications in Firefox OS: packaged and hosted apps. The first ones are just a zip file containing all the important content like HTML, CSS, JavaScript, images and manifest, etc. Hosted apps are basically like a website, they run from a server at a given domain. Either way, a valid manifest is required. To list them in Firefox Marketplace, the user will either provide the URL where the hosted application is located, or upload the application as a zip file [87].

Regarding the applications layout, responsive design has become more and more important as the screen resolution varies between different devices. Even if the applications are intended to run only in Firefox OS devices, it's important to notice that as Firefox OS can be installed on a different variety of devices, they could use different screen resolutions and this is a matter that should be taken into consideration when developing an application. CSS media queries provide means to adapt the applications to different types of resolution.

As JavaScript APIs are being improved day by day, Mozilla's Web API brings several standard mobile features to JavaScript APIs. Features like accessing battery or vibration are just some examples. But there are some WebAPIs that are available but require

permissions for that specific feature to be enabled. Apps can register permissions in the `manifest.webapp` file.

3.2.3 TESTING FIREFOX OS

There are some simulators like B2G Desktop or B2G emulators that are a bit hard to configure and most of the time are significantly different from Firefox OS on a real phone. So Mozilla created Firefox OS Simulator, an experimental prototype test environment for Firefox OS.

This simulator is nothing more than an add-on for the Firefox browser. Its installation is as easy as any Firefox add-on. The simulator launches with a JavaScript console so that developers may debug their application from within the simulator [62].

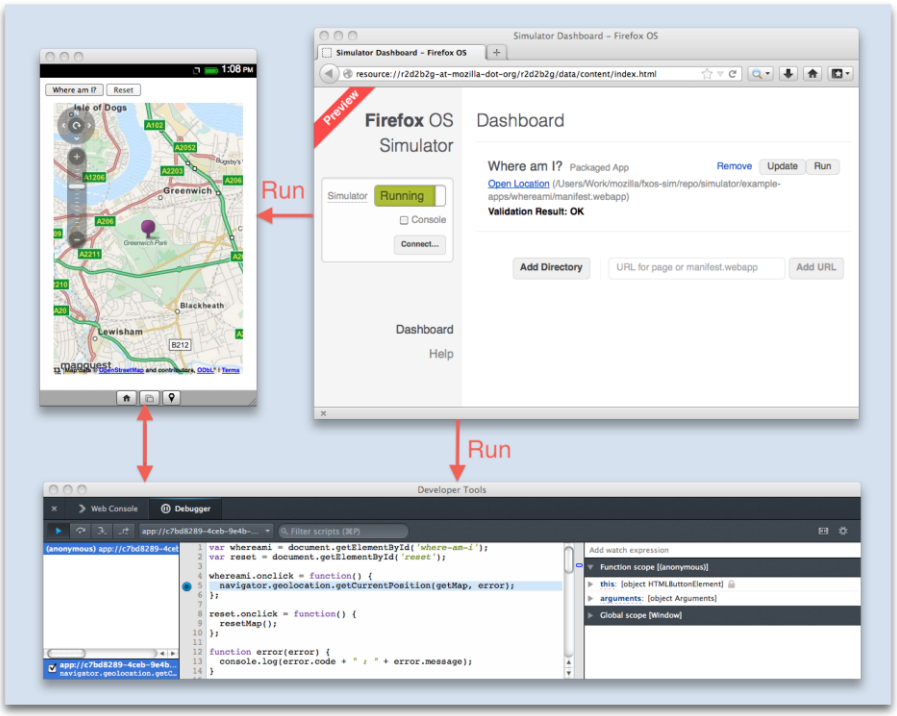

Figure 15 - Firefox OS simulator

It is possible to add applications to the simulator by providing a URL to a website, or even better, the path to a manifest file, that could be either a URL or a path to a local directory. If the developer uses the last option, he creates a packaged app on the local computer and no web server is needed. Firefox OS simulator does not create a virtual computer. It is a version of B2G project that is built for operating systems. This allows the simulator to run very quickly, with minimal startup time, which is clearly an advantage for developers. It includes the Firefox OS desktop client, which is a version of the higher layers of Firefox

OS that can run on desktop. Simulator may not be a perfect real environment simulation, but it is a very useful way of testing Firefox OS apps.

Apart from the simulator it is possible to test Firefox OS on a real device. To connect the device to the simulator ADB is needed and remote debugging must be enabled in the developer options of the device.

Currently, there are three main targets for development. The primary target will typically be the first to receive feature updates and bug fixes, devices like *Pandaboard*, the emulator or GeeksPhone's Keon and Peak are just some examples of this group. The second target group of devices are the functional ones and are supposed to be especially used by app developers. They will receive updates and fixes secondarily. Samsung Nexus S and Samsung Nexus S 4G are the devices in this group. Finally builds can be made to Samsung Galaxy S2 and Samsung Galaxy Nexus, although these devices are not being worked on a regular basis by core developers. Their reliability and features may be well behind the other two targets and these devices are just being used to test the operating system most of the time [63].

So, there are some tools and requirements that the system that will build Firefox OS must have. ADB and *fastboot* for example, as well as an "*android.rules*" file, where the user will specify to which devices will try and send the build via USB. Other things like GCC 4.6+ or bison are also needed.

3.2.4 SENSOR FRAMEWORK

Standard Web APIs and Firefox OS specific Web APIs are used to access sensors in Firefox OS. *Document Object Model* (DOM) events allow languages like JavaScript to register various event handlers or listeners. Device Orientation specification defines several new DOM events that provide information about the physical orientation and motion of a hosting device.

When an event happens, the browser sends the event to the related element. If that element has a handler (function) for the event, it gets called with related event info and it gets an event object as its first argument, which has a lot of fields that describe about the event. Mozilla Event Reference is used to help developers to track events from the sensors or other parts of the Mozilla application code.

This specification provides three events and the information is not raw sensor data, but rather high-level data that is not known to the underlying source of information. Those events are `deviceorientation`, `devicemotion` and `compassneedscalibration`. The latter is not implemented in Firefox OS [64].

The basic use of events is made by adding an event listener for the specific event:

```
window.addEventListener ("deviceorientation", function (event) {
  // process event.alpha, event.beta and event.gamma}, false);}
```

Device Orientation Event

The user agents that implement this specification must provide a new DOM event, named `deviceorientation`. The event occurs whenever a significant change in orientation is detected. For most implementations, a change in one degree is sufficient to fire this type of events, while other implementations may also fire the event if they have reason to believe that there is no sufficiently fresh data available.

There are four attributes that can be retrieved when a `deviceorientation` event occurs. `Alpha`, `beta`, `gamma` and `absolute`.

The first three of those attributes must specify the orientation of the device in terms of the transformation from a coordinate frame fixed on the Earth to a coordinate frame fixed on the device. East (X) is the ground plane, perpendicular to the North axis and positive towards east. North (Y) is the ground plane and positive towards true North and Up (Z) is perpendicular to the ground and positive upwards. For most mobile device, the coordinate frame is defined relatively to a screen orientation, usually portrait. This means that even the orientation of the screen changes, this does not affect the orientation of the coordinate frame relative to the device. That said, alpha is the rotation degrees around Z axis, beta is the rotation degrees around X axis and gamma is the rotation degrees around Y axis. Those angles form a set of intrinsic *Tait-Bryan* angles [9] (X-Z'-Y'') that do not match the roll, pitch and yaw convention used in vehicle dynamics. This also means that alpha is the opposite of a compass heading. The absolute attribute is a `Boolean` that is set to true if the orientation provided is the difference between the device's coordinate frame and the Earth's coordinate frame. If not, this parameter is set to false [65].

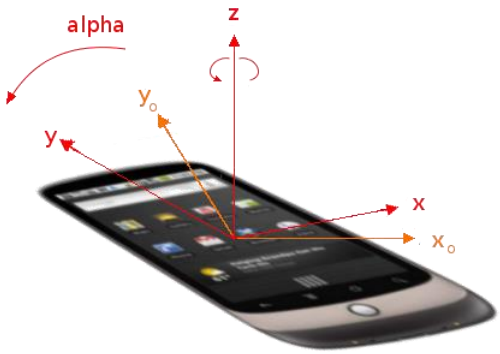

Figure 16 – Alpha in Device Orientation Event

Device Motion Event

This event is triggered when a significant change in motion is detected and returns four attributes: acceleration, acceleration including gravity, rotation rate and interval.

Acceleration must be initialized with the acceleration of the hosting device relative to the Earth frame. It must be expressed in m/s^2. Implementations might be unable to provide acceleration data without the effect of gravity, may supply the acceleration including gravity value. This is less useful in many applications, it provides a best effort support. This attribute is initialized with the acceleration of the hosting device, plus and acceleration equal and opposite to the acceleration due to gravity. This value is also in m/s^2.

The rotation rate is initialized with the rate of rotation of the hosting device in space. It is expressed as the rate of change of the angles defined in `deviceorientation` event and must be expressed in deg/s.

The interval attribute is the interval at which data is obtained from the underlying hardware and must be expressed in milliseconds. It should be a constant in order to simplify the filtering of the data by the application. There is no access to magnetometer or to other useful virtual sensors such as altitude, which is clearly a flaw that can be rectified in the future.

Sensor API specification provides several DOM events that allow applications to obtain information given by the various sensors available on the device. This information, unlike Device Orientation specification, is raw sensor data. This specification is aimed at covering the most common sensors found in devices. A different DOM event will be fired for each sensor type that will supply with physical raw sensor data. Like Device Motion and Device Orientation, each sensor will turn on when the listener is added and turn off when the last event is removed. The corresponding event must be fired when there is a change in the sensor data. If the user agent does not support the sensor wanted, then the event will not be fired. The event handler provides the following signature:

```
Interface SensorCallback {
    void ondata (double value, double min, double max);};
```

The `ondata` callback is lifted when there is sensor data to report, being the value the sensor data. Min and max are the minimum and maximum ranges that the sensor can detect [64].

Light and Proximity Events

There is a set of DOM events for sensors that the device supports and they define which sensor will be listened to be able to report data. Those events can be temperature related, humidity related, among others, but in Firefox OS only light and proximity events are specified in the API, so these are the only events that can retrieve these kind of values at the moment.

Using the `devicelight` event is possible to verify the level of ambient light in lux. It provides developers with information from photo sensors or other sensors that can detect ambient light levels of the surrounding environment. This is mostly used for adjusting the luminosity of the device.

The `deviceproximity` event gives information about the distance of a nearby object using the device's proximity sensor. This event is usually capable of returning three values: the maximum and the minimum sensing distances that the sensor is able to report, along with the current device proximity in centimeters [64].

3.3. UBUNTU TOUCH

Canonical's Ubuntu Touch was developed around Android's kernel, but with no use of Java Virtual Machine, as Android does. It's built around Qt5 and QML and the Ubuntu Touch Developer Preview was released on Feb 21st, which is intended to be used for evaluation and development purposes only. It does not provide yet all the features and services of a retail phone. This version is still in development, but Canonical is making efforts to evolve this platform quickly. It is expected that the first devices with Ubuntu Touch are expected to be released at the end of 2013. Canonical claims that the mobile version is going to be fully compatible with the PC and TV version, which allows more interaction between devices. Figure 17 shows the home screen and the sidebar of Ubuntu Touch.

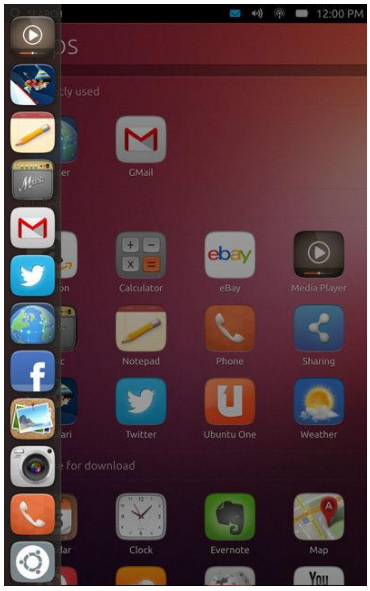

Figure 17 - Ubuntu Touch home screen

One big difference between Ubuntu Touch and Firefox OS is that it not only targets the low-to-midrange market, but also the high-end smartphones, in the future.

A 1GHz cortex A9 processor will be required for entry-level Ubuntu phones, with at least 512 MB of RAM, while higher-end models aimed at doubling as PCs and will require a quad core A9 or Intel Atom processor and at least 1GB of RAM [66].

3.3.1 ARCHITECTURE

Ubuntu Touch is based on CyanogenMod, a custom ROM for Android, which is an open source replacement firmware for smartphones and tablets based on Android operating

system. With this custom ROM, the user is offered with features and options not found in the official firmware distributed by vendors of these devices.

The OS uses some of the code pulled straight from CyanogenMod repositories, so porting Ubuntu Touch into new devices could be as easy as porting CyanogenMod to any device it supports. It does not use AOSP because some of the work had been done by the project and they are borrowing it, since it is open source and based on Linux distributions.

In order to support a wide range of devices, Touch's architecture reuses some of the drivers and hardware enablement available for Android, so some of the Android's services are constantly running at the device along the Ubuntu file system, living separately, in a container. So porting into these new devices could be quite simple, because the Ubuntu Interface is running in this container and can be easily accessed.

The components that are being used from Android are:

- A Linux kernel, with some modifications to support extra features needed by Ubuntu;
- OpenGL ES2.0 HAL and drivers;
- Audio/Media HAL and services in order to reuse the hardware video codecs;
- Rild modem support.

It's possible to say that Ubuntu "just" uses kernel (modified), HAL and *SurfaceFlinger* display manager from Android. As this OS is running in a separated container on top of Android kernel and services, the communication between them happens via *Binder, Sockets* and *libhybris*. Hereupon, other than the basic services needed to reuse binary blobs available, every other feature is made by Ubuntu [88].

3.3.2 DEVELOPMENT ENVIRONMENT

One of the biggest issues Canonical claims other operating systems to have, is the overhead inherent to them. Android, for example, relies heavily on a Java Virtual Machine for executing its applications. It really allows easy cross-reference compatibility, but the performance is damaged by this fact. Ubuntu Touch is different: code runs natively on the ARM-architecture processor with no virtual machine getting in the way, while developers

can create applications using either the high-speed native applications or HTML5 applications. The CyanogenMod fork that Ubuntu Touch uses has been stripped of the Dalvik Virtual Machine (Android's virtual machine) along with other components that are needed to run Android applications, so Ubuntu cannot run Android apps. Canonical makes use of its Ubuntu Linux distribution for the basis of Ubuntu Touch, so applications developed for smartphones will run just fine on an Ubuntu desktop or laptop machine. Developers can create a single application for cross-device use and sell it though the company's integrated *Ubuntu Software Center*.

But in order to really start making applications, developers need to get Ubuntu SDK (just a preview for now) from the developer's website [67]. They have adopted QML (*Qt Meta Language*), as the core of the technologies to bring Ubuntu into mobile devices. QML is a powerful JavaScript-based declarative language, used to design responsive and intuitive user interfaces. As the core of internet technology seems to be more and more HTML5, Ubuntu toolkit also offers the flexibility to support HTML5. The choice of which technology to use, has to be made by the developer.

The developer must have an Ubuntu distribution as well as Qt5 and Ubuntu QML toolkit. They recommend using Qt Creator [78]. Qt is a cross-platform C++ integrated development environment and a powerful tool to develop applications based on the Qt framework. It includes an advanced code editor, UI designer, project management tools, integrated visual debugger, among other things. It uses the C++ compiler from the GNU *Compiler Collection*.

Then, after opening the editor, the user simply chooses to create a C++ application if he wants to write an app using C++ or chooses Qt Quick application if the desired language is QML [67].

3.3.3 TESTING UBUNTU TOUCH

At the moment, there is no simulator or emulator that can behave similarly to the Ubuntu phones, but any developer who wants to make applications for mobile, theoretically can run them in Ubuntu desktop and, at least, see design of the application at a specific moment.

But it is already possible to install Ubuntu Touch Preview on devices and give both developers and users the opportunity to try the OS right away.

Currently, there are four supported devices that can be flashed with the operating system, and give the user an accurate idea of how the system will work on future releases. The devices are Samsung Galaxy Nexus, Google Nexus 7, Google Nexus 10 and LG Nexus 4. The tools needed are somewhat similar to the tools used by Android and Firefox OS: *adb*, *fastboot*, among others, as well as *phablet* tool provided by Ubuntu's team. The device must be unlocked and USB access must be given.

After the installation, if everything is running properly, it is possible to test developed applications on real devices. To do that it is simply necessary to plug in the device over USB and connect it to a wireless network, in order to the SSH key pairing to happen over the air. Then, with Qt Creator, the user can connect to the device and by pressing Ctrl+F12, the application will be installed and executed on the remote device [66].

3.3.4 SENSOR FRAMEWORK

It was not possible to test and study the sensors in Ubuntu Touch. Although it was possible to build the source and install it on Galaxy Nexus device, the APIs that control sensors are not available for developers. By the way, not all the operating system's functionalities are available to developers at the moment. Ubuntu's developers are still trying to expose all functionalities through the SDK, and thing like sensor access, content management and some others are still not available, or at least not fully implemented, as it is possible to check in [29]. So, for that reason, sensor testing was not made on this new operating system.

3.4. TIZEN

Tizen is an open source, standards-based software platform supported by leading mobile operators, device manufacturers, and silicon suppliers for multiple device categories such as smartphones, tablets, netbooks, in-vehicle infotainment devices, and smart TVs. It is made up of the popular Linux kernel and Webkit runtime. Tizen offers an innovative operating system, applications, and a user experience that consumers can take from device to device.

Tizen's goal is to create an open ecosystem, compatible with Firefox OS and all web browsers, but unlike Firefox's Operating System, Tizen can run both native and web apps, as well as apps that are a little bit of the two. The aspect of this OS is shown in figure 18.

Figure 18 - Tizen appearance

Tizen provides a flexible and strong environment for app developers, based on HTML5. The Tizen SDK and API allow developers to use HTML5 and related web technologies to write applications that run across multiple device segments. The operating system features a home screen similar to iOS and Android with a grid of applications, a pull-down settings menu like Android, and a sliding lock screen as the one used by Windows Phone 8. Tizen is not limited to smartphones and tablets, it will also work with in-car entertainment systems, smart TVs and netbooks, although the emphasis so far has been on smartphones [68].

3.4.1 ARCHITECTURE

Tizen's efforts are focused on smartphone and tablet development. In the future, it is planned to include more device types. Currently, the architecture diagram looks like the one seen in figure 19.

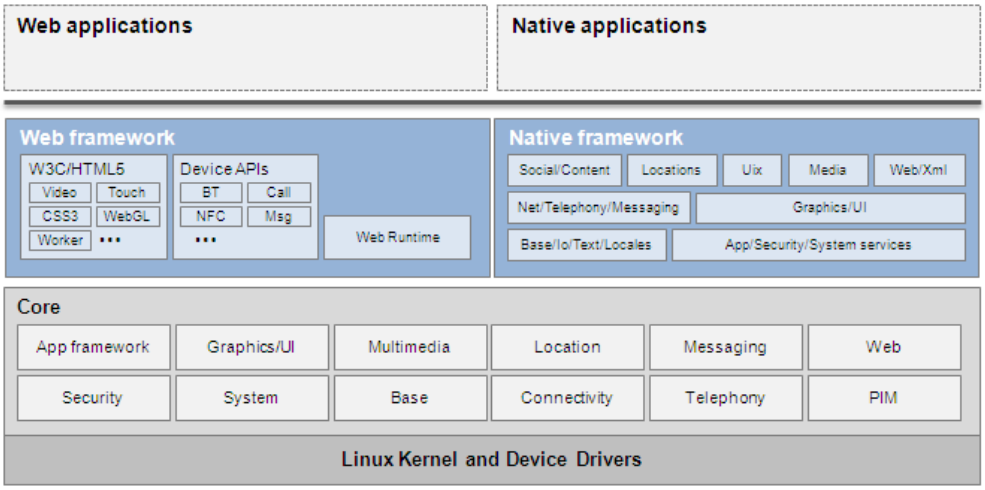

Figure 19 - Tizen architecture

It is possible to clearly identify the gap between native and the web in this operating system. Both application and framework layers are divided in these two groups. The core of the OS is the same as well as the kernel and device drivers.

The web framework provides the most up-to-date web technologies. It provides a large number of HTML5 features defined by W3C and other standardization groups, such as video, CSS3, vibration, among others. It enables access to device functionalities as Bluetooth, NFC or messaging. These device APIs come with a strict rule-based security control system that restricts malicious use, therefore it allows more security.

The native framework is a set of system services and native namespaces that could be used to access a large number of APIs with which native applications can be developed. Besides native namespaces, this framework includes popular standard open source libraries like *glibc*, *libstdc++*, *libxml2*, *OpenGL® ES*, *OpenAL*, and *OpenMP®*, to support a more efficient application development.

The core subsystem provides upper layers with features they require. It consists of a set of core services, open source libraries and an additional set of APIs that are necessary by the upper subsystems [69].

The kernel layer includes the Linux kernel and device drivers.

3.4.2 DEVELOPMENT ENVIRONMENT

This platform provides developers with two different frameworks for application development: the web framework, used to create web applications and the native

framework, used naturally to create native applications. Whether developers are using web or native, Tizen platform ensures that all the applications have a consistent design and performance.

With Tizen Web APIs, developers can build powerful web applications, using HTML5, CSS and JavaScript. Tizen supports the latest HTML5 capabilities such as animation or video. Web applications in Tizen are capable of being launched across various devices and platforms with minimal customization. It is possible to have total device access with these APIs like NFC or Bluetooth.

Through native APIs and C++ is also possible to develop powerful applications for Tizen. Applications that can access the sensors or that can perform call operations can be created with some methods in the API Reference. It is important to point that Tizen supports both core and reference applications. The core applications are developed with platform internal interfaces, such as *Enlightenment Foundation Libraries* (EFL) and other 3rd party libraries. The reference applications are developed with Tizen native APIs. Developers can switch a preloaded sample application between core and reference applications using the *MIC image creator*. This creator is used to create images for Tizen. With this tool, users can create images of different types for different platforms, including live CD images, live USB images, raw images for Kernel-based Virtual Machine, among others.

The Tizen SDK is a software development kit that includes the platform binaries, sample code and documents. It is available for Windows, Linux or Mac OSX. It includes an IDE, platform binaries and libraries and sample applications. The Tizen integrated development environment is based on the JSDT (*JavaScript Development Tools*) and Eclipse CDT (C/C++ Development Tools). It also provides an emulator that allows to emulate the target device and *Smart Development Bridge* (SDB) that works similarly to Android Debug Bridge [70].

Figure 20 shows the Tizen IDE. With this IDE it is possible to test applications on an emulator, web simulator or send directly to device over SDB. This IDE is somewhat similar to Eclipse IDE, so Android developers shouldn’t find too hard to work with it.

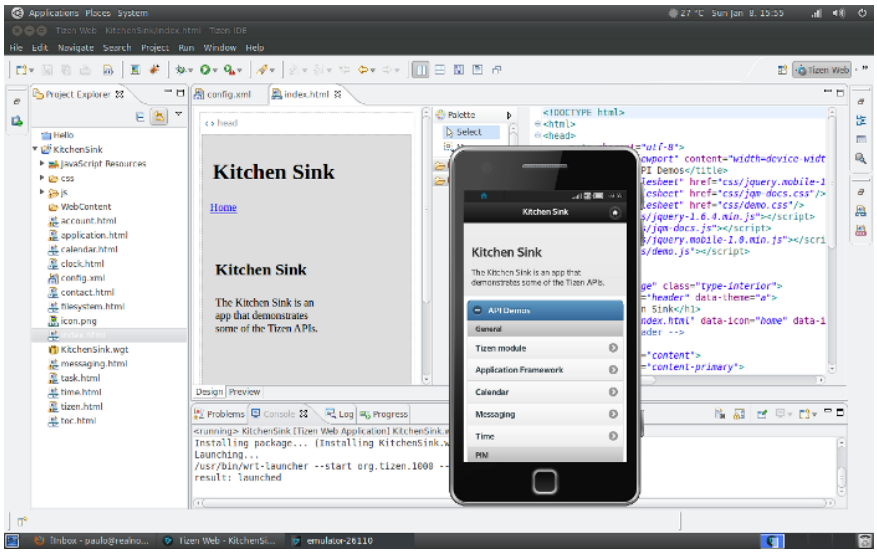

Figure 20 - Tizen IDE

3.4.3 TESTING TIZEN OS

As Tizen is an open platform, it is possible to modify the source code, repackage and publish as the developer wants. So it is possible to build or flash an image to a supported target. Currently, Tizen 2.2 (latest version) can be installed in the emulator or in one of the reference devices [68].

The emulator is an x86-based *Qemu* image that can be run on computers. Qemu is a generic and open source machine emulator and virtualizer [71]. This emulator contains a variety of preloaded applications that give the user an experience similar to actual devices and it provides the full stacks of the Tizen platform so developers can test applications before deploying them to real devices. Using the Emulator Manager developers can create and launch a Virtual Machine instance and the communication is made by SDB. There are some very useful features in the emulator like event simulation, acceleration of the guest operations using CPU or GPU, among other minor features.

There is also the possibility of building and installing Tizen in one of the reference devices. These devices are designed based on commercial target devices. Reference device-210 [73] is a device that uses Exynos 4120, C210 *System-on-Chip* (SoC) and is based on Samsung Galaxy S2 HD and reference device-PQ [72] that uses Exynos 4412 (PQ) SoC and is based on Samsung Galaxy S3. This device has, obviously, a better hardware, resulting in better performance. It is hard to get one of these devices, because they only give them in specific

conferences or to specific companies that may be connected or working with Tizen Corporation.

3.4.4 SENSOR FRAMEWORK

As Ubuntu Touch, it was not possible to test the sensors in the Tizen Operating System, but for a different reason. It was not possible to install this OS because we did not have any device compatible with it. As referred in [3.4.3] it is only possible to install Tizen on two different reference devices.

3.5. OPERATING SYSTEMS COMPARISON

Table 4 resumes the major advantages and disadvantages of the operating systems investigated.

Table 4 - Advantages and disadvantages of the Operating Systems

	Advantages	*Disadvantages*
Android	○ An "infinite" number of free applications available; ○ Phone options are very diverse; ○ Personalization – users can add widgets, themes, giving an "unique" user experience; ○ Google gives a wide range of services from Gmail to Google Reader, for example;	○ Advertising – free applications will come with ads; ○ Hard for developers to test applications because of phone variety; ○ Wasteful battery life because of lots of processing in the background; ○ Many devices mean many processor types, and applications like games may not run on a specific one;
Firefox OS	○ Primarily aimed at low-end devices – cheaper phones; ○ Easy to personalize; ○ Easy to update; ○ Uses HTML5 that allows web developers to build apps with no need for learning a new language;	○ Too much web-oriented; ○ Users in not-connected territories may find it hard to use; ○ Because of the low-end devices, the user interface may seem a little basic, for some users;

Ubuntu Touch	○ Intuitive and easy to navigate; ○ Uploaders don't need to pay to upload apps like in Android; ○ No VM, so it is faster and powerful; ○ In the future, smartphones will power the desktop because of the Linux software compatibility;	○ Requires a minimum of a dual core processor; ○ Lack of native apps, not much developers are using QML; ○ Many of the major tasks are still based on the terminal;
Tizen	○ Can run both native and web apps; ○ Not limited to smartphones and tablets; ○ Backing of Samsung and Intel; ○ Can run Android apps;	○ Needs more clarity regarding platform's future to win developers and consumers; ○ OS may need a few years to be fully stable;

4. Optimizing Sensor Access in Mobile Operating Systems

In order to fulfill the objective of optimizing access to sensors, an application was first developed to enable testing the main sensors and getting information on performance. In a second phase, some changes in the Operating Systems source codes were performed and tests were conducted to understand the achievements obtained.

4.1. Android

Firstly, a study about sensor sampling frequency in some Android devices was performed to give an overview and some results that those devices achieve out-of-the-box. Then some changes in the source code were made.

In order to test the sensors in Android, a Java application was developed using Eclipse IDE. It has two activities: one for the main screen where it is possible to choose which sensor to test and other with the values obtained from each sensor.

When the sensor is chosen in the main activity, the application creates another activity. In this new activity sensor values are gathered. In figure 21 it is possible to see the main screen (left) and the sensor values acquisition (right). On this specific case the information given is about the accelerometer, and the device is the Sony Xperia S.

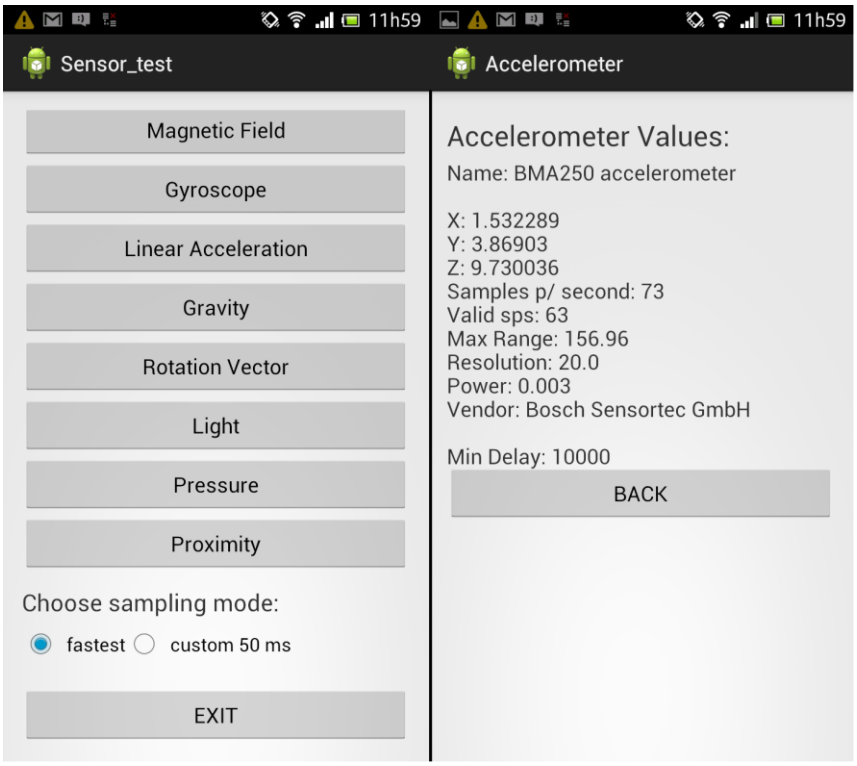

Figure 21 - Android application

It is possible to see the name and the manufacturer of the sensor, the X, Y and Z values, but most important the samples per second that give a very approximate estimate about the sensor's sampling frequency. Other hardcoded values can also be seen like the Min Delay that estimates the minimum period between sensor events in microseconds and the Power, which shows the operating current of the sensor in mA. These values sometimes are not the real consumption of the sensors and it is always better to consult the datasheets.

The code to start the magnetometer activity, for example, that will do all the operations in order to gather the information about the sensor is the following:

```
    Button mag_button = (Button) findViewById
                    (R.id.magnetometer);
    mag_button.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        Intent intent = new Intent( SensorActivity.this,
                        Magnetometer.class );
     startActivity(intent); }
}
```

This code is the same for all the sensors. First it is necessary to put a listener on the button and if the user clicks the button then it will start a new activity correspondent to the sensor chosen. It migrates from `SensorActivity` to other specific sensor activity.

In order to identify the sensors available on the device and to choose a specific sensor, a reference to the sensor service is needed and this can be done by creating an instance of the `SensorManager` class and to get the default specific sensor using the `getDefaultSensor` () method and passing the parameter specific to the sensor. This returns null if the sensor is not available.

In order to save battery it is a good practice to register the listener on the `onResume` () method and unregister it on the `onPause` () method. This way the sensor stops to acquire data and use battery when the activity pauses or when the user is done using the sensor.

With the listener registered for the wanted sensor, when it reports for a new value, the system invokes the `onSensorChanged` () method, providing a `SensorEvent` object. It is on this method that the values for each axis (X, Y and Z) for the case of magnetometer are acquired and the samples per second are determined. This is done using the `System.currentTimeMillis` () function, that returns the system time in milliseconds. Then it is just necessary to check if the time value plus 1 second is greater than the value returned by the function every time that a sensor event is generated. If not, it means that a second has passed and it is possible to count how many times the system entered the `onSensorChanged` () method for the last second. Repeating the process will give a reasonable idea of the sensor's frequency.

```
time = System.currentTimeMillis ();
…
Override
public void onSensorChanged(SensorEvent event) {
       if (time + 1000 > System.currentTimeMillis()) {
                        countSps++;
                  x = event.values [0];
                   y = event.values [1];
                     z = event.values [2];
                        } else {time = System.currentTimeMillis ();
                countSps = 1;}
```

A condition was also made in order to confirm valid values. If the last value acquired by the sensor is the same as the current value this could mean that the value is invalid, because the sensors are suitable to sense minimal changes in the environment and it is almost impossible to get the same value twice in a row (at least for magnetometer, accelerometer and gyroscope).

4.1.1 GOOGLE NEXUS 10 TABLET

This device was the one used to make the source modifications on Android. It is a powerful 10-inch tablet that has all the three sensors that we want to test. But the main reason for this choice was the fact that it is possible to build the source of an AOSP version for this device. In this case 4.2.2 Jelly Bean was the Android version used. Google Nexus 10 can be seen in figure 22.

Figure 22 - Google Nexus 10 tablet

Accelerometer and Gyroscope

Almost all the Nexus family uses the *Motion Processing Unit* (MPU) from the InvenSense manufacturer [76]. Google Nexus 10 uses one of the InvenSense's devices that

incorporates motion sensor fusing. Sensor fusing is the combining of sensor data from multiple sources in a way that the final information result is in some sense better than the information when the sources are used individually. The MPU-6050 for Nexus 10 devices, for example, combine a 3-axis gyroscope and a 3-axis accelerometer on the same silicon die together along a *Digital Motion Processor* (DMP) that is capable of processing complex 9-axis motion fusing algorithms. To integrate a 9-axis motion fusing algorithm, external sensors are accessed. Magnetometers are the most common external devices to be linked with the MPU and are accessed through an auxiliary master I^2C bus. This family is comprised in two parts with features that can be seen figure 23 [79].

Figure 23 - MPU6000/6050

More details about accelerometer and gyroscope are shown in table 5.

Table 5 - MPU6050 characteristics

	Accelerometer	**Gyroscope**
Operating Current	500 µA	3.6 mA
Max output data rate	1024 Hz	1024 Hz (slow mode)
Startup time	Not significant	30 ms
BUS interface	I^2C	I^2C

Magnetometer

The magnetic field sensor present in Nexus 10 is the AK8963 from *Asahi Kasei Microdevices* (AKM) [77]. This is a 3-axis electronic compass with high sensitive Hall sensor technology. It incorporates magnetic sensors for detecting terrestrial magnetism in the three axes. More details are shown in table 6 [80].

Table 6 - AK8963 characteristics

	AK8963
Operating Current (Typ.)	5 mA
Time for a single measurement	7.2 ms
Interface	I^2C or SPI
Measurement range	± 4900 µT

4.1.2 RESULTS

Some sensor tests were made among the available Android devices, allowing an overview about the maximum sampling frequencies in smartphones nowadays. Figures 24, 25 and 26 show those results:

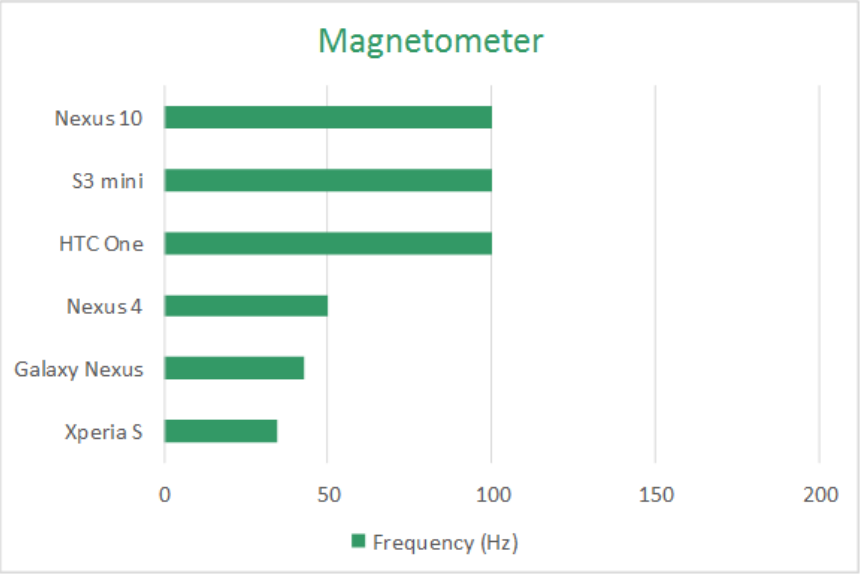

Figure 24 – Android Magnetometer results

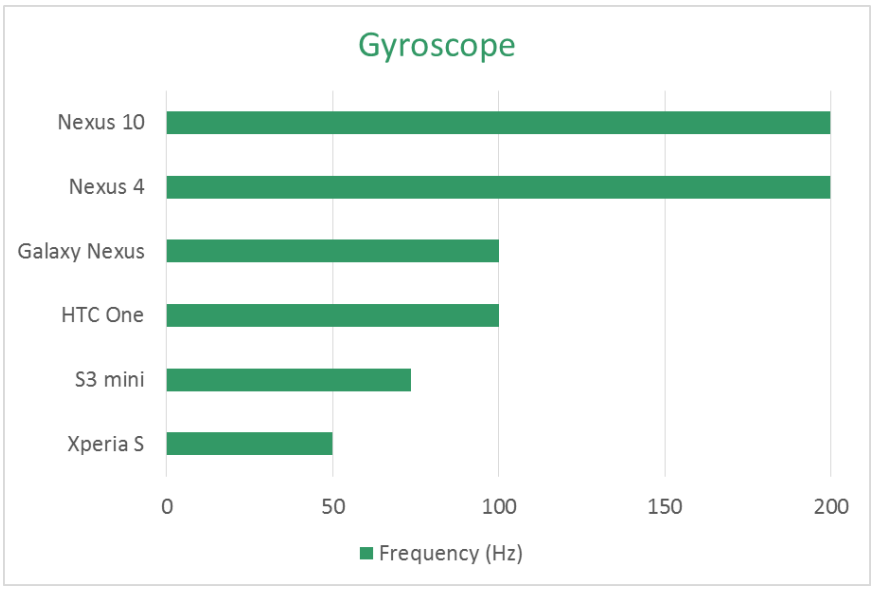

Figure 25 – Android Gyroscope results

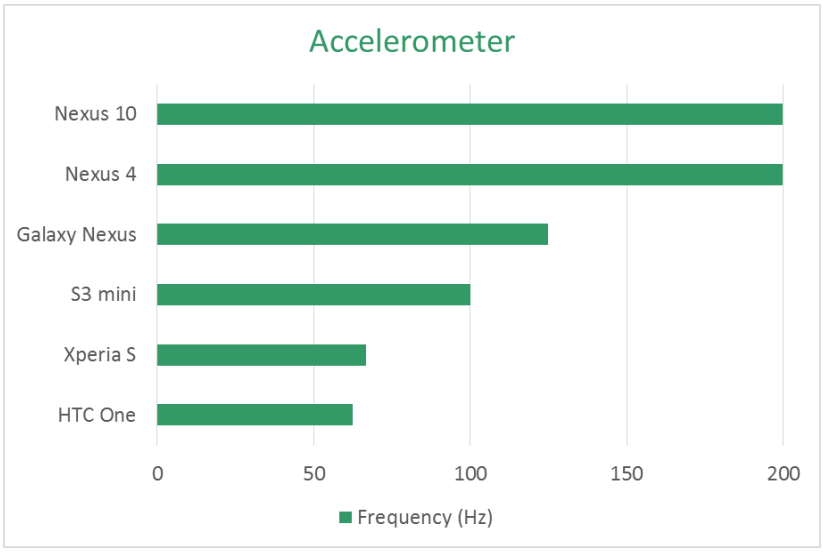

Figure 26 – Android Accelerometer results

Making a quick analysis of the sampling frequencies obtained, it is possible to see that Nexus 10 is the one with better results out-of-the-box. So, it seemed obvious that we should take this device and try to improve its frequency results. This device is one of the devices supported by AOSP, so we can easily access and modify the Android source code relative to this device.

A 4.2 *Jelly Bean* version of Android was built and installed from the AOSP. The tests made were mainly focused on the sampling rate that each sensor could achieve. Android has a fastest mode of collecting data that seemed to be limited somehow, so if we want to

overcome that limit some changes had to be made in the source code and later a re-compilation of the Android system.

Some sensors were tested before, and after the re-compilation and the results regarding the sampling rate were noticeable and they can be seen in table 7. The fastest mode available was used:

Table 7 - Android Sensor Results

Device	Sensors	Before	After
NEXUS 10	*Gyroscope*	200 Hz	1000 Hz
	Accelerometer	200 Hz	1000 Hz
	Magnetometer	100 Hz	127 Hz

By default, gyroscope and accelerometer have a sampling rate of 200 samples per second on Nexus 10, while the magnetometer's is 100 samples per second. It was possible to increase the sampling rate to 1000 samples/second on the gyroscope and the accelerometer, and 127 samples/second was the maximum value achieved on the magnetometer's case.

Theoretically, these values were approximately the expected ones. In [4.1.1] was made an analysis of the sensors' datasheets where it was possible to verify that the maximum output data rate for the MPU 6050 in the accelerometer and the gyroscope was 1024 Hz, while the minimum measurement time in the magnetometer is 7.2 milliseconds, which is approximately 139 Hz. Those values can't be reached as the Android system has some delays due to the system's processing. They are the absolute maximum sampling frequency that the sensors can achieve. Maybe the gyroscope can go even further, because the slow mode was the one defined in the source and no modifications were made in this mode. But this was not investigated.

To achieve those values, changes were made in the Motion Processing Library, located on the *Sensor HAL*. This library configures and defines the parameters for the MPU usage. Features like delay, which sensor to use and other features are defined here. The file is inside `libsensors_iio` folder and it is called `MPLSensor.cpp`.

By default, a limitation is defined here, that limits all the sensors rate by 5 milliseconds. If at any point, the developer wants for the samples to be delivered faster, he would be blocked by this limitation and 5 milliseconds was the fastest rate possible. So, the obvious

thing to do, was to remove that limitation. That way, it was possible to increase by five times the amount of samples per second in the gyroscope and accelerometer.

In Nexus 10, as the MPU combines an accelerometer and a gyroscope, it was expected that the rate of both of them could be the same. This was what happened. They are on the same silicon die together and the data is processed by the DMP for both sensors. So the rate is the same before and after the changes in the source.

The magnetometer data is also processed by the DMP, but the device is external, it is not embedded in the same silicon die and the limitation was 10 milliseconds. So the sampling rate after the modifications could only reach approximately 8 milliseconds, instead of the 1 millisecond like the other sensors. The magnetometer has it minimum measurement time defined in 7.2 milliseconds, and this value is impossible to be reached because of delays imposed by the Android operating system [80].

```
if (ns < 5000000LL) {
        //ns = 5000000LL;
        ns = ns;}
    /* mDelays array for each sensor */
mDelays[what] = ns;
```

In the function `setDelay()` making the modification seen above allows to define the delay directly when registering the sensor. There are other alike limitations around the source code and also had to be removed.

The modifications consist in either hard code a delay on the `MPLSensor.cpp` (such as `ns=1000000LL`, that is equal to 1000 microseconds) or registering the sensor with the delay desired in microseconds (such as 1000). With the limitation gone, this delay determines the minimum period between sensor events. Gyroscope's delay controls it all. If both gyroscope and accelerometer are registered, if the accelerometer's delay is superior to the gyroscope's delay, the delay will be set as being the gyroscope's delay. The other way around does not occur. It is possible to say that in terms of delay the importance is `gyroscope>> accelerometer>>magnetometer`. If any delay for the most significant sensors is lower, then the delay of the least significant sensors is changed. The figure 27 represents the layers that had to be accessed in order to change sensor sampling frequency values.

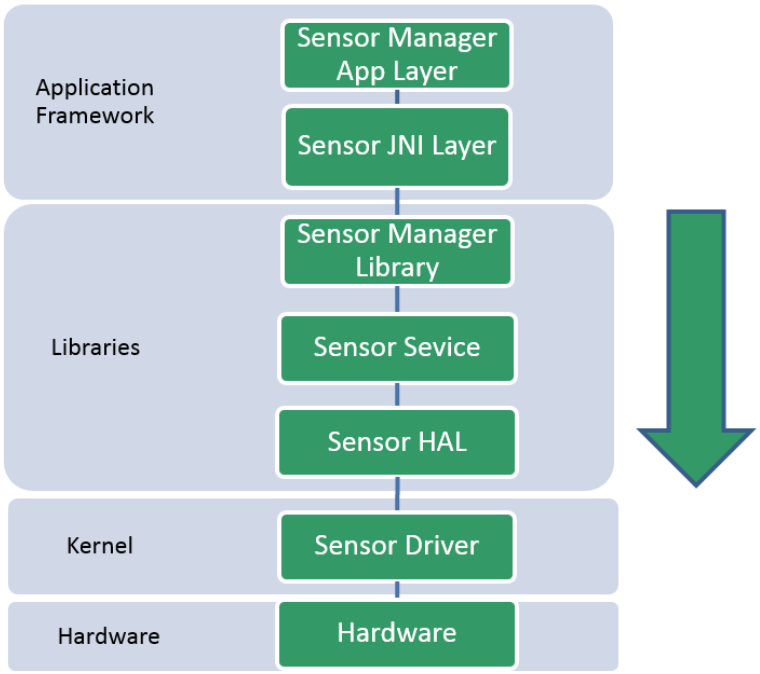

Figure 27 - Android sensor layers

After sensor HAL and Motion processing Library, Android will enter the kernel driver for each sensor, performing all the fundamental tasks in order to define delays and retrieve values from the actual sensor (hardware). This is optimized in all cases, retrieving the values from the hardware as soon as the value of each axis changes, but limited in the layers above.

This study proves that sampling frequency in Android can be modified and it is really limited by software.

4.2. FIREFOX OS

The same tests as [4.1] were made for Firefox OS, with the difference that it was not possible to test the operating system on an actual Firefox OS device. Instead, these tests were made on a Nexus S that was running Android before the installation of the Firefox OS version 1.0. The choice of this device resides in the fact that it possesses the three sensors that were intended to be studied and it is supported and compatible with this operating system. It is possible to build and install a Firefox OS version on this device.

4.2.1 SAMSUNG NEXUS S SMARTPHONE

This device was the chosen one to perform all the tasks for Firefox OS defined in this dissertation. The choice was based on the availability of the Firefox OS source code for

this device. It is possible to build the source code for other devices, but this device was available on the company and the three sensors are present in this smartphone, and this can help to explain the choice, as well.

Figure 28 - Samsung Nexus S

Accelerometer

The accelerometer in Nexus S is the KR3DM from STMicroelectronics. The low power consumption is the feature that stands out in this 3-axis accelerometer. Table 8 shows some specifications of this sensor.

Table 8 - KR3DM characteristics

	KR3DM
Operating current	225 µA
Interface	I^2C or SPI
Output data rate	3.125 Hz to 1.6 kHz
Max range	± 16.0 g

Gyroscope

The low power 3-axis angular rate sensor used in Nexus S is the K3G from STMicroelectronics. It has an embedded temperature sensor that can be used to measure the sensor's temperature and a good high shock resistance. More details in table 9.

Table 9 - K3G Gyroscope characteristics

	K3G
Operating current	6.1 mA
Interface	I^2C or SPI
Output data rate	95 Hz to 760 Hz
Max range	± 2000 degrees per second

Magnetometer

AK8973 from AKM is the magnetic sensor present in Nexus S. This 3-axis sensor has high sensitive Hall sensors integrated and was developed with the intention of being used in mobile phones and handy terminals. Its characteristics can be seen in table 10.

Table 10 - AK8973 Magnetometer characteristics

	AK8973
Operating Current (Typ.)	6.8 mA
Time for a single measurement	12.6 ms
Interface	I^2C

4.2.2 RESULTS

With the device motion and device orientation events was possible to get sensor data, as well as the sampling rate associated. The application developed aimed to get the values from the device motion event and the device orientation, as well as the value of the light sensor and proximity sensor. It was possible to verify that in this device, the acceleration without the effect of gravity is an attribute that didn't exist or couldn't be reached. In this operating system, it is not possible to define a delay, like in Android, so the frequency that the values are gathered cannot be defined. This frequency was, in this case, 10 Hz for device motion and device orientation events. The light and proximity events were only fired when some significant changes were made to the ambient light around the device and when an object became close to the proximity sensor, so on this cases, the frequency couldn't be measured. This 10 Hz frequency is proven by the interval attribute given by the device motion event. This had the value of 100 milliseconds. In the application it is also possible to verify how much time passed between the current and the last event fired for

the specific event defined. Those values were approximate 100 milliseconds on the motion and orientation events, confirming the 10 Hz frequency on this device.

The application was made using HTML5 + CSS + JavaScript like all of the Firefox OS applications. The figure 29 demonstrates how the application looks. The Firefox OS Simulator for the Firefox browser was the tool used to deploy the application into the real device. No special IDE is needed to develop a Firefox OS application. Any IDE can be used to develop application code, or even simpler, developers can use a text editor like Notepad for developing HTML5 + CSS + JavaScript applications.

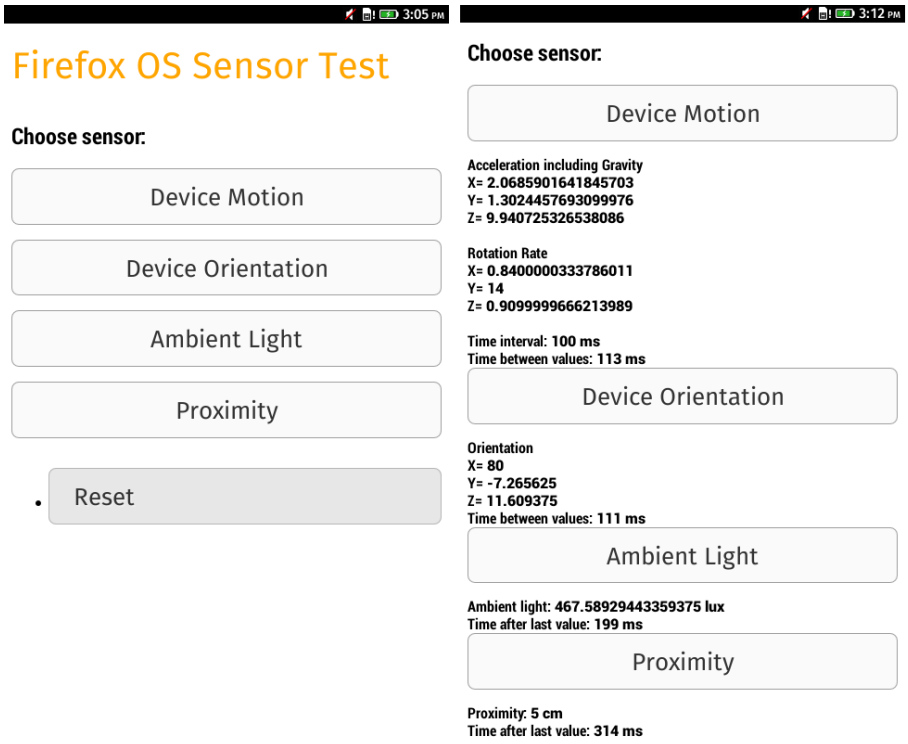

Figure 29 - Firefox OS application

This application is only composed by one screen, that will show the wanted parameters when buttons are clicked. It is used a JavaScript file named `sensortest.js` that contains the listeners for each button and the code that will run once the button is clicked. The code for light sensor, for example, is the following:

```
var ambientLight = document.querySelector("#light"),
 ambientLightDisplay = document.querySelector("#light-display");
    if (ambientLight && ambientLightDisplay) {
        ambientLight.onclick = function () {
            ambientLightDisplay.style.display = "block";
            window.ondevicelight = function (event) {
                now = Date.now();
                diff = now-last;
                last = now;
                var lux = "<strong>Ambient light: </strong>" + event.value + "
                lux<br>";
                lux += "<strong>Time after last value: </strong>" + diff + " ms
                <br>";
                ambientLightDisplay.innerHTML = lux; }; };}
```

In this code, `document.querySelector` returns the first element that matches the "light" selector, which in this case is a button. Then listens for a "click" with function `onClick` (), and when the button is clicked the applications listens for an event, that in this case will be every `ondevicelight` events. The lux value is acquired and sent back to the HTML through a variable called "`lux`". Then it is necessary to know the sampling rate of the sensor values and this is done using `Date.now()` method that returns the milliseconds elapsed since 1 January 1970 00:00:00 UTC up until now as a number. Knowing that allows to calculate the time between sensor events giving an approximate estimate of the sampling rate. The access to other sensor events is similar to this one.

After a quick look at the figure 29, it is possible to find the time between events or after the last event attributes. These values theoretically should be approximate to 100 on the cases of device orientation and device motion events, because of the 100 milliseconds constant given by the interval attribute. But it almost never reached that value, which shows that the interval constant is just an informative attribute and gives the minimum interval that the sensor data can be gathered. The light and proximity sensors only give new values when the data is changed, so the time parameter found on the application is very variable for those sensors. In the figure 29 it is possible to see that 199 milliseconds is the time interval since the last lux value change. But if there is no changes in the light for a long time, this value continues on growing, until some noticeable light change occurs. The same occurs for proximity events. It was not possible to obtain the value of acceleration without the effect of gravity or linear acceleration. Comparing with Android, for example, the application seems a little easier to develop, although it is not possible to define parameters like delay and it is not possible to access some of the sensors like the magnetometer. WebAPIs provide limited access to the sensors and to their values.

It is possible to increase the sampling rate of the sensors like in Android. Most of the action inside Gecko is triggered by input events. Input events enter Gecko through the Gonk Application shell. Sensor events are no exception. In the Gonk App Shell is a constant defined with the value 100 named `sDefaultSensorhint`. This is the interval in milliseconds that the sensors will be requested for sensor data. This constant is related to another defined in `nsDeviceSensors.cpp` named `DEFAULT_SENSOR_POLL`. If we change both constants to a lower value, it will change the sensors sampling rate as well.

```
sm.registerListener(GeckoApp.mAppContext, gOrientationSensor,
                    sDefaultSensorHint);
```

When the library registers the listener for the sensor it passes the sensor poll value and this value was the one changed.

Those libraries allow to interpret the JavaScript defined in the application level and allow the code to interact with Android libraries. As Firefox OS is "sitting" on top of Android, it is possible to maximize the sampling rate of sensors to values similar to the ones provided by Android devices. In the case of Nexus S it was possible to increase the frequency from 10 Hz to 100 Hz maximum in motion events and 50 Hz in orientation events. Because Nexus S is limited to 100 Hz by default in the accelerometer, gyroscope and 50 Hz for the magnetic field sensor, this was the maximum value reached, but if the same study and changes in the source were made for Nexus S like the ones in Nexus 10, the frequency could even be higher than 100 Hz as occurred in Android. The figure 30 describes the data flow of Firefox OS sensor events in Nexus S.

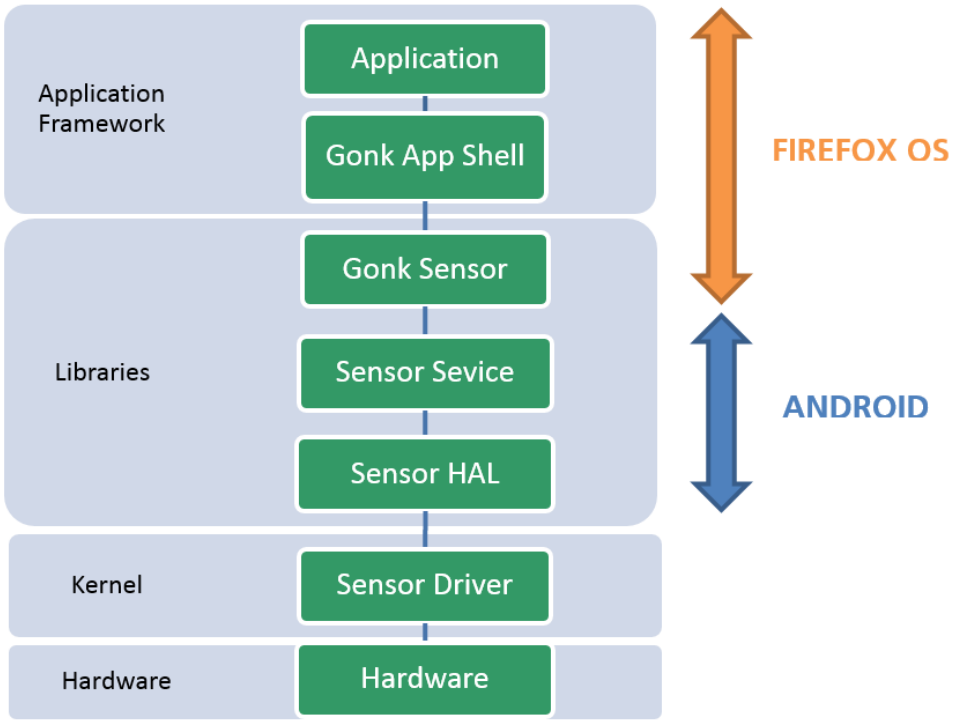

Figure 30 - Firefox OS sensor layers

An application registers handlers for sensor events and this way the developer is notified and can use the values obtained from physical sensors in Firefox OS anyway he wants. After registering the handlers, the Gonk application shell register the sensors that will be enabled. Gonk sensor is responsible to determine the minimum time interval between sensor events and to retrieve the values from the sensors. With the help of Android libraries, the Sensor Manager Library in Firefox can control and fire DOM Events with specified poll intervals, allowing to change the sampling frequency of a sensor application. The process after that is similar to Android, with the sensor drivers and sensor HAL defining all delays and conversions for sensor values.

5. ADAPTING AN EXTERNAL DEVICE TO AAL ENVIRONMENTS

CC2541 SensorTag from Texas instruments [74] was the device used to achieve the other goal of this master thesis - adapting an external device to AAL environments. The idea was to develop firmware for this device that allows a smartphone to interact with it and be able to receive sensor values via Bluetooth Low Energy.

This single mode device is low cost, low power that can be very useful in BLE applications. It combines a Radio Frequency (RF) transceiver with an industry enhanced 8051 MCU. It is programmable and combined with the BLE protocol stack developed by Texas Instruments, may well be one of the most flexible and cost-effective solutions on the market nowadays. In figure 31 it is possible to take a look at SensorTag's aspect.

Figure 31 - CC2541 SensorTag

It is easy to put this device in the pocket for example, due to its dimensions: 7x4 cm approximately. That way, it is possible to make activity monitoring on a patient, fall detection and, if the patient can send a call for help, for example. He just needs to take the device out of the pocket and press a button. This will notify a smartphone, or other device that uses BLE that the user needs help.

5.1. SENSORTAG COMPONENTS

Besides all the components that are necessary to make a device like this BLE connectable and running, the SensorTag has some sensors and buttons that increase the interactivity and the utility of this device. The sensors that are of more interest are the ones that have been studied in this master thesis: accelerometer, gyroscope and magnetometer. But beyond that, this device has a temperature sensor, a humidity sensor and a pressure sensor.

5.1.1 ACCELEROMETER

The accelerometer in SensorTag is the KXTJ9 from Kionix. It is a 3-axis silicon micromachined accelerometer. This product specifications can be consulted in table 11.

Table 11 - KXTJ9 accelerometer characteristics

	KXTJ9
Operating current	135 µA
Interface	I^2C or SPI
Output data rate	0.781 Hz to 1.6 kHz
Max range	± 8.0 g
Max Data rate in SensorTag	10 Hz

5.1.2 GYROSCOPE

InvenSense MPU 3000 is the gyroscope that is incorporated in SensorTag. This device has a 3-axis gyroscope and a DMP on the same silicon die, along with a secondary I^2C master port that can function as an interface for third party accelerometers providing 6-axis motion fusing with processed acceleration and rotation motion data. More specifications can be seen in table 12.

Table 12 - IMU3000 Gyroscope characteristics

	IMU 3000
Operating current	6.1 mA
Interface	I^2C or SPI
Max output data rate	1000 Hz (slow mode)
Max range	± 2000 degrees per second
Data rate in original firmware	1 Hz

5.1.3 MAGNETOMETER

Freescale's MAG3110 is the 3-axis magnetometer that is used in SensorTag. This component output data rate is slower than the other magnetometers studied in this master thesis, but still, the 80 Hz output data rate is more than enough for the majority of the applications that need to know the values of the magnetic fields surrounding. Table 13 shows some characteristics of this sensor.

Table 13 - SensorTag magnetometer characteristics

	MAG3110
Operating current	900 μA
Interface	I^2C
Max Output Data Rate	80 Hz
Max data rate in original firmware	10 Hz

5.1.4 OTHER COMPONENTS

Powering SensorTag is made using a 3 V coin cell battery. This cheap battery is expected to run for many days or maybe years on a device like SensorTag. There are other

components for user interaction: three buttons and two LEDs give users some visual information and allow users to perform some actions by pressing the buttons.

5.2. PROFILES AND SERVICES OVERVIEW

As mentioned in [2.3.4] in order for a device to interpret data that comes from SensorTag, profiles and services have to be implemented in the Tag's firmware. The original firmware comes with some profiles and services that were not studied because they were not the scope of this master thesis. But other profiles and services had to be implemented or modified to fulfill the objectives proposed.

Accelerometer, Gyroscope and Magnetometer

It was not necessary to develop from scratch profiles for receiving values from these sensors and send them to a client, because they are implemented in the original version of the firmware, but in order to make the device capable of activity monitoring and fall detection, it was necessary to change sampling rate. The accelerometer and the magnetometer had the option of changing the sampling rate by writing it on the service's characteristic for period, but it was limited to 10 Hz of maximum frequency, but the gyroscope service does not have the option of modify the sampling rate. This had to be introduced.

Battery

A battery service was created in order to verify the battery's current voltage. The intent of this service was to send notifications for a client informing that the battery reached some critical value and needs to be replaced.

Proximity Profile

This profile was adopted by Bluetooth SIG and enables proximity monitoring between client and server and is composed of three services: Link Loss, Immediate Alert and Tx Power. In this profile, Link Loss is a mandatory service, while Immediate Alert and Tx Power are optional.

This profile defines two roles: Proximity Monitor that shall be a client and Proximity Reporter that shall be a server, in this case the SensorTag.

Link Loss service was created to perform an alert when a BLE connection is lost. This way it is possible to determine when a patient is being monitored or not, or if maybe he is out of the BLE's range for the application.

Immediate Alert Service as the name suggests indicates an immediate alert when a client sends a predefined command to the server (SensorTag). In this case when a client writes a "1" on the alert level characteristic an alert is performed and it is shut down when receives a "0".

Tx Power service allows for a client to read the transmission power of a connection by reading the Tx Power level characteristic. This level will never change during a connection thus the client should only read this value once. This value is important on path loss conditions.

The Proximity Monitor shall maintain a connection with a Reporter and monitor the *Received Signal Strength Indication* (RSSI). The Monitor shall calculate the path loss by subtracting the RSSI from the transmission power level of the Reporter, discovered by reading the Tx Power level characteristic. If the path loss exceeds a predefined threshold it shall write on the immediate alert characteristic, causing the Reporter to alert. Monitor can also alert if necessary. On the other hand, if path loss falls below the threshold defined, Monitor shall write again on immediate alert's characteristic, causing the alert to stop [75].

Find Me Profile

This SIG approved profile was implemented in SensorTag in order to cause an alert when a client wants to know where the SensorTag is located. It uses the Immediate Alert Service without any extra requirements than those defined in its specification.

When the client wants to locate the target device (SensorTag) it shall write in the Alert Level characteristic causing an alert. Client can cancel the alert when the device is found and the target device may be able to cancel the alert too by pressing a button, for example.

Call for Help Profile

This custom profile is not a part of the specification but it is very important for this work. It uses a custom service, Call for Help service and its main goal is to make an alert on a client when the user presses a button on SensorTag.

In a connection when the button is pressed, a notification is sent to the client (intended to be a smartphone) that can dismiss the alert by writing to the alert level characteristic on this service. What client will do with that information it up to the application to define.

When the device is not connected it will queue the alert request, sending it right after a link was established. This way, a user can ask for help request anytime he wants. Different visual responses are supposed to be seen when a help request is sent and when a help request is waiting to be sent.

5.3. IAR

All the software developed was made using IAR Embedded Workbench [30]. The code was all developed in C language. This tool incorporates a compiler, an assembler, a linker and a debugger, all in one *Integrated Development Environment* (IDE).

Most renowned corporations in the world that manufacture products with embedded systems use IAR Embedded Workbench in critical applications, including medical and health-related applications. In addition to this, Texas Instruments provide a series of sample applications created in this IDE, to help developers to adapt the code for their devices, making this a useful application.

Apart from this IDE, it is necessary to deploy the firmware into the real device. This is done using CC Debugger, from Texas Instruments (TI). This tool is primarily used for flash programming and debugging software running on CCxxxx 8051-based *System-on-Chip* (SOC) devices from TI, as it is the case of the CC2541 investigated here. The aspect of CC Debugger can be seen in figure 32.

Figure 32 - Texas Instruments CC debugger

5.4. IMPLEMENTATION

A The purpose of modifying the SensorTag's firmware was to create a device capable of sending not only sensor values, but also alerts to a client, via BLE. It needs to receive alerts as well in order to perform tasks related to immediate alerts.

So, services have to be created or modified, making this device capable of performing those tasks. Each service has a distinct *Universally Unique Identifier* (UUID) that allows for a client to know exactly to which characteristic should write or read. Profiles can share the same service by creating two instances of the same service, as well.

Each service has also a table of attributes, that define attribute levels, characteristics, making these values readable-only, writable-only, both or none, allowing a client to perform the available tasks.

In `Sensortag_Init()` function are defined some of the initial connection parameters and if the device becomes discoverable right after the power on or not, as well as the advertising interval. But it also defines which services are being registered in the GATT server with the function `AddService()`. The order that the services are being registered is very important because it will define the instances of the services and the handles. Handles are addresses to individual attributes and the value of a handle is defined by how many attributes are registered in GATT.

Then the drivers for each component, accelerometer, LEDs, gyroscope, etc. are initialized with the init functions, as well as the callback functions for each service, that will monitor the attribute values of each service, allowing to perform the required functionalities depending on those attribute value changes.

For the scope of this study, the LEDs and a button of the SensorTag were used and the device aspect can be seen in figure 33.

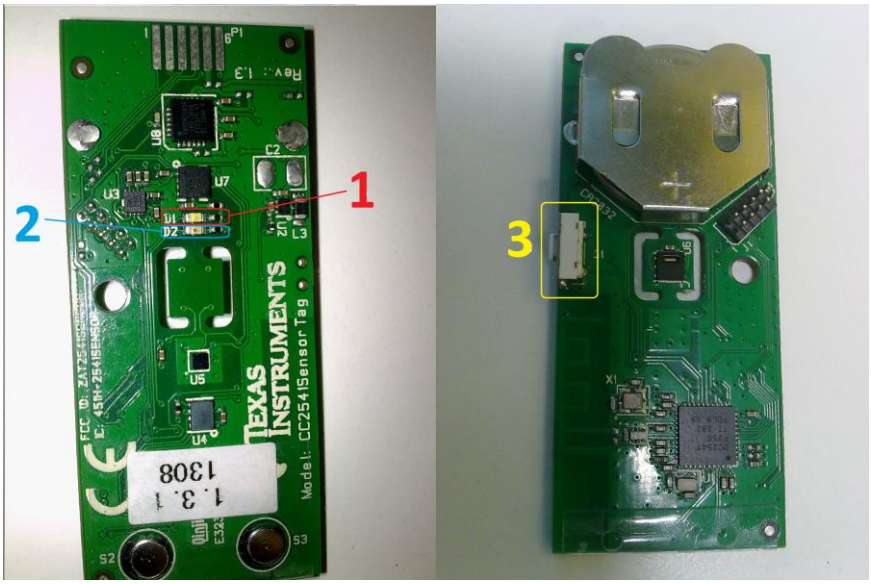

Figure 33 - SensorTag LEDs and button

It is possible to see both sides of the board and the three components that are fundamental parts of the user's interaction in this application.

LEDs 1 and 2 are represented by numbers 1 and 2 respectively and the button is represented by the number 3. There are other two buttons in SensorTag but the side button was the chosen one because as this device is primarily intended to be used on a pocket, a side button is less likely to be pressed accidentally.

5.4.1 SENSORS

The modifications in the sensors' services were made having in mind the sampling rates needed by some of the ongoing projects in Fraunhofer AICOS. Mover application makes activity monitoring and fall detection and needs a frequency of 50 Hz for the accelerometer and 35 Hz for the gyroscope. For indoor location using the magnetometer the sampling rate needed is approximately 60 Hz.

Accessing the sensor data in SensorTag is made using events that are triggered periodically. The client writes on the attribute value defined for powering on or off the sensor. When the sensor is enabled, the system schedules an event with the default period that was defined and then sends a notification and the sensor data to the client. Once the data rates needed were not superior to the maximum output data rates of the sensors (can be checked in [5.1]) it was possible to achieve those data rates simply by changing the

default periods on the three sensors. The code for reading and schedule a new accelerometer event, for example, is the following:

```
if(accEnabled)

{readAccData();
osal_start_timerEx(sensorTag_TaskID,ST_ACCELEROMETER_SENSOR_EVT,
sensorAccPeriod );}
```

If accelerometer is enabled the device will read the sensor data. After reading the data, this device will send it to the client if this data is not equal to the last data gathered and then schedule a new read event with the help of a timer defined by the OSAL. OSAL stands for *Operating System Abstraction Layer*, but it is not really an operating system, it is a control loop that allows software to setup the execution of events. All sensor events in the application are called using this method.

Other thing that had to be done was the removal of the rate frequency limitation in the magnetometer and accelerometer. These sensors were suitable of sampling frequency change if a client wrote on the period characteristic, but the minimum value for executing a new event will always be 100 that would correspond to 100 milliseconds. In the sensor callback, a multiplier is there to make sure that 100 is the minimum value. This multiplier was removed, and a condition was added only to check if the rate wanted is higher than the maximum rate given by the hardware.

Gyroscope did not have the option of defining the sampling rate, so a new attribute was created in the attribute's table, making this sensor also suitable to this feature.

5.4.2 BATTERY

This service will allow to read the current voltage of the coin cell battery and used the inner ADC of the CC2541 chip. The operating supply voltage of this chip is 2 - 3.6 V [31]. A new coin-cell battery has 3 V of voltage. So, a mapping of the voltage has to be made in order to get a percentage of the battery, where 2 V is 0% and 3 V is 100% of battery level. This is done with the help of the inner ADC.

The ADC has an internal voltage reference of 1.24 V and a 10-bit resolution was used so the maximum value is $2^{10} = 512$, which is 511 in reality (0 – 511). The voltage is measured in the AVDD5 pin of the ADC which gives the VDD divided by 3. So:

$$ADC = 511 \times \frac{(VDD \div 3)}{1.24}$$

This allows to map the voltage value between 273 and 409. Apart from this, in order to make a battery read is it needed to bypass the inner DC-DC converter present in the chip. The TPS62730 [32] is a high frequency synchronous step down converter optimized for ultra-low power wireless applications and reduces the current consumption drawn from the battery. This component is active by default and it gives 2.1 V of output voltage. So, before a battery read this component is put in bypass mode, allowing to measure directly the coin cell battery voltage level. The converter is then put back in active mode right after the measurement.

A notification including the battery level is sent to the client if it dropped down a level and a connection is established.

5.4.3 PROXIMITY PROFILE

As mentioned in [5.2] this profile is composed of three services. Tx Power, Link Loss and Immediate Alert.

Tx power level is an attribute that can only be read by a Reporter and it is the Monitor that defines it. In this case, this is made in `SensorTag_Init()` function and 0 dBm is the value defined, which is the maximum output value for this chip. This value should be used for calculate the path loss by the Reporter, and with this information send an immediate alert that can use the Immediate Alert Service to do that. The Reporter can define two types of alert by writing a `LOW_ALERT` or a `HIGH_ALERT` on the alert level characteristic. In the case of a `LOW_ALERT`, the LED 2 is turned on using the function `HalLedSet()`, but if a `HIGH_ALERT` was the one written, the LED 2 shall blink using the `HalLedBlink()` function. These are defined in a LED driver called `HalLed`. These alerts shall stop when a `NO_ALERT` is written by the Reporter.

The link loss alert will occur when the devices that were in a connection get too far apart of each other. For that is defined a supervision timeout value that defined the maximum period that the device can be in a connection without receiving any packet from the client. This period is defined in 10 seconds. After this period, if the Reporter wrote a low or a high alert on the link loss alert level characteristic when they were connected, the device will behave in the same way as in the Immediate Alert service.

5.4.4 FIND ME PROFILE

This profile uses the Immediate Alert service in order to perform an alert when the client wants to know where the SensorTag is. When the client writes an alert level in the characteristic, the LED 1 is turned on, behaving exactly like in [5.4.3], depending if the alert is a low or a high alert. The alert can be cancelled by the user by simply pressing the button in SensorTag, or by writing a no alert on the characteristic. This should be done by the client. Obviously, this profile does not have any practical utility, because with just only a LED it may not possible to determine the location of the device. But this can prove this concept and with a component that can reproduce a sound, like a buzzer, this profile might be very useful.

5.4.5 HELP REQUEST

The user can send a help request using the call for help service. There are endless scenarios where this service can be very useful. The user might be feeling sick and in need of assistance, a burglar might be trying to get in the house, among many other scenarios. By just pressing the button, a help request is sent to the client (smartphone, pc), and this device can then send this information to a caregiver or a doctor for example. When devices are connected, a notification is sent, but if there is no connection available, the device shall queue the request and send it right after a connection is established. The visual information is different whether the request was sent or is expecting to be sent. If the help request was sent, the LED 1 stays on indefinitely until the user cancels the help request by pressing for 3 seconds the button or until the client writes a zero on the characteristic help request level, giving the user the information that something has been done with that help request. If there is no connection available, the LED 1 blinks and sends the notification right after the connection is established again, stopping the blinking of the LED 1 and the visual information is the same as the described above, when the connection is established.

5.4.6 TESTING

There are some Windows applications like *Btool* [33] or *BLE Device Monitor* [34] that were used to test the developed firmware, but because the device was firstly intended to be used connected with a smartphone, most of the testing was made with an application developed by Fraunhofer AICOS.

BLE is only supported in the 4.3 version of Android, and few devices can actually connect to peripherals via BLE. HTC One is one of them and this was the device used.

Initialization of the Device

When a coin cell battery is inserted into the device the LED 1 stays on for 3 seconds, and a battery measure is made. If the battery level is below 10%, LED 2 is turned on for another 3 seconds and the device is put into sleep, nothing more can be done to the device except change the battery. This is made because it does not have the required voltage value.

The device does not ever come back to this state unless the user presses the button for 7 seconds, at any point of the execution. This will make a software reset and the device is initialized again.

Advertising

If the device "passes" the initialization, the user can then press the button making the device advertisable and ready for a connection. The device advertises indefinitely, and unless a software reset is made or a connection is established, the device will never stop advertising while the battery is still providing power.

Connection

The device can now connect to the smartphone and start sending sensor values and other notifications. The application main screen can be seen in figure 34. On this state, devices can share help requests, find me requests, and path loss and link loss alerts.

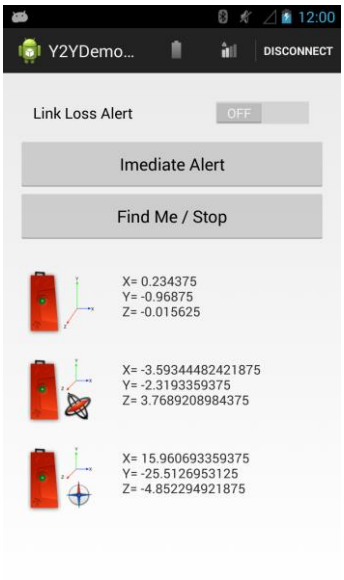

Figure 34 - SensorTag Android main screen

It is possible to analyze sensor data and to test the other alert options with the help of screen buttons.

Help Request

When a help request is made, the device notifies the smartphone of that situation and a dialog is opened and can be seen in figure 35. The smartphone then waits 20 seconds before sending a SMS to a caregiver in order to give time for a cancel by the user, if he wants to. After this period, the smartphone dismisses the help alert on both devices.

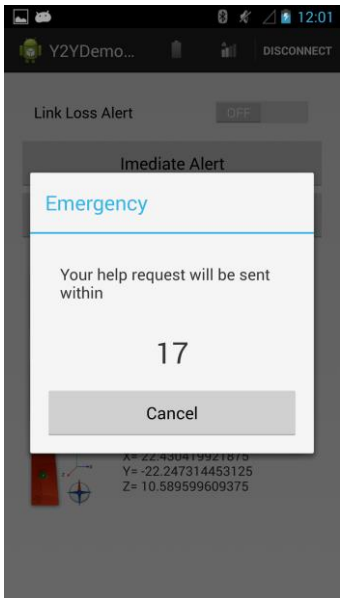

Figure 35 - Help request dialog

This alert is the priority and if an alert of this kind was made, no other alert can be sent while this alert is still active.

Path Loss

When the RSSI level is below a threshold defined by the smartphone, an immediate alert is sent to the SensorTag and a notification is shown in the smartphone telling that they are becoming too far apart of each other and soon the connection may be lost. If this value is above the threshold, it means that they are "in range" and this alert is cancelled. Information of this fact is given in both sides. Figure 36 shows what this information looks like in the Android application.

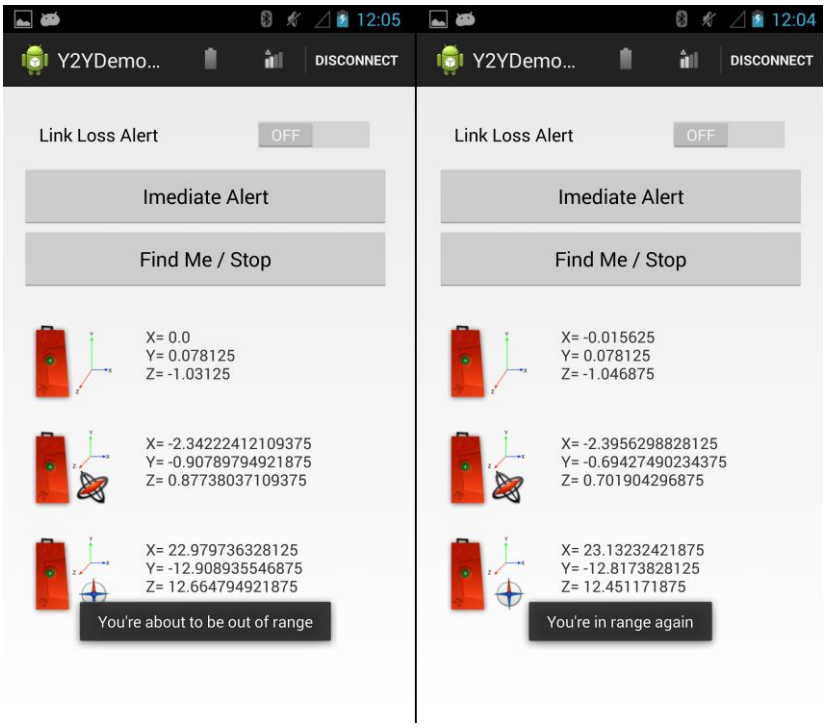

Figure 36 - Path loss events

Link Loss

If the connection is lost, there is a 120 second period for devices to be connected again. Since the SensorTag advertises automatically after a link loss, the smartphone will automatically try a re-connection, and if this not happens, it will send a message to the caregiver stating that the patient is no longer being monitored. Even in link loss, if the patient wants to make a help request, he can press the button. The help request will not be

delivered to the smartphone, but this information will be queued and will be sent after the re-connection. Link Loss event can be seen in figure 37.

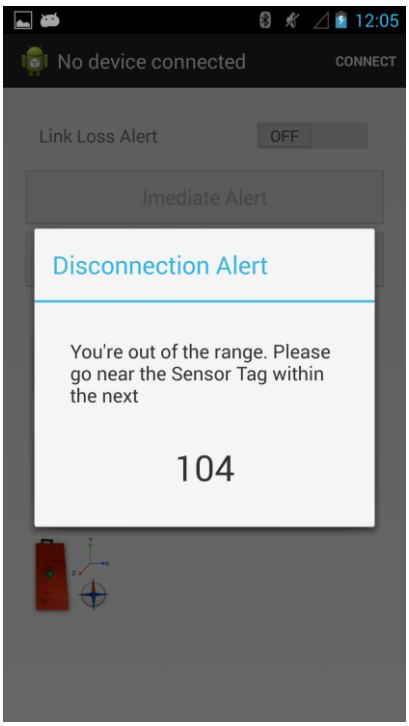

Figure 37 - Link loss disconnection alert

Forced Disconnection

If the smartphone forces a disconnection, since the SensorTag can detect if the disconnection is voluntary or not, the device will only advertise again once the button is pressed. This allows the device to go into a sleep state, saving power.

6. CONCLUSIONS AND FUTURE WORK

In this thesis the impact of sensors in AAL environments while using mobile devices was investigated. Sensor access was also an object of study in both Android and Firefox operating systems, along with an external device that comprises both sensors and other features that can help monitoring patients who live alone in their home places.

Regarding sensor access in mobile Open Source Operating Systems it is possible to say that Android is the one with better results. This was already expected since it is the "older" Operating System among the ones studied. Firefox OS still has some limitations that maybe can be surpassed in the future.

It was verified that it is possible to increase the number of sensor values in both Android and Firefox OS Operating Systems by making some changes in the source code, since the output data rates are limited by software and this is not an imposition of the hardware maximum output data rates. Sampling frequency was improved almost to the maximum hardware data rates, in the Android's case, which was clearly a good result. Unfortunately, sensor access in Ubuntu Touch and Tizen could not be studied because these Operating Systems are not yet fully implemented.

The use of external sensors combined with a smartphone was also object of study with the implementation of some BLE profiles like the Proximity Profile and the Call for Help profile that can be used to make BLE devices to directly interact with each other, performing some tasks that can be crucial in the AAL environments and at patients' homes. With SensorTag is also possible to achieve the sensor data rates wanted making it possible to integrate this device with the Fraunhofer AICOS projects like Mover or indoor location.

Future Work

Along this thesis some limitations were faced such as the unavailability of sensor APIs or devices to test Ubuntu Touch and Tizen. Some aspects that can be better analyzed in the future are:

- Test other Android devices and real Firefox OS devices, other that Nexus 10 and Nexus S.
- Test Ubuntu Touch and Tizen sensor access.
- The repercussions of increasing the sampling rate of the sensors over the battery consumption were not investigated, and this is very important for a real use scenario. Battery tests in SensorTag should be performed as well.
- It may be possible to incorporate some activity monitoring and fall detection algorithms in the SensorTag and not only on the smartphone. Right now, the device only sends unanalyzed sensor data to the smartphone.
- Developing a prototype with a buzzer in order for the device to make sound alerts, making the Find Me Profile useful.

References

[1] "AAL4ALL", www.aal4all.org, [Accessed: August 2013].

[2] "Pendant Fall detector", http://www.emcare360.com/web/emcare/services?serviceId=202&allDevices=true&doAsUserId=, [Accessed: August 2013].

[3] "The Vision of Smart Systems", http://www.smart-systems-integration.org/public/documents/publications/EPoSS%20Strategic%20Research%20Agenda%202009.pdf, [Accessed: July 2013].

[4] "Wearable devices", http://www.jneuroengrehab.com/content/9/1/21, [Accessed: August 2013].

[5] LASHKARI, Arash Habibi, MORADHASELI, Mohammadreza, "*Mobile Operating Systems and Programing: Mobile Communications*", VDM publishing, 2011.

[6] BAO, M. H., "*Handbook of sensors and actuators*", Elsevier B. V. 2000, ISBN 0-444-50558-X

[7] "Sensewear Armband" http://sensewear.bodymedia.com/SW-Learn-More/Product-Overview, [Accessed: August 2013].

[8] "Emcare" https://www.emcare360.com/web/emcare/services?allDevices=true&doAsUserId=, [Accessed: August 2013].

[9] "Tait-Bryan Angles", http://commons.wikimedia.org/wiki/Tait-Bryan_angles, [Accessed: June 2013].

[10] "Sampling rates for analog sensors" http://www.embedded.com/design/prototyping-and-development/4024581/Sampling-rates-for-analog-sensors, [Accessed: September 2013].

[11] WILSON, Jon S., "*Sensor Technology Handbook*", Elsevier 2005. ISBN 0-7506-7729-5.

[12] HORNUNG, Mark R.; BRAND, Oliver, "*Micromachined ultrasound-based Proximity sensors*", Kluwer

[13] "Light Sensors", http://www.maximintegrated.com/app-notes/index.mvp/id/5051, [Accessed: October 2013].

[14] LABRADOR, Miguel A.; YEJAS, Oscar D. Lara, "*Human Activity Recognition: Using wearable Sensors and Smartphones*", Taylor and Francis Group 2013.

[15] "Gyroscope Uses", http://www.gyroscopes.org/uses.asp, [Accessed: August 2013].

[16] OJETOLA, Olunkule; GAURA, Elena I.; BRUSEY James – "*Fall detection with Wearable Sensors*" – SAFE, Coventry University (2011)

[17] "GATT base profiles" it https://www.bluetooth.org/en-us/specification/adopted-specifications, [Accessed: August 2013].

[18] "Mover, Fraunhofer AICOS", http://mover.projects.fraunhofer.pt/faq.html, [Accessed: October 2013].

[19] STROUD, Adam; MILETTE, "*Professional Android Sensor Programming*", John Wiley and Sons, 2012. ISBN: 978-1-118-18348-9

[20] GUPTA, Naresh, "*Inside Bluetooth Low Energy*", Artech House, 2013

[21] SMITH, Phil, CSR – "*Comparisons between Low Power Wireless Technologies*" (2011)

[22] "Wi-Fi Direct", http://www.wi-fi.org/discover-and-learn/wi-fi-direct , [Accessed: September 2013].

[23] DE SANCTIS, Constantino; AFIFI, Hossam; LABIOD, Houda, "*Wi-Fi, Bluetooth, Zigbee and WiMax*", Springer 2007

[24] "CyanogenMod", http://www.cyanogenmod.org/, [Accessed: October 2013].

[25] "Zigbee FAQ", http://www.zigbee.org/Specifications/ZigBee/FAQ.aspx, [Accessed: September 2013].

[26] VILLEGAS, Julio – Bluetooth Low Energy Version 4.0 – "*Helping create the "internet of things*""

[27] "Android sensor overview", http://developer.android.com/guide/topics/sensors/sensors_overview.html, [Accessed: October 2013].

[28] "Texas Android Sensor Porting Guide" http://processors.wiki.ti.com/index.php/Android_Sensor_PortingGuide, [Accessed: September 2013].

[29] "SDK status Ubuntu Touch" https://wiki.ubuntu.com/Touch/SdkFeatureStatus, [Accessed: November 2013].

[30] "IAR", http://www.iar.com/Products/IAR-Embedded-Workbench/, [Accessed: September 2013].

[31] "CC2541 characteristics", http://www.ti.com/lit/ds/symlink/cc2541.pdf, [Accessed: September 2013].

[32] "DC-DC converter", http://www.ti.com/lit/ds/slvsac3c/slvsac3c.pdf, [Accessed: September 2013].

[33] "Btool", http://processors.wiki.ti.com/index.php/Category:CC2540DK_Mini_BTool_Connection, [Accessed: August 2013].

[34] "BLE device monitor", http://processors.wiki.ti.com/index.php/BLE_Device_Monitor_User_Guide, [Accessed: August 2013].

[35] "Wi-Fi 802.11 standard", http://www.ieee802.org/11/, [Accessed: September 2013].

[36] "WIFI article", http://www.infowester.com/wifi.php, [Accessed: September 2013].

[37] "WiFi Alliance", http://www.wi-fi.org/, [Accessed: October 2013].

[38] "Ethernet", http://standards.ieee.org/about/get/802/802.3.html, [Accessed: September 2013].

[39] "Wi-Fi Direct Specification", http://www.wi-fi.org/knowledge-center/faq/does-specification-underlying-wi-fi-direct-certification-program-work-both, [Accessed: September 2013].

[40] “Wifi p2p android”, http://developer.android.com/guide/topics/connectivity/wifip2p.html, [Accessed: September 2013].

[41] “AOSP”, http://source.android.com/, [Accessed: October 2013].

[42] “Zigbee Alliance”, http://www.zigbee.org/, [Accessed: October 2013].

[43] “Wireless IEEE 802.15.4 standard”, http://standards.ieee.org/about/get/802/802.15.html, [Accessed: October 2013].

[44] “Bluetooth Specifications”, https://www.bluetooth.org/en-us/specification/adopted-specifications, [Accessed: September 2013].

[45] “Bluetooth basics”, http://www.bluetooth.com/Pages/Basics.aspx, [Accessed: October 2013].

[46] “BLE specification”, https://developer.bluetooth.org/TechnologyOverview/Pages/v4.aspx, [Accessed: October 2013].

[47] GOLMIE N.; REBALA O.; CHEVROLLIER, N.*;* - “*Bluetooth Adaptive Frequency Hopping and Scheduling*”, National Institute of Standards and Technology (2003)

[48] “GATT specifications”, https://developer.bluetooth.org/gatt/Pages/GATT-Specification-Documents.aspx, [Accessed: August 2013].

[49] “Indoor Atlas”, https://www.indooratlas.com/, [Accessed: August 2013].

[50] “Indoor Atlas study”, http://gigaom.com/2013/09/25/indooratlas-uses-geomagnetism-to-map-buildings-gps-cant-reach/, [Accessed: September 2013].

[51] “Geomagnetismo”, http://www.ipma.pt/pt/enciclopedia/geofisica/geomagnetismo/index.html, [Accessed: November 2013].

[52] “Proximity sensors”, http://www.fargocontrols.com/sensors.html, [Accessed: October 2013].

[53] “Industry Leaders Announce Open Platform for Mobile Devices”, http://www.openhandsetalliance.com/press_110507.html, [Accessed: June 2013].

[54] “Apache License 2.0”, http://www.apache.org/licenses/LICENSE-2.0/, [Accessed: November 2013].

[55] “Mobile Application Development using Java and Android”, http://rockcode.in/Mobile-Application-Development-using-Java-and-Android.php, [Accessed: June 2013].

[56] “Android Website”, www.android.com/, [Accessed: October 2013].

[57] “Android Architecture”, http://android.pk/android.html, [Accessed: September 2013].

[58] “Eclipse IDE”, http://www.eclipse.org/, [Accessed: September 2013].

[59] “Developer Android”, http://developer.android.com/index.html, [Accessed: September 2013].

[60] “Firefox OS Architecture”, https://developer.mozilla.org/en-US/docs/Mozilla/Firefox_OS/Platform/Architecture, [Accessed: September 2013].

[61] “Firefox OS platform”, https://developer.mozilla.org/en-US/docs/Mozilla/Firefox_OS/Platform, [Accessed: September 2013].

[62] “Firefox Tools”, https://developer.mozilla.org/en-US/docs/Tools, [Accessed: September 2013].

[63] “Build Firefox OS”, https://developer.mozilla.org/en-US/docs/Mozilla/Firefox_OS/Firefox_OS_build_prerequisites, [Accessed: September 2013].

[64] “Firefox Hardware Access APIs”, https://developer.mozilla.org/en-US/docs/WebAPI, [Accessed: September 2013].

[65] “Device Orientation Event”, http://dev.w3.org/geo/api/spec-source-orientation, [Accessed: September 2013].

[66] “Ubuntu Touch Website”, https://wiki.ubuntu.com/Touch, [Accessed: September 2013].

[67] “Ubuntu Developer Website”, http://developer.ubuntu.com/, [Accessed: September 2013].

[68] “Tizen Overview”, https://www.tizen.org/about, [Accessed: September 2013].

[69] “Tizen Architecture”, https://developer.tizen.org/help/index.jsp?topic= %2Forg.tizen.gettingstarted%2Fhtml%2Ftizen_overview%2Ftizen_architecture.htm, [Accessed: September 2013].

[70] “Tizen Developer”, https://developer.tizen.org/, [Accessed: September 2013].

[71] “QEMU”, http://wiki.qemu.org/Main_Page, [Accessed: September 2013].

[72] “Reference device-PQ”, https://wiki.tizen.org/wiki/Reference_Device-PQ, [Accessed: September 2013].

[73] “Reference device-210”, https://wiki.tizen.org/wiki/Reference_Device-210, [Accessed: September 2013].

[74] “SensorTag”, http://www.ti.com/tool/cc2541dk-sensor, [Accessed: October 2013].

[75] “Bluetooth Low Energy Profiles”, https://developer.bluetooth.org/TechnologyOverview/Pages/BLE.aspx, [Accessed: September 2013].

[76] “InvenSense website”, http://www.invensense.com/, [Accessed: November 2013].

[77] “AKM website”, http://www.akm.com/, [Accessed: November 2013].

[78] “Qt Creator”, http://qt-project.org/search/tag/qt~creator, [Accessed: November 2013].

[79] “MPU 6050”, http://invensense.com/mems/gyro/mpu6050.html, [Accessed: October 2013].

[80] “AK8963 magnetometer”, http://www.akm.com/akm/en/file/datasheet/AK8963.pdf, [Accessed: October 2013].

[81] “Gyroscope in smartphones”, http://www.mobile88.com/news/read.asp?file=/2012/4/21/20120421165938, [Accessed: November 2013].

[82] CHOWDARY, Rahul D; PRAKASH, Bhanu K., “*El Electronic Guard for Blind*”, 2010

[83] “Zigbee technology”, http://www.eetimes.com/document.asp?doc_id=1276404, [Accessed: November 2013].

[84] “Bluetooth article”, http://www.infowester.com/bluetooth.php, [Accessed: November 2013].

[85] “Litepoint”, “*Bluetooth Low Energy whitepaper”,* 2012.

[86] “Firefox OS article”, https://hacks.mozilla.org/category/firefox-os/by/comments/as/complete/, [Accessed: November 2013].

[87] “HTML5 article”, https://hacks.mozilla.org/category/html5/as/complete/, [Accessed: November 2013].

[88] “Ubuntu porting guide”, https://wiki.ubuntu.com/Touch/Porting, [Accessed: November 2013].