Departamento de Engenharia Informática
Instituto Superior de Engenharia do Porto

Optimized Near Real-time Remote Device Tracking and Monitoring System

Joni de Oliveira Dias

Dissertação para obtenção do Grau de Mestre em
Engenharia informática

Área de Especialização em
Arquitectura, Sistemas e Redes

Orientador
Prof. Dr. Luís Miguel Moreira Lino Ferreira

Júri

Presidente:
Prof. Dra. Maria de Fátima Coutinho Rodrigues, Professora Coordenadora, ISEP

Vogais:
Prof. Dr. Luís Miguel Pinho Nogueira, Professor Adjunto, ISEP
Prof. Dr. Luís Miguel Moreira Lino Ferreira, Professor Adjunto, ISEP

Porto, Outubro 2011

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank all the people that, more or less directly, helped me throughout all these years of work, to obtain a master's degree.

First and foremost, I would like to thank my advisor, Prof. Dr. Luís Lino Ferreira, who made me believe that it is always possible to improve what we think it is already perfect. Thank you for all your support, guidance and preoccupation throughout the whole process. But more than that, I thank you for all the freedom you gave me, allowing me to follow my own ideas, and I also appreciate the tolerance you demonstrated, regarding some life adversities that crossed my way.

I would also like to thank my family, specially my parents which always believed in me and provided me with everything I ever needed to be a successful student. Both you are my pride. I want to thank my girlfriend, who always kept me motivated to keep up with my goals, for tolerating all the time this project stole from her. I would also like to thank my beloved aunt which helped me to overcome one of the most difficult phases of my life. You know who you are and the special place you have in my heart.

Finally, I must thank my closest friends who were always there when I most needed. Among them all, I want to thank Pedro in particular, for his preoccupation and support during my tough times. You are like a brother to me.

RESUMO

Os serviços baseados em localização vieram dar um novo alento à criatividade dos programadores de aplicações móveis. A vulgarização de dispositivos com capacidades de localização integradas deu origem ao desenvolvimento de aplicações que gerem e apresentam informação baseada na posição do utilizador. Desde então, o mercado móvel tem assistido ao aparecimento de novas categorias de aplicações que tiram proveito desta capacidade. Entre elas, destaca-se a monitorização remota de dispositivos, que tem vindo a assumir uma importância crescente, tanto no sector particular como no sector empresarial.

Esta dissertação começa por apresentar o estado da arte sobre os diferentes sistemas de posicionamento, categorizados pela sua eficácia em ambientes internos ou externos, assim como diferentes protocolos de comunicação em tempo *quase*-real. É também feita uma análise ao estado actual do mercado móvel.

Actualmente o mercado possui diferentes plataformas móveis com características únicas que as fazem rivalizar entre si, com vista a expandirem a sua quota de mercado. É por isso elaborado um breve estudo sobre os sistemas operativos móveis mais relevantes da actualidade. É igualmente feita uma abordagem mais profunda à arquitectura da plataforma móvel da Apple - o iOS – que serviu de base ao desenvolvimento de uma solução optimizada para localização e monitorização de dispositivos móveis.

A monitorização implica uma utilização intensiva de recursos energéticos e de largura de banda que os dispositivos móveis da actualidade não estão aptos a suportar. Dado o grande consumo energético do GPS face à precária autonomia destes dispositivos, é apresentado um estudo em que se expõem soluções que permitem gerir de forma optimizada a utilização do GPS.

O elevado custo dos planos de dados facultados pelas operadoras móveis é também considerado, pelo que são exploradas soluções que visam minimizar a utilização de largura de banda.

Deste trabalho, nasce a aplicação *EyeGotcha*, que para além de permitir localizar outros utilizadores de dispositivos móveis de forma optimizada, permite também monitorizar as suas acções baseando-se num conjunto de regras pré-definidas. Estas acções são reportadas às entidades monitoras, de modo automatizado e sob a forma de alertas. Visionando-se a comercialização da aplicação, é portanto apresentado um modelo de negócio que permite obter receitas capazes de cobrirem os custos de manutenção de serviços, aos quais o funcionamento da aplicação móvel está subjugado.

Palavras-chave: Localização e monitorização móvel, Dispositivos móveis, XMPP, Jabber, iOS, iPhone

ABSTRACT

Location-based services boosted mobile developers' creativity. When portable devices with integrated location features did first appear, that led to the arrival of applications that manage and shows information based on the user's current position. Since then, the mobile market has witnessed the emergence of new application categories that take advantage of this capability. Among them all, there is the remote device monitoring category which owns a growing importance in both private and business fields.

Given that mobile devices are enclosed within a large number of location tracking systems, this work presents the current state of the art on different positioning systems and their underlying technologies. These systems are grouped into two distinct categories: indoor and outdoor positioning systems. Monitoring activities require communication between devices, therefore different near real-time communication protocols are discussed. The current state of the mobile market is also presented.

There are many different rivaling mobile platforms with unique features, thus a brief study about the current and most relevant mobile operating systems is done. Since this work led to the development of an optimized mobile tracking and monitoring solution for the Apple's mobile platform - iOS – its architecture is also approached.

Monitoring activities consume high energy resources and considerable amounts of traffic, which current mobile devices cannot support. Given the large power consumption of the GPS and bearing in mind mobile devices poor autonomy, it is presented a study that provides solutions for an optimized management of the GPS activity.

Data plans provided by mobile operators are expensive and traffic limited, hence solutions in order to minimize the bandwidth used between monitoring entities are explored.

The *EyeGotcha* application was originated from this work, which serves the purpose of allowing users to locate other mobile devices and monitor their actions through a set of predefined rules. These actions are reported to the monitoring entity, in an automated alert triggering fashion. Foreseeing the placement of the application in the market, it is presented a business model that allows generating enough profits to cover maintenance costs for the services which the mobile application depends on.

Keywords: Mobile Device Tracking and Monitoring System, Location-aware, XMPP, Jabber, iOS, iPhone

CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

AAS	Area Augmentation System
AGPS	Assisted Global Positioning System
AOA	Angle-Of-Arrival
CDMA	Code Division Multiple Access
DGPS	Differential GPS
EGNOS	European Geostationary Navigation Overlay Service
GAGAN	GPS Aided Geo Augmented Navigation
GBAS	Grounded-based Augmentation System
GLONASS	Global Navigation Satellite System
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GRAS	Ground-based Regional Augmentation System
ISR	Interrupt Service Routine
LAAS	Local Area Augmentation System
MSAS	Multi-functional Satellite Augmentation System
MVC	Model-View-Controller
NLOS	Non-Line-of-Sight
PPS	Precise Positioning System
RSSI	Received Signal Strength Indicator
RTT	Round Trip Time
SBAS	Satellite-based Augmentation System
SPS	Standard Positioning Service
TDOA	Time Difference-Of-Arrival
TOA	Time-of-Arrival

TOF	Time-of-Flight
TTFF	Time-to-First-Fix
UDID	Universally Device Identifier
UUID	Universally Unique Identifier
UWB	Ultra-Wideband Radio
WAAS	Wide Area Augmentation System

CHAPTER 1

INTRODUCTION

Now more than ever, location-based services have already secured a dominant position in the mobile market. Mobile applications fitted in this category are distinguished from all the others considering the fact they are able to deliver online content to the users, based on their physical location. The rise of this type of applications for mobile devices was possible thanks to the inclusion of location technologies like GPS receivers on those equipments. Besides that, the improvement and use of existing communication mechanisms, such as the cell phone infrastructure and wireless access points, were ways to get around GPS indoors uselessness, allowing self-tracking almost everywhere (Educause, 2009).

The mobile location-aware category already covers a large range of different purposes based on the user's location. Within the billing subcategory, there can be found applications which accesses users' tracking information for automated payment processing and electronic tolling (TRUSTe, 2010).

In the productivity area, the number of location-based applications is growing. Applications such those which allow task remembering based on user's location are just an example. Task Ave (Task Ave, 2011) is the name of an application where the user is able to assign a certain task to the specific place where that task can only be done. For instance, when a user wants to be recalled of buying milk when he gets close to a store, applications like this one can handle the situation.

Also the gaming section keeps attracting location-awareness attention. It includes games where users are dared to do perform challenges at places. SCVNGR (SCVNGR, 2011) is an example of such kind of games. By going to places and doing challenges, players can earn points which then can be used to unlock badges and real-world rewards, such as discounts and free items.

Discovery is likewise a very well attended category, regarding location-based applications. This kind of applications is usually targeted to the people who like to travel and explore different places. Abandoned (Abandoned, 2011) is the name of an application that fits in this category. It shows recommendations about abandoned places and reveals their history along with related photos.

The social-networking section has also many location-aware capabilities in their applications. Some of these features may include things like allowing a user to share his location with their friends or to get informed about their current location, thus knowing about their "places of worship" and even about the social events they attend to. Gowalla (Gowalla, 2011) and Foursquare (Foursquare, 2011) are both mobile applications which fits into this category.

When it comes to the need for controlling other people, another subcategory is found. The GPS receiver which comes embedded in most of today's mobile devices brought a new set of possibilities to the applications' development. Among them all, there is the one which turns regular phones into monitoring devices. Examples of such applications are the ones capable of tracking another device at any time, knowing information like its current location and state (whether if it is moving or not, and if it is, in which direction, etc.) as well as the path that it

has followed. It is the monitoring category that the application which resulted from this thesis' research fits into.

1.1. The Problem

When someone wants to tell another person that they just did reach or did leave a location, he usually turns himself to the regular and well-known systems: phone calls or SMS. Two facts are evident in this situation: the user needs to interact with the mobile device in order to inform other people about his new location and he also needs to recall of doing that, which turns out it is somehow difficult for some persons. Thus, the location-awareness may be used to bring automation to the monitoring process.

Developing a mobile application in order to solve this problem raises some other complications to be addressed at three different levels: monitoring methodology, devices' poor battery life and bandwidth consumption.

Monitoring solutions for mobile phones we may find, they usually miss an essential point of view regarding the monitoring methodology: a person does not want to be constantly watching other people to find something wrong; it wants to be alerted if they do something wrong (or something that may require its attention). This means that people prefer to be alerted to an event rather than be constantly looking if it happens.

As we know, over the time mobile phones tended to include larger screens and more hardware like GPS receivers or wireless communication technologies. The problem is that the technologies behind the batteries that keep our phones alive, have failed to follow the evolution of the remaining components. Several mobile applications just ignore that fact and fail in managing the power consumption of those components.

Regarding bandwidth concerns, mobile operators' data plans with high plafond limits are still not affordable to all users. Hence, applications need to apply some optimization techniques in order to reduce the amount of traffic between the different entities. It is important to note that this may provide much more benefits in several other areas than having a simple economic impact on users' budget. These benefits may include, for instance, performance and battery life improvements when using an application.

There are at least two large market segments that may be interested in a solution which addresses this problem. The first one is the parental segment. Looking at parents' controlling needs, the ability to know at any time the location of their children is undoubtedly a feature targeted at an already well-defined market niche. Any father would be relieved just by knowing that he would be alerted, in real-time, if their children leave the school during class hours, knowing also where they are at that exact moment and where they have been lately.

The need for location and monitoring stands out when we talk at an organization level. This is the second market segment that can benefit from a solution for this problem. The employees need mobility and the organization needs control over it. The fact of being possible to know the employees devices' location, not only allows to control the performance of their underlying activities in an efficient manner but it also helps to decrease the response time to different day-to-day adversities (delays due to traffic congestions, need for replacement by another employee who is closest to the destination, tracking of stolen equipment, etc.).

1.2. The Solution

In order to solve the problem mentioned in section 1.1, it is intended to develop an application for the iOS platform mobile devices (refer section 3.4), which is able to remote monitoring any other devices running the same application. The three main features are current location tracking, location history tracking and event-based alert notifications.

1.2.1. Application Overview

Figure 1 depicts a possible scenario aiming to the parental market segment.

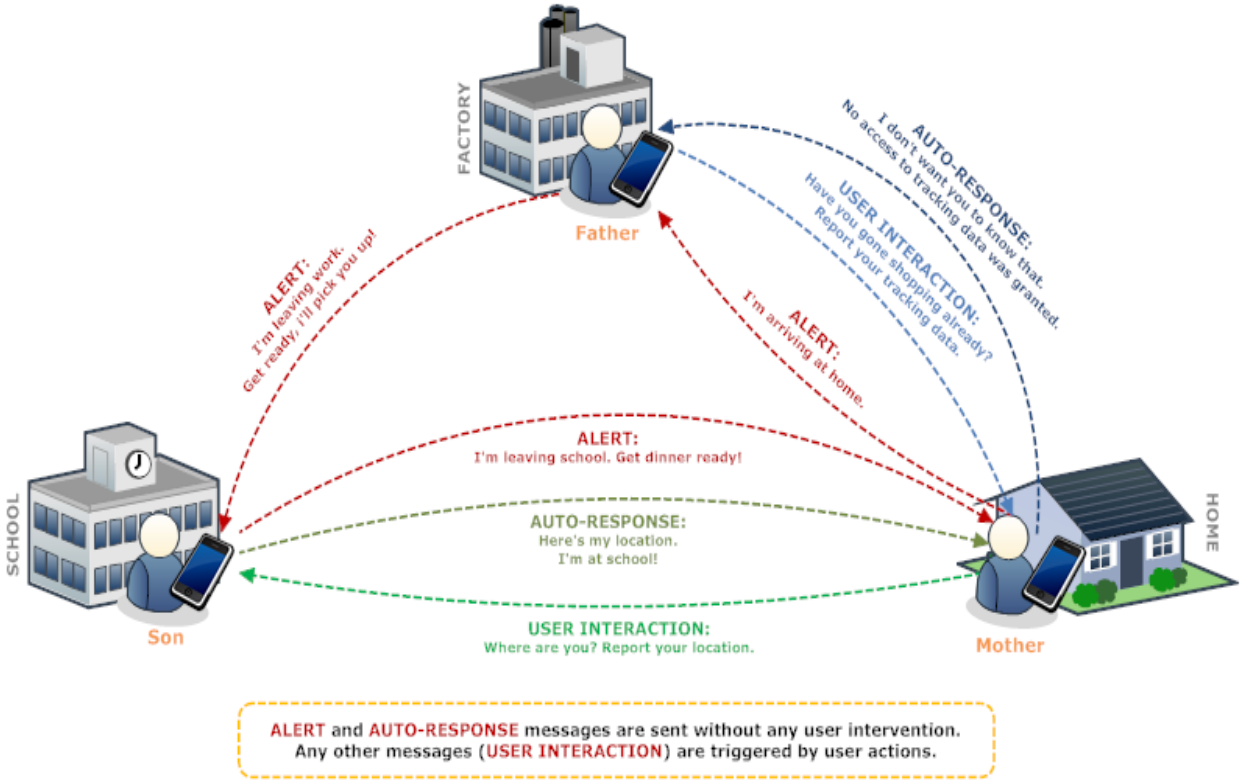

Figure 1 – Conceptual Application Scenario

In Figure 1, there are shown three family entities: the father, the mother and their son. Each of the entities is at a different place, where they can be arriving to or leaving from, at some moment. The application is installed in each entity's mobile device which has access to the Internet.

In the application there is a feature called *connection list*. This list presents all the devices a user is paired with. *Pairing* is a virtual contract that establishes the free will of exchanging messages between both parties. Both the entities are paired with each other. This means that each of the entities appears in each other's *connection list* so that they can set alerts and request tracking data from anyone.

The following set of alert configurations was set, prior to the events depicted in Figure 1:

1. The father had set in its mobile device's application that he wants to be alerted whenever the mother arrives home.
2. Since the mother wants to know when her son leaves the school, in order to prepare dinner, she did set that alert in her mobile device.
3. The son has set an alert to be notified whenever the father leaves work, during his school schedule, in order to be ready when his father comes to pick him up.

The following order of events occurs:

1. The mother arrives home.
2. The application running in the mother's mobile device detects the event and automatically sends a notification to the father, just as it was previously configured in the father's device.
3. The father's mobile device receives the notification.
4. Since he notices that she arrived home too soon, he is in doubt if she went shopping already. Hence, he uses the application in order to request her tracking data which reveals where she has been lately. The application sends the request to the mother's mobile device.
5. The mother's mobile device receives the request. Since the mother did not want the father to know where she usually goes, she already had set the application privacy options to not send that kind of data to him.
6. The application running in the father's mobile device alerts him that he does not have permission to access to the mother's tracking data.
7. The father then leaves work. By doing this, the application notices that event and just as it was configured in the application running in son's mobile device, the son receives a notification informing that his father just left the work. Hence, he knows he has to get ready because he's father is coming to pick him at school.
8. The father arrives, picks up the son and leaves the school area. Since the mother had configured an alert to notify her when her son leaves the school, the son's mobile device sends a notification to her. Thus, she learns that she should start preparing the dinner.

1.2.2. Application Requirements

The application should attend to some requirements in order to provide all the monitoring features presented in section 1.2.1.

Users should be able to see location tracking information about monitored users on a map as well as other additional information such as their current speed and direction. Users should also be able to see location information about themselves in a similar way.

In order to exchange location data, users need an authentication system that provides them with an exclusive identity in order to identify the sender and the receiver of a message, within the whole system. This system should be an automated (preferably without manual account registering) and secure authentication mechanism.

In addition to that, a pairing system is needed. A pairing system is used to establish a virtual contract between two different entities. Each contract defines that two entities are connected to each other by free choice, in a way that they can exchange information between them.

Thus, when a user wants to track another, there must be done the pairing process between them, in order to the communication to be consented by both parties.

Due to the fact that an Internet connection may not be always available, it is necessary that this pairing system contemplate many different scenarios, using the available technologies of the mobile device. The shared user ID should be easy to remember and, as the name implies, easy to be shared and entered on the application by other users.

The application should have global and individual privacy settings. While the global ones may affect all the users which a person is connected to, the individual settings affect only a certain user, thus allowing full control over the application's data flows. Yet regarding control and privacy, the application should provide a communications flow log in order to keep the user informed of all the incoming and outcoming data.

An optimized data transfer mechanism should be implemented, promoting bandwidth economization and also scalability, in order to avoid medium/long-term related problems due to exceeding the supported number of users.

The application should manage the GPS receiver state (enable or disable) by knowing whether the GPS data is needed to be acquired or not, and by knowing when to reactivate it without missing any situation which needs be monitored on time. This will improve the battery-life of the mobile devices since the GPS receiver will consume less energy.

The solution should rely on standardized protocols whenever possible. This measure is important in order to assure future interoperability with other devices and platforms. Hence, it turns possible to extend these project capabilities to other mobile devices' models or even other kind of electronic devices (vehicle embedded equipments, etc.).

A business model should also be developed in order to make profit of this location-aware application. The necessity to maintain external servers in order to keep communication and user data services up and running, demands a well-defined business model to be applied.

1.3. Thesis Overview

This thesis is organized in the following manner:

Chapter 1 presents the scope of this thesis by describing its context and the problem it is being attempted to solve. This chapter also gives a brief description of the solution that will be followed in order to solve the problem, and presents an overview about its requirements.

Chapter 2 introduces the state of the art on location systems and the mobile market. The location systems are grouped in two categories: outdoor location based systems and indoor location based ones. Related to the location systems are also the signal measuring techniques which are described along with a comparison between the different techniques used in several location systems. Mobile platforms' market share is also approached. Near real-time messaging protocols are also discussed.

Chapter 3 provides an overview about the current mobile operating systems available on the market. This chapter focuses on the iOS platform by giving it greater detail regarding the iPhone's hardware and software architecture.

Chapter 4 provides a full explanation of the architecture for the developed application. It discusses the communication protocols used and also the authentication and pairing mechanisms implemented. Solutions in order to minimize bandwidth requirements as well as to improve mobile device's battery life are also described.

Chapter 5 describes development steps and details for this solution. It gives also an overall view of its different modules and presents the relationship between them.

Chapter 6 approaches the business model to be followed in order to turn this application profitable. It describes annual profit projections for each approached model and presents ways to promote the application. A comparison between this application and other similar ones is also performed.

Finally, chapter 7 presents the conclusions about this work. The research context and objectives are described and all the accomplishments are identified. Some performed tests are presented and its underlying results are discussed. It is also presented future work based on current limitations and lack of some features that can be added to improve the application.

CHAPTER 2

STATE OF THE ART

Nowadays, most of mobile devices currently available in the market have attractive features (such as GPS receivers, Wi-Fi modules, among many other technologies) for either location-aware service providers as well as for those who seek great business opportunities regarding the strong potential that mobile software industry has achieved over the past few years.

Today, it is safe to state that the global position of a person over time is one of the most precious data, which many companies are willing to pay for in order to improve their strategies. Surprisingly or not, a simple piece of information like this can be incredibly powerful. It can help several business sectors to adapt their offers based on consumer's taste, culture and needs among other factors, allowing companies to promote their products focusing on those customers who may be more interested. In the same way, social services oriented sectors such as health care, eldercare and public security control, benefit from the use of people location data, being these just few practical examples.

This chapter will approach the state of the art on both outdoor and indoor location based systems. It will also discuss signal measuring techniques and compare its efficiency on several location systems. The state of the mobile applications' market is also evaluated. Near real-time messaging protocols are also discussed.

2.1. Outdoor Location Based Systems

Some decades ago, consumers worldwide were introduced relatively small electronic devices capable of receiving satellite signals - designated as Personal Navigation Assistant (PNA) or Personal Navigation Device (PND) – combining positioning capability and navigation functions. Due to the richness of features (mainly in the automobile navigation systems field) such as detailed maps, turn-by-turn guidance and voice instructions, those devices achieved great worldwide popularity.

Since then, we assisted to a fast evolution of mobile devices (such as mobile phones) regarding the ability to include more features while reducing overall size of electronic components at the same time. Thanks to that fact and to the available Global Navigation Satellite Systems (GNSS), today we can easily use a smartphone to determine our location (longitude, latitude, and altitude). Currently, there are a reasonable number of GNSS systems already available and fully operational, while there are others that are presently under development.

As of 2011, only two systems are fully operational. These systems are the United States' Global Positioning System (GPS) and the Russian's Global Navigation Satellite System (GLONASS). In development are Europe's Galileo and China's COMPASS. Considering the fact these last systems are global ones, it is important to mention that there are regional-only navigation systems. China's Beidou, French's DORIS, India's IRRNSS (Indian Regional Navigation Satellite System) and Japan's QZSS (Quasi-Zenith Satellite System) are both regional navigation systems while the last two are still in development.

2.1.1. Global Positioning System (GPS)

The U.S. Global Positioning System, a satellite-based radio navigation system, uses precise range measurements from GPS satellites in order to determine position and time, anywhere in the world. As of January 2011, it consists of up to 30 satellites, where each satellite completes an orbit in approximately every 12 hours. Currently, GPS provides two navigation services. The Standard Positioning Service (SPS) is intended for civilian use while the Precise Positioning System (PPS) is geared exclusively for military usage (United States Armed Forces, federal agencies and selected allied armed forces and governments) (Han *et al.*, 2010).

In order to maintain satellite data accuracy, GPS has a control segment consisting of five stations and four ground antennas, both capable of sending data to satellites. The monitor stations track all satellites in view and accumulate ranging data. This information is then processed by the master station and then transmitted via the ground antennas. Most smartphones' hardware available worldwide support this system exclusively.

The GPS also relies on a technology that provides assistance data from a communications network to assist in locking or on acquiring satellites quickly. It is called Assisted-GPS. This technology's acquired data typically includes information for determining a GPS receiver's approximate position, time synchronization mark, satellite ephemerides, and satellite dopplers (Thomson *et al.*, 2010). Since the identification of the satellites for which a device or GPS receiver should search is also provided, the mobile device attempts to acquire only the signals for those satellites. Most of todays' smartphones with GPS capabilities have this technology present in order to improve Time to First Fix (TTFF).

2.1.2. Global Navigation Satellite System (GLONASS)

Operated by the Russian Federation, GLONASS is a satellite navigation system that provides positioning, navigation and timing information, just like GPS. Its constellation consists of 24 satellites in three orbital planes (each one containing 8 satellites), being that the required number in order to provide navigation services worldwide. Intended to serve both civil and military purposes, this system uses two different types of signals, being them a Standard Precision signal (SP) and a High Precision signal (HP) (Han *et al.*, 2010).

The GLONASS system architecture includes a bigger control segment than GPS does (10 monitor stations and additional facilities, entirely located within former Soviet Union territory) to command and control the satellites.

Since its launch, GLONASS satellites suffered several upgrades, being the last version "GLONASS-K" (third generation) which offers broadcast of additional CDMA signals, including GPS and Galileo compatible navigational ones, thus promoting interoperability between both systems (Hegarty and Chatre, 2008)(Moore, 2009). "GLONASS-KM", the fourth generation, is in the requirement definition phase since 2002 and is purposed to be available by 2025 (Eissfeller *et al.*, 2007).

2.1.3. Galileo

Galileo is the European satellite navigation system, which is expected to be fully interoperable with the other radio-navigation systems. To the final user this means it will be possible to improve accuracy as well as increase redundancy through the use of other radio-navigation satellites (Hegarty and Chatre, 2008). Its constellation will consist of 27 satellites and 3 spares, being expected to be fully operational by 2014.

Regarding performance, Galileo will reach higher levels than GPS due to the more precise satellite clocks, faster satellite orbit determination and improved signal modulation (Eissfeller *et al.*, 2007; UNOOSA, 2010). Being a navigation system designed for civil and commercial purposes, it will offer a set of 4 different navigation services and yet another support service aimed to help in search and rescue operations. These services (Hegarty and Chatre, 2008; Eissfeller *et al.*, 2007) are:

- Open Service (OS): available to everyone and free of charge for the users.
- Commercial Service (CS): paid service, offering higher data rates and higher accuracy.
- Safety of Life Service (SoL): designed for aircraft landings and other critical operations.
- Public Regulated Service (PRS): can be used by the police, fire fighters and other institutions; also intended to work for military purposes (supports anti-spoofing and anti-jamming capabilities).
- Search and Rescue (SaR): to be used in search and rescue operations.

2.1.4. COMPASS

The Compass system (also known as Beidou-2) is China's global navigation system, will consist of 5 geo-stationary satellites and 30 non-stationary satellites, offering full worldwide coverage. Two services are planned to be available, being them an Open Service (civil usage) and an Authorized Service (authorized entities by Chinese government usage only like, for instance, Chinese military).

The first service provides free positioning, velocity and timing services, while the second one offers safer services already provided by the Open Service, as well as system integrality information. The full system is expected to be completely operational (providing global coverage) between 2015 and 2020 (Hegarty and Chatre, 2008).

2.2. Area Augmentation Systems

Sometimes a system like GPS alone is not able to provide satisfactory performance levels, especially when a high level of integrity is required. For example, airplanes need to have better than 15-meter accuracies in the vertical and horizontal directions (Das, 2010). GNSS augmentation systems (ICAO, 2005; CASA, 2006) are capable of improving navigation systems performance and reliability by providing additional information.

2.2.1. Satellite-based Augmentation System (SBAS)

Through the use of additional satellite-broadcast messages, this system delivers error corrections, extra ranging signals via geostationary satellites and integrity information for each monitored satellite. The European Geostationary Navigation Overlay Service (EGNOS) is the European SBAS that complements the GPS system. It is currently operational and is able to improve current GPS services from about 10 meters to just about 2 meters and keep it available more than 99 percent of the time (CASA, 2006).

Apart from EGNOS, there are other systems like the Wide Area Augmentation System (WAAS) for the USA, the Multi-functional Satellite Augmentation System (MSAS) for Japan, and the GPS Aided Geo Augmented Navigation (GAGAN) system planned for India and expected to be compatible with those last described (ICAO, 2005; CASA, 2006).

2.2.2. Grounded-based Augmentation System (GBAS)

The Grounded-based Augmentation System (GBAS) is a different approach to enhance GPS Standard Positioning Service (SPS) through the use of data obtained from the ground. The basic idea lies in the fact that, if two stations position are accurately known, then it is possible to find out the error by comparing the position data obtained from GPS with its accurate position. Then the calculated error data can be transmitted, for example, to aircraft GPS receivers allowing them to correct the position data and thus achieve higher accuracy (Das, 2010). The U.S. Local Area Augmentation System (LAAS) is an example of an implementation of ground based augmentation system (Murphy and Imrich, 2008). Figure 2 describes the overall architecture of GBAS.

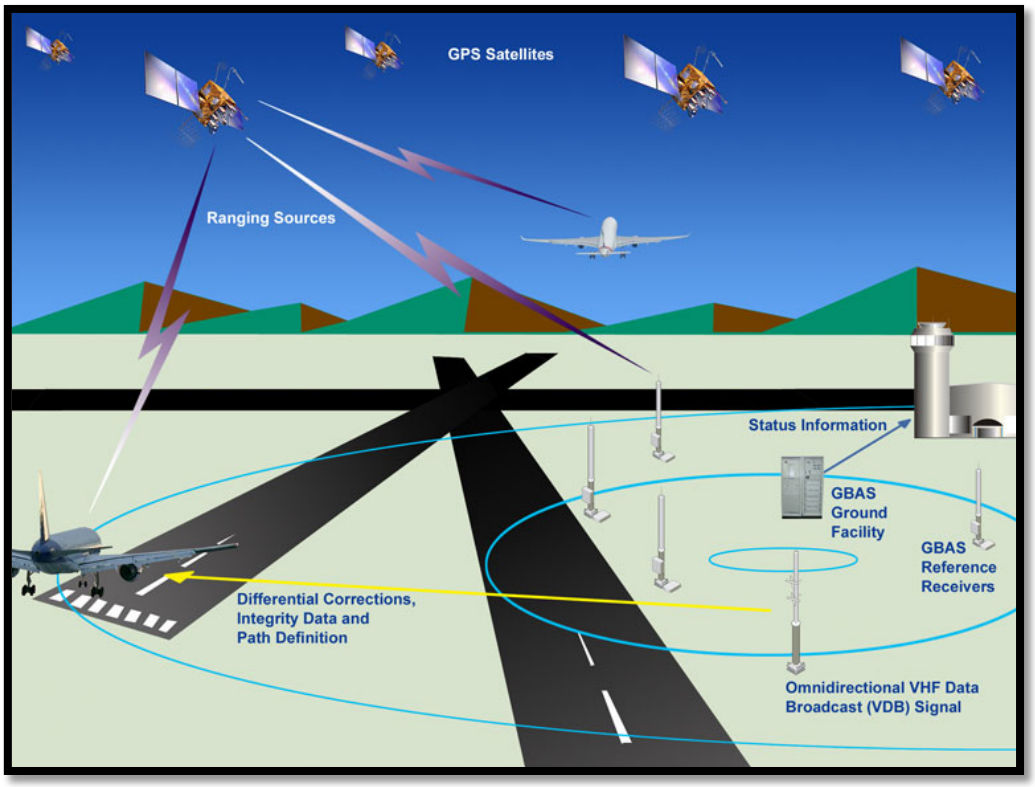

Figure 2 – GBAS Architecture (FAA, 2011)

There is also the Ground-based Regional Augmentation System (GRAS) which uses a distributed set of reference stations and centralized processing sites to compute differential GNSS corrections and integrity data. For that purpose, it relies on a VHF broadcast which uses the GBAS physical link and message format, rather than using geostationary satellites to broadcast data to users, as stated in (Hegarty and Chatre, 2008). Differential GPS (DGPS) (Morgan-Owen *et al.*, 1995) is an example of this kind of systems.

2.2.3. Hybrid Navigation Satellite Systems' Receivers

Due to the emergence of new global navigation satellite systems, device manufacturers eventually have to choose what they consider to be the best option for their customers as well as for their production techniques and/or costs. To minimize or even eradicate this situation, there are already manufacturers working on single-chips positioning devices for multiple global navigation systems.

ST Microelectronics (STMicroelectronics, 2011), a leading supplier of semiconductors for navigation and car infotainment, announced the first chip in the world provided with such capabilities early the year of 2011. Teseo II is the name given to the new generation of single-chip stand-alone positioning receivers for portable navigation devices, in-car navigation and telematics applications. They distinguish themselves from all others since they are capable of receiving signals from multiple satellite navigation systems, including GPS, GALILEO, GLONASS and QZSS.

In addition to the improvement of user position accuracy and navigation, in poor satellite visibility conditions, this new kind of chips also offer shorter signal-acquisition times by resorting to what they call Self-Trained Assisted GPS technology. Regarding their current status, the first chipset in the Teseo II family has its volume production scheduled for Q3 2011.

2.3. Indoor Location-based Systems

Presently, it is not possible to rely in only a single global positioning technology in order to achieve location data with an acceptable accuracy rate. The society in which we live, force us to spend the most part of our time inside buildings than in outside environments. In this last situation, GPS is doubtless the most powerful technologic resource able to provide us with the bigger accuracy rate regarding positioning data retrieving.

In the other hand, GPS is very limited to the environments in which it is able to operate, subjecting itself to a state of total uselessness when in places where satellites signal reception is precarious or completely inexistent. In order to successfully operate, GPS needs at least a signal level above -130dBm. This means that even some outside areas may be problematic ones for this kind of technology, such as forests and settlement areas, because attenuation obstructs signal reception and multi-path effects cause interference, just as the author explains in (Kofahl *et al.*, 2004).

GPS is also the most power consuming positioning method, regarding the fast battery drain problem when applied to the current mobile devices (Premchaisawasdi *et al.*, 2009). A feasible alternative found to surpass this limitation is to take advantage of the existing mobile network information, in order to identify the cell zone and sector (Cell-ID) where the user

currently is. A Cell is a section of coverage that is defined by a certain number of directional antennas, which are present in mobile phone stations.

Since Cell-ID based location is implemented the server-side, no modifications need to be made in the mobile devices, therefore this method works with any mobile phone. On the other hand, this positioning method is not a GPS substitute regarding accuracy as the Cell-ID based method is significantly lower, when compared to the last one. In fact, its accuracy depends on the size of the cell, which can be as low as 150 meters in urban environments, and as high as 30 kilometers in rural ones, as stated (Kofahl *et al.*, 2004).

The lack of precision may be a problem or not, depending on the application context. For example, a simple application that tells the user the current weather of the location he is, does not need position data to be just as precise as the one gathered by an application that needs to know the exact building where a person is. For emergency situations, tridimensional coordinates are much more valuable than regular dimensional ones.

Since Cell-ID based location works in almost any place covered with mobile network signals, and its power consumption is very low, it is a reasonable choice as a complement to GPS during its unavailability, and can even be improved if allied to some other indoor location methods, which are described further in this section. Usually, these indoor location technics rely on a variety of small electronic devices such as infrared, radiofrequency and ultra-sound sensors.

2.3.1. Hybrid Positioning Systems

Hybrid positioning systems are designed for locating mobile devices through the combination of several different positioning technologies such as GPS, cell tower triangulation, Wi-Fi signals, and Bluetooth sensors among others (Arthi, 2010). They are expected to provide the most accurate positional data considering many different scenarios, where signals' reception is interchanged between different sources.

Among the commercial solutions that use GPS signals, Wi-Fi access points and cell tower IDs in order to provide location tracking, there is the Core Engine from Skyhook (SkyhookWireless, 2011). It is a software-only based location system that allows acquiring the most accurate positional information possible, based only on those three sources. The system can be deployed in a large set of mobile devices and it is able to provide accuracy within a 10 to 20 meters range. It is also important to refer that it provides a TTFF of just about one second, which is very fast and it is constantly availability indoors and outdoors.

In order to work, a mobile device with a location system embedded collects raw location data from each of the signal sources. That information is then transmitted to a server, which in turn, will only return back a single location estimate to the mobile device.

The process that occurs in the server during the position calculation phase, makes use of an worldwide reference database of Wi-Fi access points and cell tower ID's, in addition to the raw position data obtained from each signal source. To provide continued accuracy through the time, this location system recalibrates its data with the one provided from clients which use the service. In short, this means that the more devices make use of it, the more updated and refreshed the database is.

Another related and interesting system is Navizon (Navizon, 2011), a positioning system that combines Wi-Fi and cellular triangulation, which supports GSM, CDMA and 3G networks all

around the world. Similar to Skyhook's system, this one uses the same approach to increase its database with Wi-Fi and Cellular towers' locations by relying on its users to collect this kind of information.

In addition to a client-side SDK be available, Navizon also offers a web service API that allows developers who already have Cell-ID and/or Wi-Fi Mac address information, to just use it to obtain a geographical location through a simple *http* query. In addition to a pair of coordinates (latitude and longitude), the response also contains a radius of confidence in order to have a perception of the accuracy of the returned data.

Another valuable feature is the fact that Navizon provides cell tower maps, which can be installed on the devices. This basically means that A-GPS is provided without connectivity to the network (Hallauer, 2010). Apple's iPhone used to rely on Google and Skyhook Wireless to handle location-based services. However, since the release of iPad in April 2010, Apple switched to its own proprietary location technology (IpodNN, 2010).

2.3.2. Bluetooth Indoor Location Services

Nowadays, Bluetooth technology is present in almost every mobile phone in the market. Due to its cheapness and low-power consumption, industries began thinking of it as a good way to bring accurate location indoors. That is the case of WirelessWERX (Wirelesswerx, 2011a), a cloud-based indoors location service platform, able to deliver precise location to mobile phones.

The location information provided by the referred system includes an altitude coordinate, greatly improving positioning accuracy. The way it works is based on a Bluetooth Node Network comprised by low-power nodes as well as a Bluetooth IP-Bridge, which is responsible for delivering mobile device location to a server. Its API is available through a nice range of mobile operating systems including Apple's iOS, Android, BlackBerry and Windows Mobile. While this kind of technology may serve numerous purposes, it is undeniable that a large portion of them is related to marketing services.

WirelessWERX holds a mobile marketing technology that allows brands to influence their customers indoors and to ease the gathering of statistic information about them. This technology is known by MobiWERX (Wirelesswerx, 2011b). It is based on WirelessWERX platform and features a good content management system. It allows building location-based campaigns while ensuring the elimination of spamming concerns, since it is permission-based.

SiteWERX (Wirelesswerx, 2011c) is also another variant of WirelessWERX platform suited for mobile emergency, first response and security customers. Its main purpose is to allow a mobile phone caller's exact location to be transmitted, when making a 911 call from inside a building or within a dense urban environment. Safety zones are delineated by a location node network so that when calls are made anywhere inside these areas, the location information containing which building, floor and room the caller is, is sent to the communication center.

2.3.3. Real-time Location Systems

Real-time Location Systems (RTLS) are used to track and identify the location of objects in real-time. For that purpose, tags are used in order to identify the objects as well as devices capable of receiving and processing wireless signals coming from these tags to determine their current location (Clarinox, 2009).

Currently there is a significant number of commercial systems and prototypes being able to provide indoor location with high accuracy. Those infrared-based ones available in the market can determine a person's position through the usage of electronic badges that emit a unique identifier code, which in turn is identified by a sensors' network able to recognize those infrared signals. Despite their small size, ultra-low power consumption and cheapness, it is easily understandable that this kind of implementation may lead to high demands in terms of maintenance while at the same time, may provide a low level of scalability.

Ubisense (Ubisense, 2011) is an example of a real-time location system able to provide 30 cm 3D tracking accuracy, both indoors and outdoors. It relies on tags capable of transmitting ultra-wideband (UWB) radio signals to a network of fixed sensors. In turn, this network uses those signals to locate the tag's positions, by combining its Angle-of-Arrival and Time-Difference-of-Arrival technologies (Porrmann, 2009). These signal measuring techniques are approached later.

Other indoor location systems make use of radiofrequency and ultra-sounds sensors. Two widely known public solutions are RADAR and Cricket (Paul and Wan, 2008). RADAR system operates by storing and processing signal strength information of several stations, which are properly located in order to overlap its coverage under the various units in the area of interest (Bahl *et al.*, 2000).

The RADAR system compares the user's RSSI (Received Signal Strength Indication) with a set of signal strength measurements already stored. This type of information is usually referred to as fingerprints. Considering the main drawbacks of this method, we may refer to the need for extensive coverage learning as well as the poor extrapolation in the areas that were excluded during the learning process (Paul and Wan, 2008).

Compared to the RADAR system, Cricket differs from it since it also uses ultrasonic pulses in addition to the radiofrequency signals in order to track a person's position (Priyantha et al, 2000). To understand Cricket operation, it is necessary to understand what Time of Flight (TOF) means. TOF corresponds to the necessary time for a wave to travel from its transmitter to its receiver through a given medium (Marioli *et al.*, 1992). To achieve such, TOF between radiofrequency and ultra-sounds pulses is calculated, because by knowing TOF difference among the several beacons, it is possible to infer which one is the closest beacon to the current position.

In (Paul and Wan, 2008) the author states that, despite the increase of accuracy and stability of this solution over RF signals ones (like RADAR), Cricket still demands high maintenance levels and requires very significant efforts regarding its calibration.

From an indoor-oriented perspective, also Wi-Fi based technologies did progress, assuming today an important role in many location-based services used worldwide. Those solutions take advantage of pre-existing Wi-Fi network infrastructures, by making use of already implemented access points and radio equipments based on 802.11 protocol. In order to track the position of targets, these systems compare the calculated RSSI against a map of pre-existing radio signals.

Ekahau Real-time Location System (Ekahau, 2011a) is a location tracking solution which operates based on Wi-Fi signals, providing great accuracy levels: sub room, room, floor and building-level accuracy. At its current state, Ekahau RTLS is capable of tracking tens of thousands of mobile objects, assets or people in real time, with an accuracy rate of 1-3 meter, with short update intervals.

In order to work, Ekahau does not use traditional triangulation methods. Instead, it uses a method called Multi-Hypotheses tracking, which is constantly calculating multiple possible locations for the objects being tracked, assigning a score to each of them based on several factors. These factors include environment characteristics, the mobile device differences, the signal history and the movement models. The location estimate is then chosen based on the higher score (Ekahau, 2011b).

2.4. Signal Measuring Techniques

The performed process to obtain a device's location can be segmented into two distinct phases. Initially, it is necessary to measure some signal parameters related with the position of nodes involved in the conducted wireless communications. Further in the second phase, it is calculated the physical location of the target with the parameters evaluated at the first phase. Therefore, one can designate these two phases by signal measurement and position calculation, respectively. Below it is discussed the most common techniques used to implement the first phase.

- **Time Based**
    - **Time-of-Arrival (TOA):** in this technique, the calculation of the distance between the transmitting node and the receiving one is done through both the transmission delay and speed of signal. Today's top-notch techniques combine TOA with UWB (Ultra-Wideband Radio) technology in order to achieve greater accuracy (Gezici *et al.*, 2005). Such is due to time sensitivity at which UWB is subject, thus compensating the existing synchronization inefficiency in TOA, as the author explains in (Gezici, 2008). This kind of inefficiency is based on the need to set synchronized clocks on both transmitter and receiver nodes, something that consequently leads to difficulty regarding system installation and the underlying process of maintaining their accuracy during runtime.

    - **Time Difference-Of-Arrival (TDOA):** given the two distinct transmitted signal types using this method, the time difference between both of them is calculated and then it is used to "rebuild" the position of transmitter node (Gezici, 2008).

    - **Round Trip Time (RTT):** this is a method that arises with the goal of trying to solve the synchronization problem observed in TOA technique. Unlike the TDOA method (which uses two local clocks in each node involved), RTT does only use a single node to store transmission and reception times (Gezici, 2008).

- **Angle-Of-Arrival (AOA):** in this technique, nodes have the ability to measure the angle of arrival by the analysis of the obtained information, which will allow knowing the directionality of receiver node. Although the fact that no required synchronization time between nodes is seen as an advantage of this method, their limitations may be

an obstacle in certain indoor tracking scenarios. Such is due its high sensibility to multipath and Non-Line-of-Sight (NLOS) (Gezici, 2008).

- **Received Signal Strength Indicator (RSSI):** method that allows evaluating distance through the attenuation applied by the signal propagation, from transmitter node to the receiver one. This method states that the received signal strength is inversely proportional to the square of the distance (Gezici, 2008). Due to its low cost and availability, it is one of the most commonly implemented techniques (GU *et al.*, 2009).

By using the parameters obtained at the first phase through the use of the techniques mentioned before and by knowing reference nodes coordinates (i.e.: access points, sensors, etc.), tracking systems are able to calculate the physical location of a person or device. In this phase, it is usually applied trilateration and triangulation algorithms.

Table 1 discusses several location systems using the different signal measuring techniques, presenting also their advantages and disadvantages.

Table 1 – Location Systems Comparison (GU *et al.*, 2009)

Type	System	Coverage	Technique	Overview [A: Advantage / D: Disadvantage]
Radiofrequency (RFID)	WhereNet	2m to 3m	TDOA	A: Uniquely identifies a person and equipment. D: Requires several infrastructure components.
Infrared (IR)	Active Badge	Covers small areas	RSS	A: Ensures privacy. D: Low accuracy; long transmission periods; influenced by sunlight and fluorescent lights.
	Firefly	3.0mm	-	A: High accuracy; low measuring delay: 3 ms. D: Uses cables to connect tags; coverage area limited to 7m.
	OPTOTRAK	0.1mm a 0.5mm	-	A: High accuracy; able to measure certain movements in different parts of an object. D: The need for “line-of-sight”.
	IRIS_LPS	16cm per each 100m2	Triangulation	A: Extended coverage area. D: Influenced by sunlight and fluorescent lights.
Ultra-Wide Band (UWB)	Ubisense	Tens of centimeters	TDOA e AOA	A: Not restricted to "line of sight"; it covers an extended area; tri-dimensional positioning; high accuracy. D: Expensive system.
Ultrasounds (USID)	Sonitor	Covers small areas	-	A: Efficient power consumption. D: Low accuracy.
	Active Bat	3cm per each 1000m2	Multilateration	A: Extended coverage area; provides 3D positioning. D: Subject to reflections from obstacles; requires several transmitters in the ceiling of a room.
Wireless (WLAN)	RADAR	2.26m per each 312m2	Triangulation	A: Makes use of existing 802.11 infrastructures. D: Low accuracy; disregard privacy.

	EKAHAU	1m	RSSI	A: Low cost and good battery life. D: Low accuracy; provides bi-dimensional positioning only.
	COMPASS	1.65 per each 312m2	Fingerprint	A: It takes into consideration the impact of user orientation. D: Only a single user is considered.
USID + RFID	Cricket	10cm	TOA and Triangulation	A: Considers privacy aspects; low cost; decentralized administration. D: Consumes more energy.

2.5. Near Real-time Messaging Technologies

Near real-time (NRT) messaging protocols started being used in instant messaging services. Today, they empower a vast range of applications areas such as gaming, voice-over-IP, geo-location and cloud-computing, among many others (Barrett, 2009). This section approaches two open and widely used near real-time messaging protocols nowadays.

2.5.1. eXtensible Messaging and Presence Protocol (XMPP)

The eXtensible Messaging and Presence Protocol (XMPP) is an open near real-time communication protocol. Many applications like instant messaging, content syndication and lightweight middleware are based in this protocol (XMPP, 2011a). The XMPP, formerly known as Jabber, uses XML streams to exchange messages through TCP sockets. These XML streams are containers for exchanging XML elements between any two entities over a network. Inside the XML streams there are the XML stanzas which are discrete units of information.

The three most important stanzas defined in XMPP are the **<message/>**, **<presence/>** and **<iq/>** stanzas (Saint-Andre, 2005).

- **<message/>:** used by one entity to push information to another one.
- **<presence/>:** is mainly used for broadcasting information about an entity's network availability status. Most of the XMPP-based instant messaging applications use the presence feature.
- **<iq/>:** stands for info/query and is a request-response mechanism used in contact-list management, configuration, feature negotiation and in any other information structure than requires more complexity than regular messaging.

Most of XMPP implementations follow a client-server model although peer-to-peer implementations do exist. In XMPP, if two entities intend to communicate, they have to belong to each other's *roster*. *Roster* is the terminology used in this messaging protocol to define "contact list" (XMPP, 2004).

Each entity (which can be a person or a device such as a router or a terminal) has an associated address (or Jabber ID - *JID*) that is used to route the messages to the proper destination. Those addresses are composed by the structure that follows:

```
1. [ node "@" ] domain [ "/" resource ]
```

While the node and the domain are needed to be specified, the resource is optional. The resource is used for situations in which a user logs into a XMPP server multiple times in parallel, from different XMPP clients. Thus, the resource is commonly used to denote a location for each of those XMPP parallel sessions, allowing messages to be received at all the different locations or at just one in particular (Tim Jones, 2009).

Even if the receiver entity of a message is not available, most of XMPP implementations store the messages for later delivery, once it becomes available (Saint-Andre, 2005).

Figure 3 describes the XMPP Architecture. It also includes XMPP gateways which have the purpose of translating between foreign messaging domains and protocols.

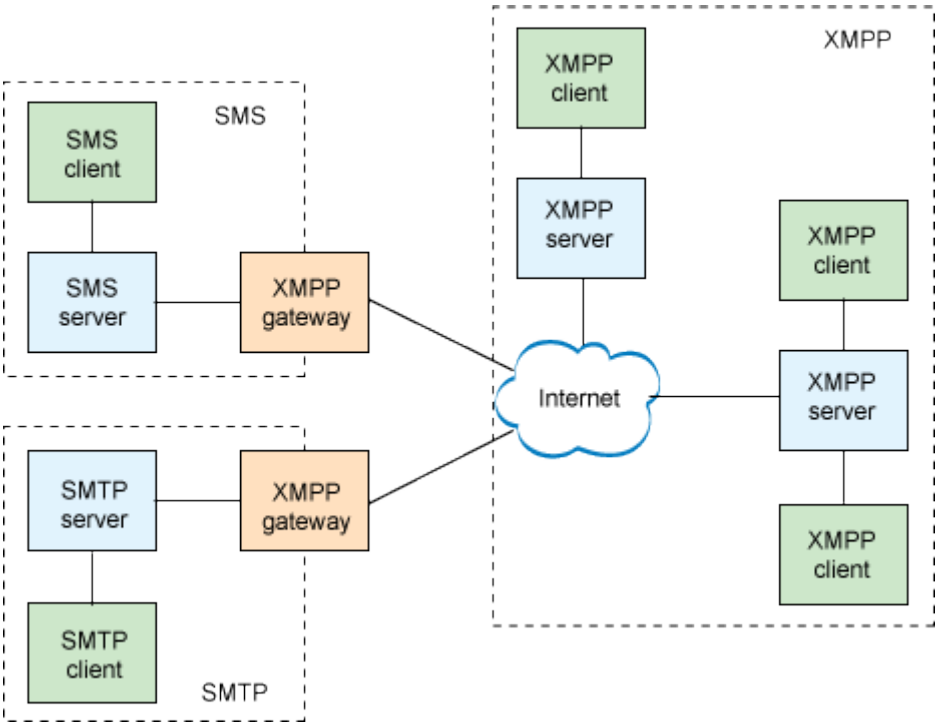

Figure 3 – XMPP Architecture Including XMPP Gateways (Tim Jones, 2009)

In Figure 3, there are depicted XMPP gateways to a Short Message Service (SMS) domain and a SMTP domain. This means that XMPP is also a protocol able to provide universal connectivity among different endpoint protocols.

As of September 2011, Microsoft's Windows Live APIs supports the XMPP protocol, thus allowing third party applications to interact with most of the company's XMPP supported services like the Windows Live Messenger (XMPP, 2011b). Prior to Microsoft, also Skype had already included XMPP support (XMPP, 2011c).

2.5.2. Advanced Message Queuing Protocol (AMQP)

Similar to XMPP, the Advanced Message Queuing Protocol (AMQP) is a general messaging purpose protocol. Originally designed to provide interoperability across multiple vendors, the

AMQP is today an open standard for passing business messages between applications and organizations (AMQP, 2011).

Although it may seem identical to XMPP, there are some important features that distinguish them. One of them lies in the format of the exchanged messages. Whereas the XMPP uses a XML format, the AMQP uses a binary format. The presence feature is also an important functionality that AMQP lacks where XMPP does not. The presence mechanism allows knowing the availability of both parties in a message exchange. For instance, through a presence supporting protocol it is possible for a user to know, if the other entity he wants to send a message to, is connected to the service or not.

On the other hand, AMQP supports many use cases (at least once, exactly once, select subscribers, persistence, etc.) allowing to control the way messages are routed (to where and how) (Grigorik, 2009). Figure 4 depicts the overall AMQP architecture.

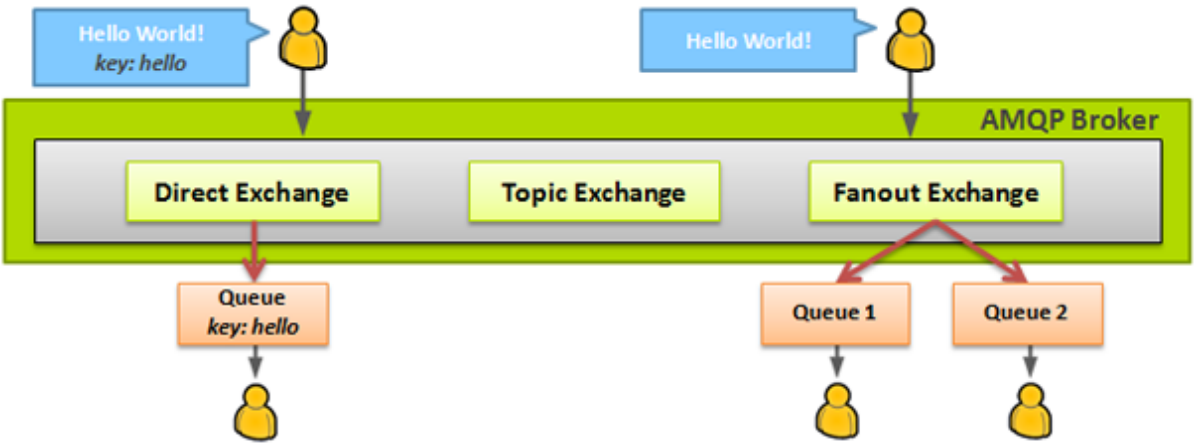

Figure 4 – AMQP Architecture (Grigorik, 2009)

In the AMQP architecture there are 4 components. They are the *Publisher*, the *Exchange*, the *Queue* and finally the *Consumer*. The *Publisher* is the one who produces data and pushes it to an *Exchange* and the *Consumer* is the one who receives the *Publisher's* produced data.

The AMQP *Brokers* are servers to which AMQ clients connect using the AMQ protocol. Within an AMQP *Broker* there are the *Exchanges* and *Message Queues*. While the *Message Queues* are self-explanatory since they are used to store messages, the *Exchanges* are not so explicit.

Exchanges are routing engines responsible for delivering the messages to the appropriate queues. It is also important to refer that the exchanges do never store any messages.

Three kinds of exchanges are illustrated in Figure 4. The *Direct Exchange* routes messages to just a single queue. On the opposite, the *Fanout Exchange* routes messages to every queue. At last, the *Topic Exchange* routes messages based on a certain key (Grigorik, 2009).

2.6. Mobile Applications Distribution

As of July 10 of 2008, mobile developers weren't making too much money from their applications. The existing digital distribution platforms prior to that date were not operating system-native platforms forcing the user to know other third-party platforms in order to buy and install applications. From that date on, everything changed with the launch of Apple's digital distribution system, App Store (Apple Inc., 2008b). It was a system providing both free and paid applications with organization and convenience for consumers while at the same time, offering new applications great visibility and attractive revenues to the developers (30% to Apple and 70% to the seller of the application) (Venture Beat, 2008).

Following Apple's success path, which reached 10 million application downloads in the first week right after the launch, Google launched Android Market just three months later, and in December of that year even Palm launched its own software store which, however, didn't prevailed for long and was later replaced (Apple Inc., 2008a).

Today, almost every mobile operating system has its own native distribution platforms including Nokia/Symbian (Ovi Store), RIM/Blackberry (App World), Palm/WebOS (App Catalog) and Windows Phone (Windows Phone Market Place), which was the last one to enter in the competition. In early 2011, a partnership announcement between Nokia and Microsoft to include Windows Phone operating system on Nokia's smartphones was made (Nokia, 2011), pushing Symbian OS to its end and leaving the three major competitors with their own place at the podium: Apple's iOS, Google's Android and Microsoft's Windows Phone.

According to the Lookout Mobile Security Report of February 2011 (Lookout, 2011), the number of Android applications emerging in the Android Market keeps growing at a faster pace than the ones appearing at the App Store. Despite these values, the iOS platform still attracts more developers. Figure 5 depicts the growth of the number of developers for both platforms between September 2010 and February 2011.

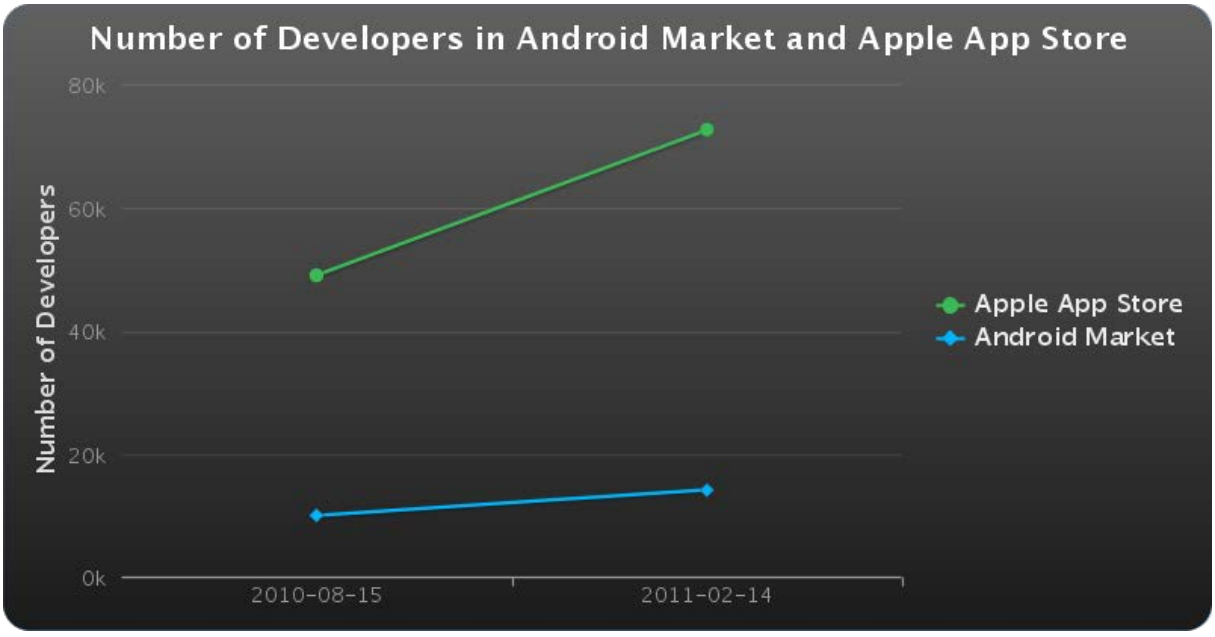

Figure 5 – Number of Developers in Android Market and App Store (Lookout, 2011)

Regarding business models' results between each platform's applications, the iOS stills having more paid ones than the Android has. However, the Lookout's report states that the number of Android paid applications has increased 12 per cent between August 2010 and February 2011, whereas the number of iOS paid ones has decreased by a 5 per cent quote.

In what concerns the implemented advertising solutions in free applications, the AdMob solution by Google still being the one which is most adopted in both Android and iOS platforms, when compared to the Apple's iAd advertising solution. iAd is only available for the iOS applications though. Figure 6 illustrates the comparison between the percentage of free and paid applications for both the platforms.

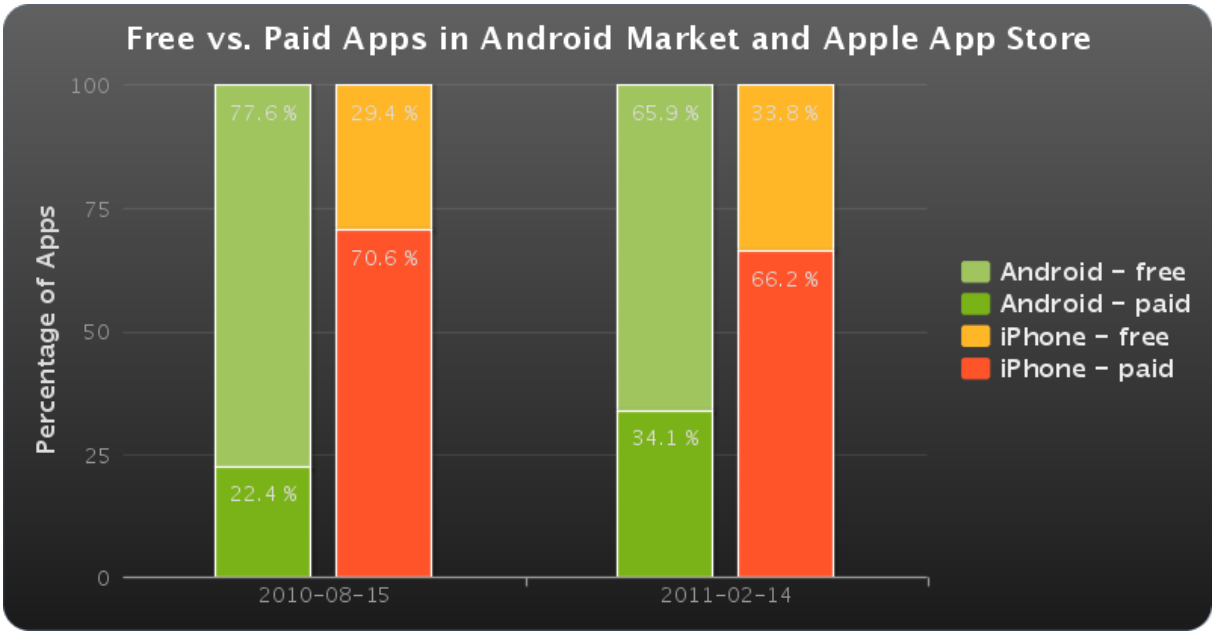

Figure 6 – Free vs. Paid Apps in Android Market and App Store (Lookout, 2011)

According to another report from Millennial Media (Millennial Media, 2011), the iOS ad impressions grew 60 per cent year-over-year but stills not being enough to beat the current mobile platform leader, the Android, with 56 per cent of the smartphone device impressions, against the 28 per cent share owned by iOS (as of October 2011).

It also notes that over the past year, Android took the mobile platform leadership from iOS mainly due to its openness to many device manufacturers. Millennial Media analysts agree that the rapid consumer adoption of Android platform is related to the fact of having manufacturers creating multi-priced devices for this platform, thus reaching a wider audience. The Chart B in Figure 7 illustrates the advertisement share for the most important mobile platforms.

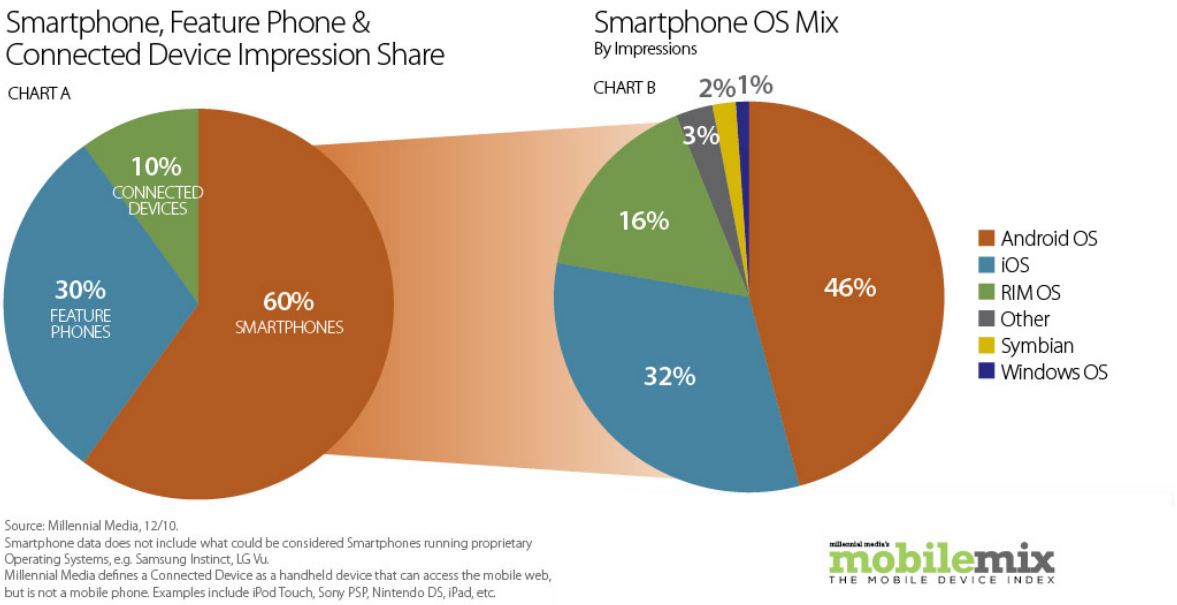

Figure 7 – Smart vs. Connected Device Impression Share (Millennial Media, 2011)

Concerning the values regarding applications that access users' location, the App Store still has a greater quantity than the Android Market does, by a percent rate of 34% against 28%, respectively. Other personal info like contacts' information stills being accessed in more iOS applications that in Android ones according to the Lookout's report.

According to ABI Research (ABI Research, 2011), the Android platform has surpass the iOS regarding the number downloaded applications in the second quarter of 2011. They state that 44% of all the downloaded applications were done by Android mobile devices while the iPhone ones could not reach more than 31 per cent. However, it must be noted that these values were influenced by the increase in the sales of mobile devices when compared to the first quarter of the same year.

Despite all these facts, the ABI Research declares that the iOS platform still surpassing Android regarding the number of applications downloaded per user. They state that two iOS applications are downloaded per each Android downloaded one. Another factor that has to be taken into account is the reported superior quality of the iOS applications. This is due to the Apple's higher controlled environment and acceptance criteria for its applications when compared to the Android ones.

CHAPTER 3

MOBILE OPERATING SYSTEMS

Over the last few years, smartphones have invaded our daily lives and later, tablets have achieved a good position in the market. Their evolution has redefined the way we communicate and access information. From one side we have all the new hardware capabilities that were introduced over the time which contributed to this overall massive spread of such devices. On the other hand, the supporting software was the revolutionary factor in their evolution, particularly their operating systems. This chapter provides an essential overview about the most relevant mobile operating systems available on the market and highlights the Apple's iOS architecture.

3.1. Mobile Operating Systems Categories

We may divide mobile operating systems in three distinct categories:

- Manufacturer-built proprietary operating systems
- Third-party property operating systems
- Free & open-source operating systems

The manufacturer-built proprietary operating systems category refers to those manufacturers who develop their own operating systems to their own devices. Products of well-known mobile brands like Apple, RIM and HP are good examples of such manufacturers. These operating systems are known to make the difference among all the others as they have a very consistent look and feel across all the devices they are installed into.

The third-party property operating systems are proprietary systems that are developed by companies that do not manufacture any devices. Since they develop for devices produced by other manufacturers, they license their operating systems for running it on those manufacturers' devices. The best examples we can think of are Windows Mobile and Windows Phone, both developed by Microsoft.

While Windows Mobile was already discontinued, Windows Phone continues to gain even more ground, especially after the partnership announcement between Nokia and Microsoft (Nokia, 2011), making it the replacement for Symbian OS, the operating system until then implemented in the majority of Nokia devices. Usually, these operating systems provide a consistent look and feel but they tend to come stocked with additional manufacturer's software, which may have a positive or negative impact on these characteristics.

At last, the free & open-source operating systems category concerns to those open-source systems built by a company, a group of companies or a community of developers. These systems are made available for everyone to modify them in any way they choose, and to install them on their choice of devices.

3.2. Application Development Support

Although we have at our disposal many platforms for mobile development, the developers tend to opt for the two operating systems which are mostly widespread: the Apple's iOS and Android. Both the mobile platforms carry some features that may be pleasant for the developers by turning their job easier. However, the opposite can also happen.

When choosing a platform to develop to, developers have to evaluate some factors like the programming language learning curve, the availability or price of the development tools and the need for subscription (which may be paid) in order to be able to submit applications to the underlying online store. Table 2 discusses both the advantages and drawbacks of developing for iOS or Android.

Table 2 – iOS vs. Android Developer Features Comparison

	iOS	Android
Programming Language	Objective-C; As it is an exclusive language for Apple's platforms it may be an obstacle for developers who are not familiarized with other languages; The learning curve may be slow due to concepts inexistent in other common languages.	Java; Common programming language used by many developers and for many platforms.
App Development Platform	Restrictive set of developing tools and developer guidelines; Limits developer's creative skills to the frameworks they offer.	Open Development Platforms; Third-party tools; Allow developers to play with features, adding more functionality to them.
Hardware Diversity	The development is limited to Apple's devices holding iOS as operating system. On the other hand, developed applications may work on both iPhone, iPod and iPad (through emulation if it is not iPad native) and may also be distributed as a single package on App Store to run on iPhone and iPad with native features for both devices, avoiding the need of having an application on sale per each device.	Very large range of devices supporting Android OS. The hardware fragmentation makes the whole developing process very confusing for the coder (different processing power, screen resolutions, etc.); The applications' performance may differ from device to device, which may become a big obstacle on games development.
Multitasking Abilities	Stable, exclusive and easy to use platform; Tools clearly specified with their potential and boundaries well defined.	Long learning curve; Highly fragmented platform, making it a big challenge for inexperienced developers.
Mobile App Testing	Apple's development tools used to be a little bit fragmented, as the interface builder was not fully integrated with the main programming tool. With the emergence of version 4 of XCode, all the tools have become fully integrated with each other, providing a more pleasant developing. Testing may be made through a software simulator or a real device, where the first one is very limited, as it does not possesses many of the capabilities the real device does (GPS, Bluetooth, Accelerometer, etc.). Testing is easier when it comes to the need of evaluate	Offers excellent testing environment; All testing tools are neatly indexed and the IDE offers a good model of the source code. There is a large range of hardware where the application can be executed, so it is difficult to estimate the behavior of it on all those devices.

	real-world performance as there are a few set of devices where the app can be run.	
App Approval	App Store may take from days to weeks for app approval, but the average time is usually about 4 days. The applications' revision is very rigorous helping to maintain the quality and security of the new applications that arrive to the store every day. Apple's App Store is the only place where developers can submit their applications.	There is no approval delay; It usually takes only 10 minutes to get the app featured in the Android Market. The quality of the applications is not taken into account, which may lead to a lot of useless applications and malware. Developers are not stuck with Android Marketplace and may submit their applications to other different Android Stores if they wish so.
Payment Procedure	Developers earn 70% of the revenue generated from the sales of their app in App Store.	Developers earn 70% of the revenue generated from the sales of their app in Android Market.
Development Subscription Fee	Developers need to pay $99 as annual fee in order to gain access to the iPhone SDK.	Developers only need to pay $25 for a life-time registration.

3.3. Business Models

Every single developer has a goal in mind when developing an application. Whether it is to make money or just for fun, the developer has to choose the business model that best suits his needs. Among all the options, he may opt for distributing his application for free or at some cost for the end-user. Many coders tend to create two versions of the application, a full add-free version and a light version (which may contain advertisements or lack some features).

With the introduction of In-App Purchases this year, a new business model has been made possible by allowing the end-user to individually purchase digital content/services he likes instead of paying for the full application. Sections 3.3.1 and 3.3.2 approach, in greater detail, the basic concepts of these two mobile software business models.

3.3.1. In-App Advertisements

There are several companies that provide advertising and monetization solutions for mobile developers. Among the big ones are AdMob (AdMob, 2011) and the iAd Network (Apple Inc., 2011a). AdMob is owned by Google and serves ads on the iPhone, Android and WebOS platforms. The iAd Network is from Apple and serves ads exclusively on the iPhone. While the first may be used in many different platforms, the iAd Network is only available for iOS applications. In the iAd Network, Apple sells the advertisements and serves them to the applications, while the developer receives 60 per cent of the ad generated revenue (Apple Inc., 2011b).

The big mobile software industries are the ones to first state that in-app ads are one of the greatest bets a developer may take. A good example to be considered is the case of the well-succeeded multiplatform game Angry Birds, published by Rovio (Rovio, 2011). The game publisher, states that by the end of 2011, it will be making $1 million each month, solely from in-app ads incomes, on the Android platform alone. As we may recall, it is a

multiplatform game, so we may add to this value the additional revenues coming from the game distributed on the other platforms (O'Dell, 2011).

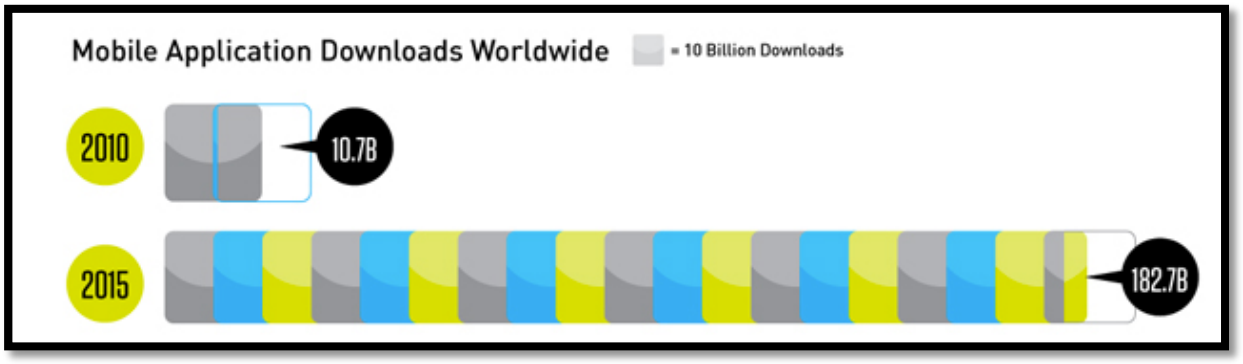

Figure 8 – Mobile Application Downloads Worldwide (BuySellAds, 2011)

Figure 8 indicates the number of downloaded applications in 2010 as well as an estimate of what number will it possibly assume in 2015, according to a research report from IDC (IDC, 2011). At the time of this writing, in-app ads represent 5 per cent of total mobile ad spending, which is an incredible value if we consider the time that has passed since the first implementation of this business model. Once again, mobile market research companies expect In-App revenue to surpass $860 Million by 2014 (Figure 9).

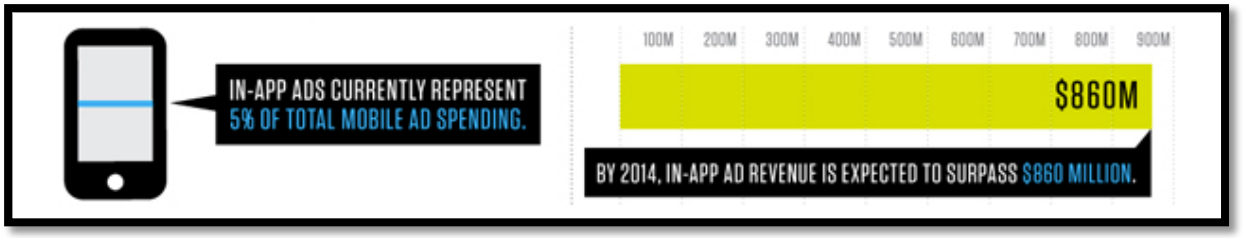

Figure 9 – In-App Ad Spend (BuySellAds, 2011)

In spite of the fact that the majority of the users may dislike in-app advertising, the truth is that a great portion of them are able to recall those ads at a higher rate, than those ads we see when browsing the web, through the mobile device (Figure 10).

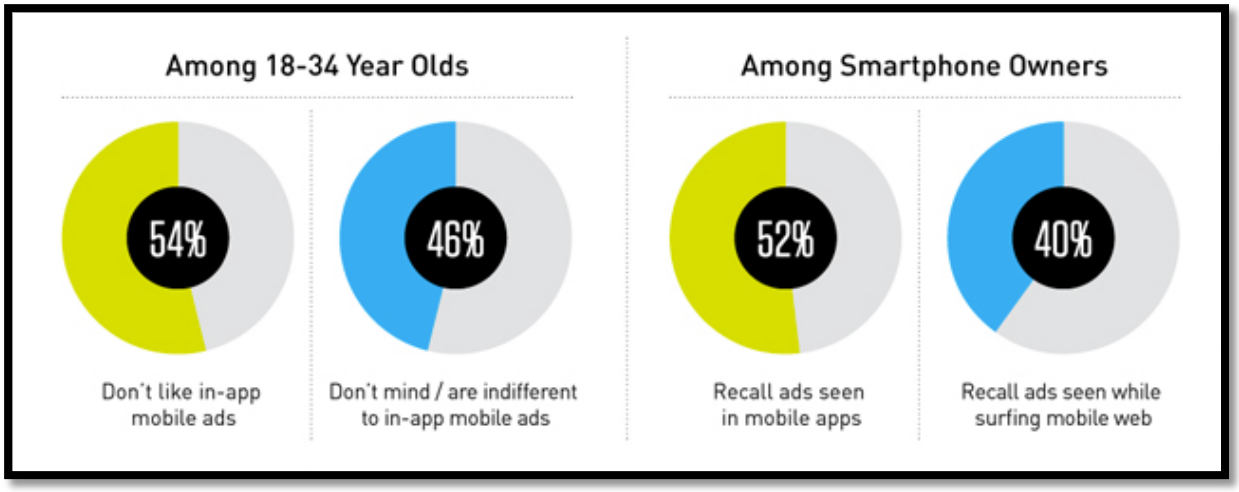

Figure 10 – How Consumers Feel About In-App Mobile Advertisements (BuySellAds, 2011)

The leading sector of in-app advertising is definitely the mobile games area. Many developers are taking the steps that some giants of mobile gaming industry already took by distributing full versions of the games for free, thus generating the income directly from in-game ads. As mentioned earlier, Angry Birds is a game where its publisher applied this approach successfully with surprising results (Figure 11).

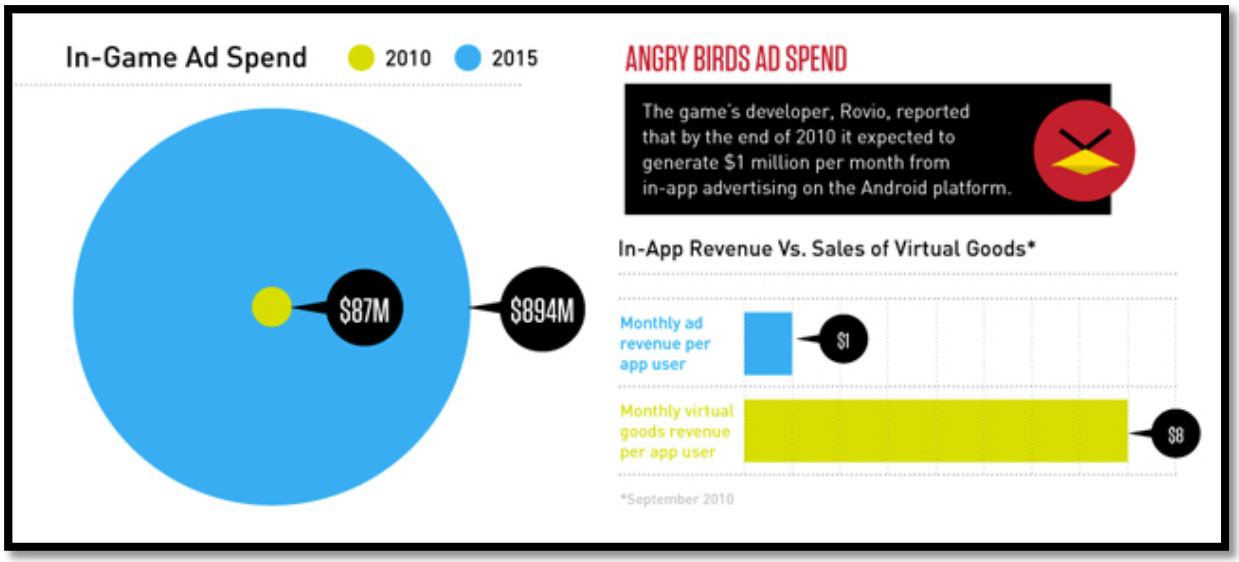

Figure 11 – In-Game Ad Spend (BuySellAds, 2011)

Figure 12 approaches a comparison between the use of different business models applied to mobile applications by 2010 and 2011. As we can see, the percentage change in just one year for in-app advertising and in-app purchases is considerably high. This fact strongly indicates that in a near future, developers will prefer those two business models over the old application purchase one, in order to make money from their applications.

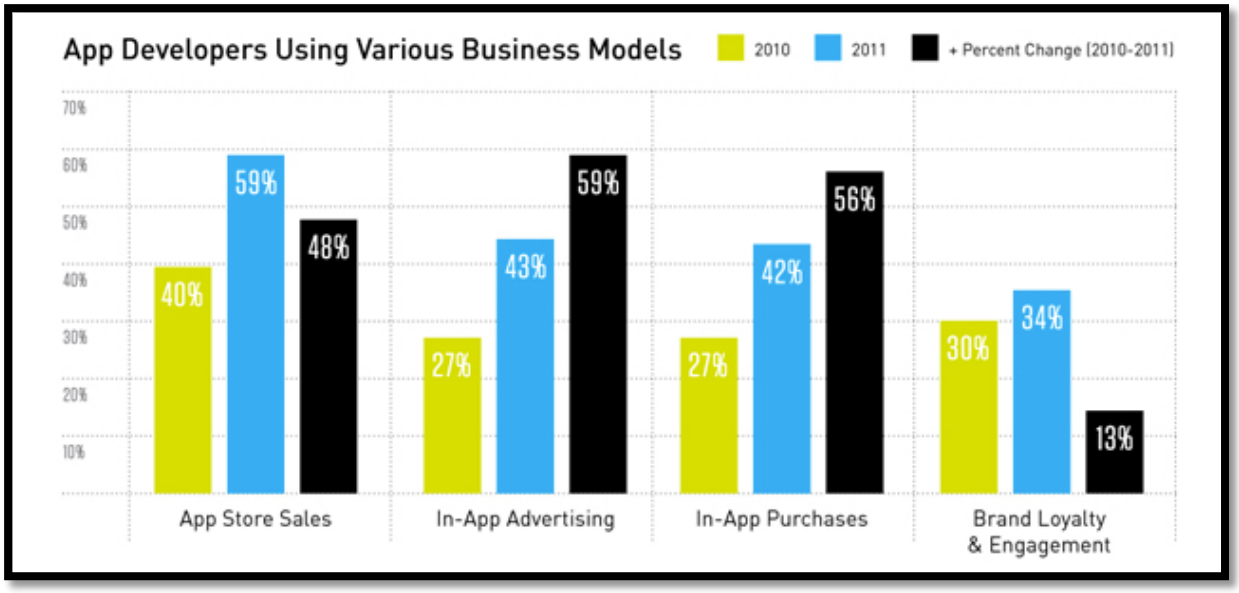

Figure 12 – App Developers Using Various Business Models (BuySellAds, 2011)

3.3.2. In-App Purchases

In-app purchases, first introduced on iPhone and later on Android, are seen as an alternative to in-app advertisements. From this model, many other business models can derive from, giving the developer many other possibilities to make more profits from their products.

Let's take as an example a mobile application that allows the user to create electronic songs. Now let's imagine the application itself comes with a few electronic sounds to compose our songs and it is distributed in a full-purchase fashion. This means the user buy the full package and then, if the developer decides to introduce new songs, they can only be obtained by the end-user through free updates. This situation does not leave any room for the developer to extend its profits from the freshly added content, leading to the lack of motivation of the developer regarding the delivery of new content/features, in order to enhance the application.

From the user perspective, this model has its drawbacks too as he is forced to buy the full package instead of him being free to buy only the content he is really interested in. Let's now approach a possibility on how in-app purchases model can be applied to overcome those mentioned limitations. The developer releases the core application for free to any user. In order to make the use of the application more attractive, the developer makes available some free content so that the user can experience some of its potential. Then, the developer makes available packs of new sounds that can be bought by the user at any time.

From a single user, the profits can now be extended depending on the cadence of new released content as opposite to be confined strictly to the cost of the application. With this model developers may, if they want, focus only on the delivered content as source of revenue, thus not limiting their selves to the programming area (i.e.: multimedia content). Developers can even delegate this job to any person/company who has nothing to do with programming and wants to make profit from the content they create. They may also want to offer some app services for limited time through paid subscriptions, which is entirely possible when relying on this model.

Figure 13 – In-App Purchase on iPhone (Neowneow, 2011)

Mobile games are perhaps the kind of applications that makes more use from in-app purchases (Figure 13). In order to offer a whole brand new gaming experience to the user, the developer does not need to release a new application. Instead, he can release new levels, characters or whatever which can prolong the value of the game, so that users can easily buy right inside of it. This kind of extra content may be added over time, being a good reason to make users coming back to the game.

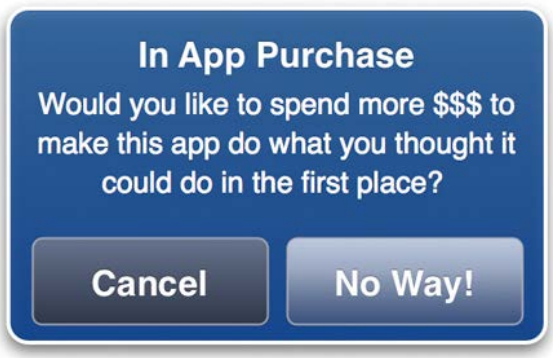

Figure 14 – In-App Purchase (TheAppleLounge, 2011)

Not every single aspect of in-app purchases is seen as being beneficial to the consumer. The reason of this statement lies in the fact that, many developers and companies, take advantage of this business model to make consumers believe that they are acquiring something capable of performing what the ones who are selling it claim it does when it does not (Figure 14). In fact the application may be capable of doing what the consumer expects, but he will need to pay more for it.

For instance, let's take a look at the example given before, a mobile application that allows the user to create electronic songs. The developer states in the application description that the application is able to add fade-in and fade-out effects to a song track. Instead of selling the application for free (which would give the consumer the opportunity to freely test the application) the developer acts wisely by selling the application at an underpriced value, for example $0.79. As the consumer feels that the application really does what he wants it to do, and for a good price, he buys it. However, when he launches the application and notices that in order to use the effects he wants to there's a catch: he needs to buy those effects inside the application. It can be a very frustrating experience but the fact is that, since the launch of the in-app purchases model, this kind of situations just grew up exponentially.

3.4. Apple's iOS Overview

Also known as iPhone OS prior to June 2010, iOS is the mobile operating system developed by Apple and is the one that is currently deployed on all the company's "iDevices" such as the iPhone (which was the original reason of the iOS development), iPad and iPod Touch.

Revealed to the world in 2007 at the first iPhone's launch, iOS was immediately a case of success, despite all the limitations around it and which other operating systems already had overtaken at that time. Simple features like multitasking and Bluetooth pairing were ignored. So one may ask: where did all this success came from? The answer is actually pretty simple. The quality of the equipment, especially of the screen and its great response to the user interaction was indisputable. However, what really made the difference was the Apple's bet on a user-friendly and innovative interface, where there was no more need for stylus in order to interact with the device but our own fingers.

Before going into detail about the iOS architecture, section 3.4.1 discusses the iPhone architecture in order to better understand where does the operating system layer, is placed in.

3.4.1. iPhone Architecture Overview

Figure 15 describes all the layers that compose the whole mobile device.

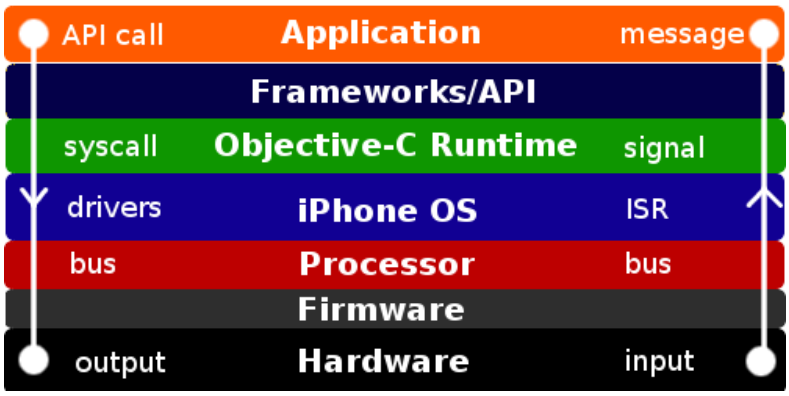

Figure 15 – iPhone Architecture's Diagram (TheCoffeeDesk, 2009)

By exploring the figure above, the first thing one may probably notice, is the fact that we have a hardware layer and a processor layer. This can be explained by the fact that, whereas the hardware layer refers to the chips of the device's peripherals (accelerometer, gyros and display), the processor layer corresponds to the ARM CPU. For a better understanding of each layer that comprises the iPhone architecture, follows a summarized description for each of them (TheCoffeeDesk, 2009):

- **Application Layer:** This is the highest level layer, thus referring to the user bought applications from App Store. For security reasons, all these applications runs on a highly controlled environment, often called "user space".

- **Frameworks/API Layer:** This layer contains API calls such as Cocoa Touch and upper-level OpenGL ones, which lies on top of Objective-C runtime as seen on Figure 15, due to the fact that many of them are written in this programming language.

- **Objective-C Runtime Layer:** Objective-C dynamically-linked runtime libraries and underlying C libraries make up this layer.

- **iPhone OS Layer:** Here fits the core part of iPhone operating system, that is, its kernel, drivers and services. This layer sits right between the "user space" and hardware.

- **Processor Layer:** This layer refers to the ARM CPU and the interrupt descriptor table as set up by the iOS, during boot and driver initialization.

- **Firmware Layer:** This layer refers to the chip-specific code that is either contained with memory in/around the peripheral itself or within the driver for that peripheral. Due to the fact that many people refer to the "firmware" as the entire OS, one must not confuse what is being described here, as that is not what this specific layer is referring to.

- **Hardware Layer:** Refers to the physical chips soldered to the iPhone's electrical structure. It is important to note that, despite of the fact that the physical processor chip fits on this layer, its instruction set and in-memory descriptor tables are contained in the processor layer and not in this one, as described above.

Figure 16 describe all the steps taken in order to process a hardware action (in this case, movement detection by the accelerometers), that goes from the bottom layer (referred to as the hardware layer) through all the described layers, up to the application layer.

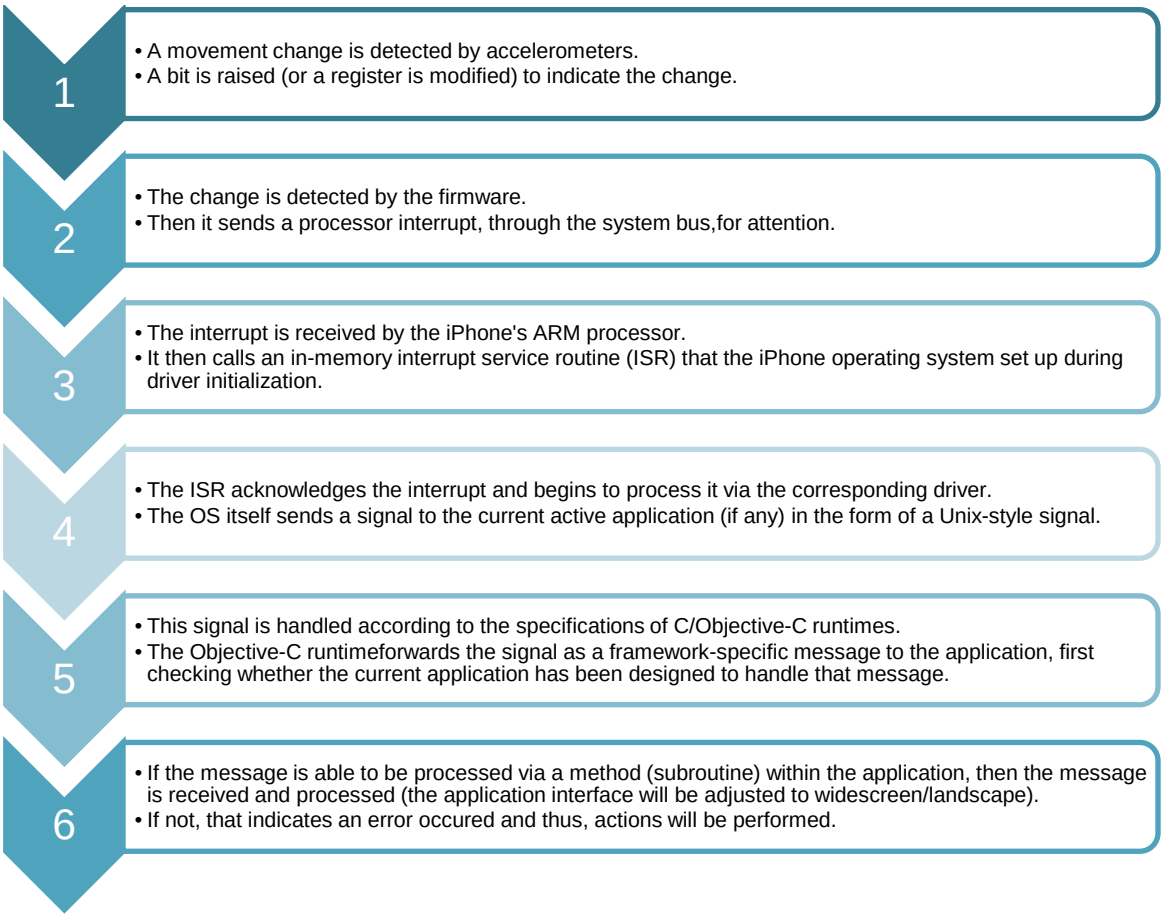

Figure 16 – Software Processed Hardware Action Overview (TheCoffeeDesk, 2009)

The iPhone is unique in the sense that the programmer knows it is almost guaranteed that his application can be executed on all devices that support the iOS. This is due to its low number of models and hardware homogeneity. As the time has passed, the newer models have integrated new sensors which brought a new breeze of fresh air to the developers, opening new doors to their creativity.

Some hardware features such as magnetometer, gyroscope and GPS were not included since the first model of iPhone. Since the iOS applications are capable of running on iPod Touch and iPad models, it is important to note that these devices also have their limitations regarding hardware, which are in fact bigger when compared to iPhone ones. Figure 17 describes the hardware support for every Apple “iDevice” as of the time of this writing.

Hardware Feature	iPhone				iPod touch				iPad		iPad 2	
	Original	3G	3GS	4	1st Gen	2nd Gen	3rd Gen	4th Gen	WiFi	3G	WiFi	3G
Cellular	☑	☑	☑	☑	☐	☐	☐	☐	☐	☑	☐	☑
WiFi	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
Bluetooth	☑	☑	☑	☑	☐	☑	☑	☑	☑	☑	☑	☑
Speaker	☑	☑	☑	☑	☐	☑	☑	☑	☑	☑	☑	☑
Audio In	☑	☑	☑	☑	☐	☑	☑	☑	☑	☑	☑	☑
Accelerometer	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
Magnetometer	☐	☐	☑	☑	☐	☐	☐	☐	☑	☑	☑	☑
Gyroscope	☐	☐	☐	☑	☐	☐	☐	☑	☐	☐	☑	☑
GPS	☐	☑	☑	☑	☐	☐	☐	☐	☐	☑	☐	☑
Proximity Sensor	☑	☑	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐
Camera	☑	☑	☑	☑	☐	☐	☐	☑	☐	☐	☑	☑
Video	☐	☐	☑	☑	☐	☐	☐	☑	☐	☐	☑	☑
Vibration	☑	☑	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐

Figure 17 – Hardware Support in Various iDevices (Allan, 2011)

The iOS developers may allow their applications to work in a large range of devices, although making use of limited hardware resources. This is possible since there are API calls in order to know hardware availability for a certain device, thus being possible to adjust the application behaviour.

3.4.2. The iOS Layer Set

Being derived from Mac OS X, iOS is a Unix-like operating system which is composed by four abstraction layers, as shown in Figure 18. The basic services and technologies on which all the applications rely in order to function, are at the lower layers; as opposite, the more sophisticated services and technologies are at the top layers.

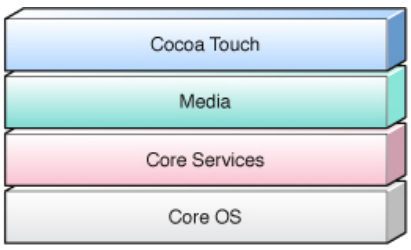

Figure 18 – iOS Layers (Apple Inc., 2010a)

For a better understanding, each one of the four layers can be described as follows:

- **Cocoa Touch Layer:** This layer should be the first to be examined whenever a developer is designing an application. This is due to the fact it contains the key frameworks for building applications, such as the ones supporting multitasking, touch-based input, push notifications, and many high-level system services.

- **Media Layer:** All the graphics, audio and video technologies are contained in this layer. Mobile games are probably the developing sector that makes more use of this framework.

- **Core Services Layer:** This layer contains the fundamental system services that all applications make use of. It is very likely that regular developers will not use them directly but the truth is that many parts of the system are built on top of them.

- **Core OS Layer:** All the low-level features that most other technologies are built upon are present at this layer. The need for deal with security or even communicating with external hardware accessories, may force a developer to use the frameworks contained in the Core OS layer.

Figure 19 describes the main features and services which are attached to every single layer of the iOS architecture.

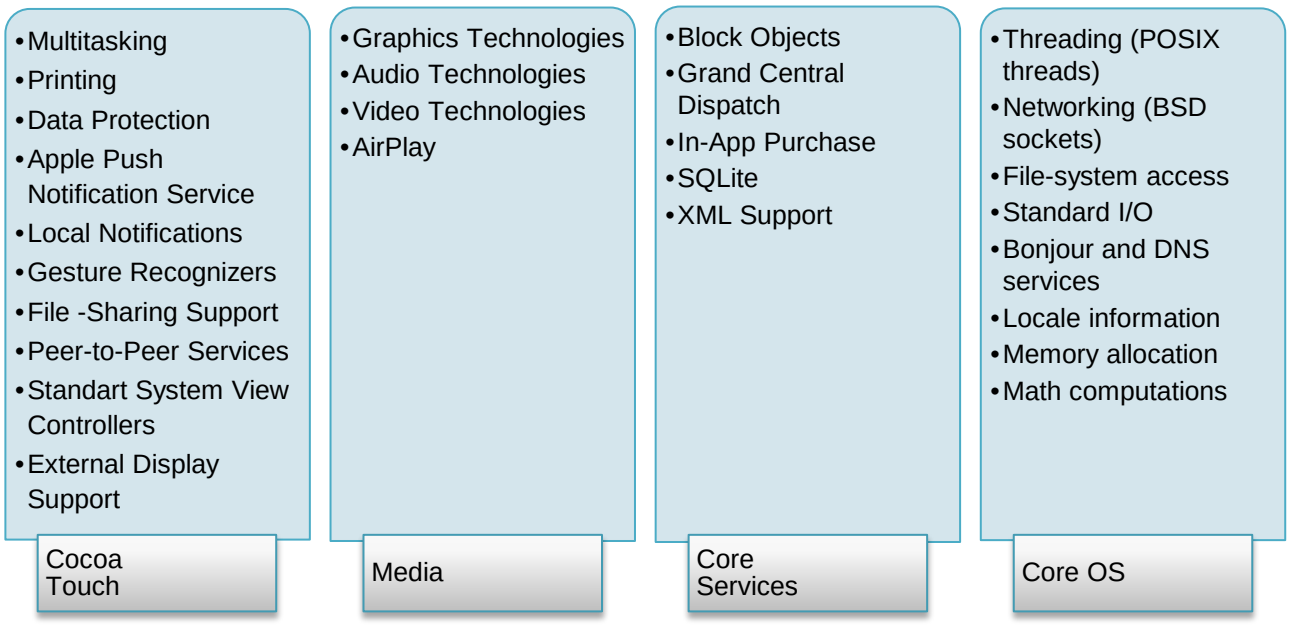

Figure 19 – iOS Architecture Layer's Features and Services

CHAPTER 4

APPLICATION ARCHITECTURE

This section presents the overall application architecture and its underlying communication flow. As it happens in most cases, when choosing a technology over another we submit ourselves to both the advantages and disadvantages that it will bring to our project. Therefore this section justifies every decision made when choosing the most appropriate technologies for the application. It also describes some solutions in order to overcome energy use and bandwidth consumption limitations which mobile devices have.

4.1. Overall Requirements

The application developed, entitled *EyeGotcha* (coming from the *"I Got Ya"* slang expression), is intended to be used as a location monitoring tool for other mobile devices. Both the monitoring device and the monitored one run the same application, so that there is the possibility of reverting roles. Thereby, the application should work as a universal distributable package.

The main target of development is the iPhone's iOS, but the application architecture should be engineered in a way that other mobile platforms, like Android and Windows Phone, can latter join the network by having this same application ported to their devices.

In order to understand then the remaining content of this chapter, the following nomenclature should be perceived:

- **Monitoring Device:** mobile device which is monitoring other mobile devices running the *EyeGotcha* application.
- **Monitored Device:** mobile device which is being monitored by any other mobile devices running the *EyeGotcha* application and which have permission to do so.
- **Alert:** a rule targeted at a certain monitored user and based on a set of parameters. It is configured in order to the monitored user's mobile device send alert notifications about predefined events that may occur (i.e.: reaching or leaving a certain location).
- **Alert Notification:** notification sent from the monitored device to the monitoring device, whenever an event that meets a certain configured rule (alert) occurs.
- **Authentication:** required step where users need to log into a unified system (being uniquely identified) in order to be able to communicate with each other.
- **Pairing:** process that establishes an agreement between two parties in order to both be able to send and receive data from each other.
- **Presence:** mechanism which allows a user to be informed about the current availability ("Available", "Away", "Offline", etc.) of other users he is paired with; system similar to the one used in regular instant messaging applications (Saint-Andre, 2005).

The application should be able to ask another user for location tracking data and receive it. The received information has to be presented on a consolidated basis (through maps and other graphical components), so that the common user perceives the full representation of the whole received data set.

Alerts should be easy to set for the monitored users and should be triggered by the application, running in their mobile devices, as soon as their settings meet the event which is taking place.

Whether a monitored device is reaching or leaving a certain location, within a certain predefined schedule, alert notifications should be sent to the monitoring devices without any physical interaction of the monitored user.

The alert notifications as well as the automated responses to location tracking data requests, act without any user interaction. Users should be sure of who they are sharing information with and what type of information they are sending as well. Therefore, the application should provide a privacy component, offering options to control who is able to ask for data, and what kind of data a single user is able to ask for.

Sending and receiving data between mobile devices requires the development of a communication architecture that addresses three essential requirements: authentication, presence, and messaging.

Authentication is needed in order for a user to be identified among several other ones, in the system. It should be as automated (preferably without manual account registering) and as secure as possible. Unsecure authentication systems could lead to the sending of forged messages by malicious people, since it could allow to send fake tracking requests or to bypass user's privacy settings by disguising themselves as another legit person. Users also should not be submitted to the insecurity of not knowing who is gathering their information. They need to feel privacy control over every single aspect of the application to keep motivated in using/buying the application.

The iPhone has exclusive hardware identifiers that could allow the application to identify the equipment where it is running on, instead of the user itself. This system could bring advantages related to performance and redundancy, but could bring some drawbacks in several other aspects, making it impractical to implement. Therefore, some implementation possibilities for the authentication system are explored and discussed in section 4.3.

Presence support is also an important requirement as a user should expect to know whether the users he is paired to, are available to interact with or not. The messaging system should also be flexible and interoperable in order to be easy to implement in other platforms. Thus, a solution based on standardized protocols should be more valuable. In addition to all the explicit advantages, that would extend the application possibilities and usefulness by communicating with other kind of electronic devices (using the same standardized protocols) than mobile phones.

A starting from scratch solution, able to fulfill all these requirements takes too much time and effort to implement. In addition to that, it would not correspond to the quality level of available standardized communication protocols which stand out for its many features, interoperability and proven security. Protocols like the Extensible Messaging and Presence Protocol (XMPP) and Advanced Message Queuing Protocol (AMQP) are good communication protocols that can neatly serve these purposes. These protocols and the overall communication architecture of the application are discussed in section 4.2.

Bandwidth constraints are another factor that affects mobile devices. Mobile operator's data plans are still quite expensive and have very strict traffic limits. It is important to implement bandwidth saving solutions that minimize the traffic consumption.

Users also need a system that allows them to add new devices to be monitored, in their application, or to remove them at wish as well. Therefore, users need a pairing system. Given the hardware the iPhone has, there are many scenarios that can be covered so that the user has a bigger range of pairing options, regarding convenience and communication failure situations. Solutions pass through the use of technologies such as EDGE/3G, Wi-Fi, Bluetooth and SMS, as a way to provide long and short range connection between devices and also to offer online and offline pairing options. These solutions are approached in section 4.4.

The application send alert notifications to monitoring devices, which may be triggered by a set of configurations related with location, time and user behavior factors. The constant monitoring of a user's location usually demands a large amount of energy used by the GPS receiver. In order to decrease the power consumption of the mobile device, it is discussed in section 4.5, a solution that manages the GPS receiver activation making it work only when it is really needed.

Follows the application requirements sum-up:

- Secure authentication system;
- Devices' pairing mechanism;
- Ability to send and receive location tracking data;
- Event-based alert notifications;
- Privacy features to handle data access between devices;
- Presence support;
- Use of standardized communication protocols;
- Bandwidth and power saving mechanisms;

4.2. Communication Architecture and Protocols

Summing up the communication protocol requirements, the protocol needs to have presence and broadcasting support, message storing and dispatching for unavailable devices, and it should be interoperable with other multi-platform server technologies.

Choosing a communication protocol for message exchange between mobile devices was a task that required some attention. Among the considered solutions, there were the Advanced Message Queuing Protocol (AMQP) and the Extensible Messaging and Presence Protocol (XMPP). In addition to those two protocols, it was yet considered the development of a proprietary lightweight binary protocol.

By developing a binary protocol we could decrease the overhead ("size") of the exchanged messages between nodes, which would be important in order to reduce bandwidth use. This however, takes time to implement and is not justifiable when there are stable and standardized protocols that support many useful features (like presence support and security mechanisms), ready to use and interoperable with a vast amount of servers and clients. This approach allows the developer to focus on the mobile application development, while leaving aside protocol and server implementation concerns.

That leaves us with two well-established protocols, the XMPP and the AMQP. In order to have a clearer idea of both protocols advantages and drawbacks against each other, Table 3 presents both strengths and weaknesses for each one of them.

Table 3 – XMPP & AMQP Strengths and Weaknesses (Kalmer, 2011)

	XMPP	AMQP
Strengths	• Presence support allows to know if the other nodes are running • It is easy to get up and running • Has a queue system with the ability to store "offline chats" • Supports feedback and control via some clients • Uses explicit identities (JID)	• There are no rosters to be managed. It is used only queues instead. • The security is managed within the broker • One to one (private) communication • One to many (fan out) communication • File transfer support
Weaknesses	• Managing the roster relationships is an hard task • An high number of presence updates can result in large overhead • REST bridge • *Broadcasting* is only supported by a protocol extension (PubSub) • The file transfers support is an "extension" to the protocol	• No presence features supported • Does not use an explicit identity

From the analysis of Table 3, we can conclude that both protocols are well suited for the task. Clearly, AMQP reveals itself better as it supports more features than XMPP does. In addition to that, the AMQP uses a binary format where XMPP uses XML, which allows AMQP to be less bandwidth consuming.

However, despite all the features AMQP may have, it lacks presence support. Since presence support allows the application to know when the other paired users are running the application on their mobile devices. By knowing when another user is available for transmitting data, the application knows whether it is feasible or not to ask for that data, avoiding the application to stay expecting to receive data that will not arrive.

Moreover, it was also discovered that an open-source Objective-C/Cocoa Touch XMPP framework already existed for iPhone (XMPPFramework, 2011), whereas no solution of this kind existed for AMQP. By using an existing framework, much time and work would be saved as well as the probability of finding bugs would be smaller.

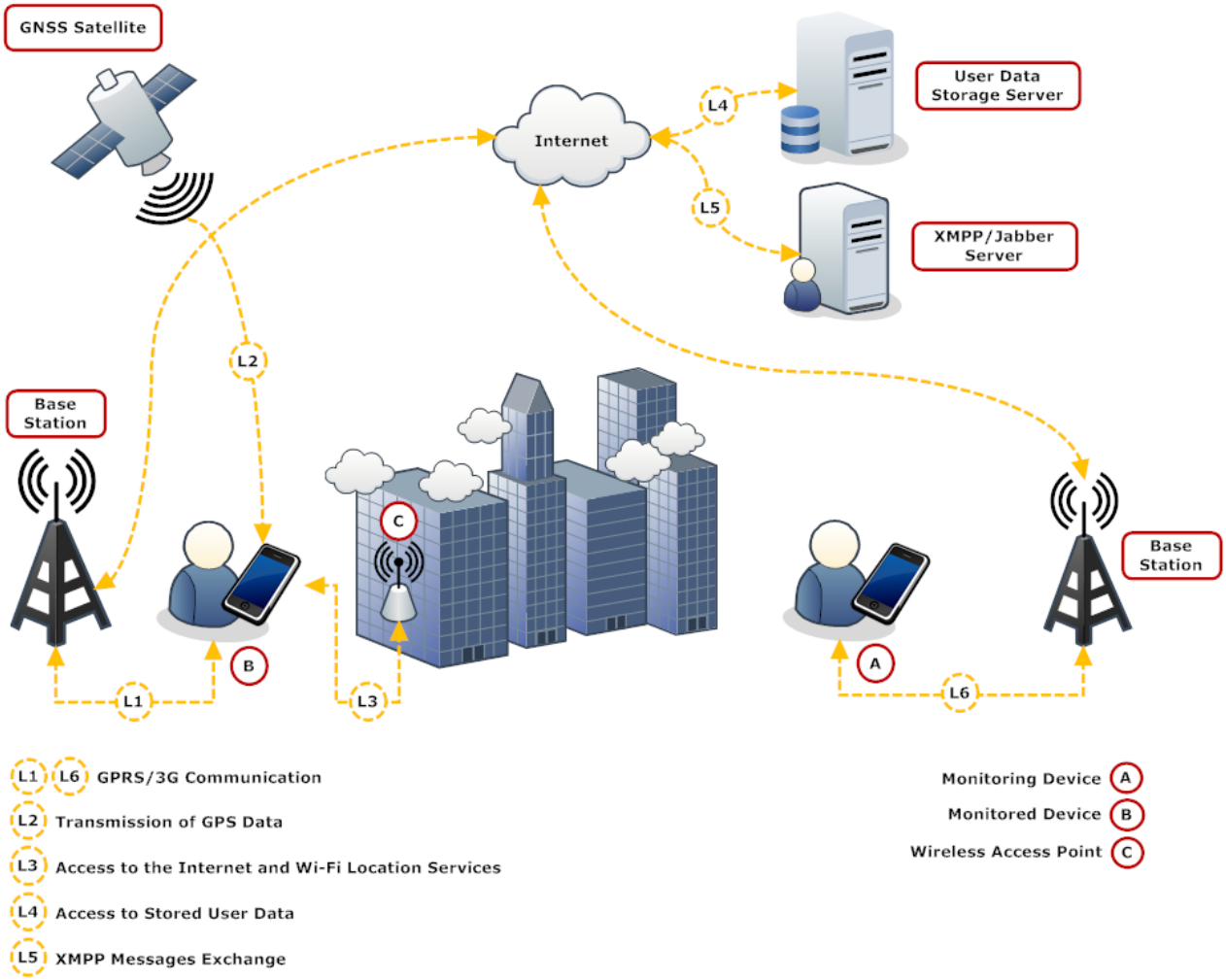

Figure 20 – Communication Architecture

Figure 20 illustrates the communication architecture of the application. ***A*** represents the device which is monitoring ***B***, the monitored device. They were submitted to a pairing process prior to any monitoring activity. Both base stations are used in order to access the Internet via an operator's mobile data plan. In addition to the *GNSS Satellite*, ***C*** represents a wireless access point which may be used as an alternative to GPS in environments where it is not available.

In the cloud, there are two different kinds of servers: a *XMPP/Jabber server* and a *User Data Storage server*. The *XMPP/Jabber server* runs a XMPP server platform which allows users to create new Jabber accounts, authenticate, pair with other users and communicate with them.

On the other hand, the *User Data Storage server* contains a database which stores information about the users, their alerts and XMPP/Jabber account information. In addition to the database, this server holds also a PHP script which handles the return of mobile application requested data, in JSON format. It also processes all the synchronization checks in order to provide the mobile devices with information about the current state of their cached data.

Every time a location request is made by a device, the following process is done:

1. ***Monitoring Device (A)*** sends a location request through ***Link 6 (L6)*** to the ***Target Device (B)***.
2. The request is routed through ***L5*** to the ***XMPP/Jabber server*** and only then, forwarded to ***B***.
3. ***B*** receives the request made by ***A*** through ***L1***.
4. ***B*** tries to receive GPS data from the ***GNSS Satellite*** through ***L2***.
5. If no data can be retrieved from the ***GNSS Satellite***, ***B*** gets information about a ***Wireless Access Point (C)*** through ***L3*** and accesses Wi-Fi location services through ***L1*** in order to try to get a position.
6. ***B*** sends a location response to ***A*** through ***L1***.
7. The response is routed through ***L5*** to the ***XMPP/Jabber server*** and only then, forwarded to ***A***.
8. ***A*** receives the location response from ***B*** through ***L6***.

The communication with the *user data storage server* is done whenever the application is launched and also when an update request is received from another user, meaning that some of his data was updated on the server. These updates are signalized through the broadcasting of XMPP messages, sent to the user's paired devices.

Another additional factor we could take into account would be the scalability. Several open-source XMPP servers support connection managers or clustering techniques. The two XMPP servers used for testing purposes, in this thesis's scope, were the *Openfire* (IgniteRealtime, 2011), a java based server, and the *eJabberd* (eJabberd, 2011) which is written in *erlang*, a programming language used in several large communication systems.

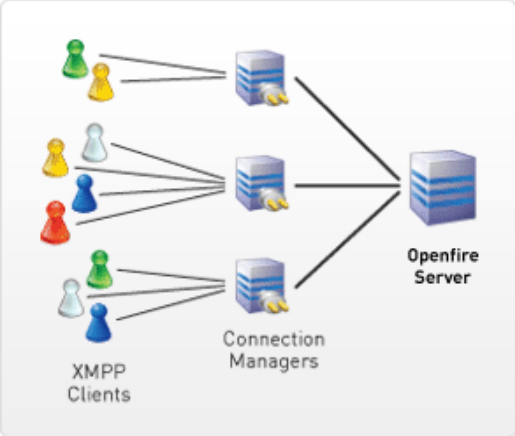

Figure 21 – Openfire Connection Managers (IgniteRealtime, 2011)

In *Openfire*, the distributed capability is reached through the use of external connection manager modules (Figure 21). Each *Openfire* connection manager module deployed increases the scalability by allowing the handling of at least five thousand concurrent users' connections (IgniteRealtime, 2011b).

In *eJabberd*, a XMPP domain can also be served by a cluster of nodes connected via a network, which uses a service load balancing system that automatically distributes all the traffic (ProcessOne, 2010). For the purpose of this thesis, connection distribution systems

were not considered. They would require a large set of machines, either servers or mobile devices, which were not accessible/affordable for testing purposes, in order to see the actual behaviour in a real deployment environment.

4.3. Authentication System

The implemented authentication mechanism suffered some changes throughout the application development. Initially, each iPhone would have an exclusive account in the XMPP/Jabber server associated with it, which means that what was being identified was not the user itself but the mobile device. This was achieved by reading the UDID (an acronym for Universally Device Identifier) directly from device's hardware.

When manufactured, every iPhone is assigned with a UDID identifier which is unique to that physical device. It is usual to confuse UDID with UUID (Universally Unique Identifier). Though they may serve similar purposes, they are different in size (UDID is 160 bits long while UUID is only 128 bits) and while the first one is calculated from hardware embed information, the last one is software-generated for a software object. Figure 22 shows that anyone is able to see the UDID of their iPhone through the Apple iTunes software.

Figure 22 – iPhone UDID Displayed on iTunes

By developing using this approach, we were taking advantage of some benefits. One of them is related to the account creation. The account would be created automatically since by resorting to this mechanism, the only parameter needed would be the UDID, letting the user start using the application right away, without providing a single piece of information.

Other implicit advantage is related to the database maintenance itself. Web applications requiring user registration usually have their database records growing exponentially, due to the amount of user registrations. The truth is, if we could be able to group the number of accounts per user, we would certainly be surprised with the amount of redundancy that exists. And why this happens? The answer is quite simple. Human brain is not perfect, so users keep forgetting passwords all the time, mainly due to the amount of authentication systems they use in a daily basis. But the human brain can sometimes be lazy as well.

For many people, it is easier to create another account instead of trying to recover the password, by just simply requesting it to be sent via email. This just happens because these applications (mostly web ones) are meant to be used by several different devices and there's not a proper way to identify, exclusively, each one of them. Besides that, a computer may not be as personal as a mobile phone is, so it is possible that it may be used by several persons. Since with mobile device applications the exclusive identification is possible, this authentication mechanism would allow existing only one account per mobile device, thus reducing the number of inactive accounts drastically, while keeping database performance at its best.

4.3.1. UUID and Its Issues

The major issue encountered while developing the application was related to privacy concerns. Apple provided access to UDID in order to allow application developers to uniquely identify a device. It would be essentially used for purposes such as storing application preferences or video game high scores. But despite its great convenience for developers, the UDID usage holds a potential for the abuse of privacy.

A clear example of an unwanted UDID usage may be an advertiser, or other entity who wants to track user behaviors and online patterns, using the hardware's unique identifier for these purposes.

Eric Smith, assistant director of Information Security and Networking for Bucknell University (Smith, 2010), made a study which reveals several interesting things about iPhone applications and their underlying privacy issues. He states that third party applications could easily use users phones' UDID, along with their current IP, to track users' location in real-time. However, he also explains that the real problem is when companies send or even sell this data to other advertising agencies.

Smith's also claims in his study that about 68% percent of the 57 top App Store's applications tested transmit the UDID back to a remote server, and only 18% encrypt the data. Additionally, it is important to refer that many of those applications involved in this study, belong to large companies such as Amazon and Chase Bank and while a few were secured with SSL, many of them did send the UDID along with personal info via plain text. This essentially means that anyone who intercepts it can easily view it.

As of this writing, there is no ability in iOS to block the visibility of the iPhone's UDID to any installed applications, nor any mechanism to prevent the transmission of the UDID to third parties. In fact, Apple addresses this concern in their application development guide by stating the following: *"For user security and privacy, you must not publicly associate a device's unique identifier with a user account"* (Apple Inc., 2010b).

Another problem lies in the fact that UDID cannot assure reliability on *jailbroken* devices. Some users apply the process of *jailbreaking* to their equipments in order to gain full access (also known as root access) to all the features of the operating system, which Apple restrict to the end user (these include unsigned applications installation, system software tweaking, adding additional features to operating system, etc.). Thus, it is not surprising that there are applications conceived to spoof the UDID to any other application that requests it.

UDIDFaker (MyRepoSpace, 2011) is a well-known application by the jailbreak community, which essentially does what its name says. Early this year, *jailbreaking* was considered legal

in the U.S. against all Apple's objections, where federal regulators declared that there is no basis for copyright law to assist Apple in protecting its restrictive business model (Dailyator, 2011). Despite all its legality, Apple's engineers work hard to keep newer iPhone iOS versions clean from flaws which may allow *jailbreaking*, mainly due to piracy concerns and network unlocking.

The usage of UDID as user identifier may also be troublesome, since users sharing the same device, cannot be uniquely identified (only the physical device can). This could lead people whom users share its location with, to be unaware of the person that is in fact using the mobile device.

Using UDID as a pairing resource also has been proven painful for the users as well. The 160 bits hash is calculated based on iPhone's hardware, more precisely the phone's serial number, IMEI, and both the MAC addresses of Wi-Fi and Bluetooth modules. Figure 23 describes a simple Python script that can be used to calculate the UDID of a device.

```
import hashlib

serial=""         #go to your iPhone's Settings - General - About
imei=""           #go to your iPhone's Settings - General - About
wifi_mac=""       #go to your iPhone's Settings - General - About - WiFi
Mac
blue_mac=""       #go to your iPhone's Settings - General - About -
Bluetooth MAC

udid=hashlib.sha1(serial.upper()+imei+wifi_mac+blue_mac).hexdigest()

print udid
```

Figure 23 - UDID Calculation Python Script (121282.com, 2011)

Despite the security it provides, the 40 character length of the hash makes it difficult to use as Jabber ID (known as JID in XMPP Protocol) as it is very hard to memorize and share among user's contacts. An attempt made in order to minimize the hash's length, was encrypting it with SHA1 and MD5 algorithms. While SHA1 is stronger than MD5 algorithm, the last one returned a smaller string. However, it was still too big and hard to memorize and share. Putting this problem together with all the other issues, this approach was thus discarded, as user needs an ID which can be easily memorized and suitable for sharing purposes, like a custom username or email.

Due to all the facts explained before, it was decided to implement a two layer authentication system based on email and password combination, which aftermost gives access to the XMPP/Jabber account login data (the whole process that occurs at each layer is explained in more detail in Chapter 5).

For security purposes, the Jabber account is created using a random generated SHA1 hash for its Jabber ID (which then stays permanent for this user) which is only known by the internal system. Then, the email provided by the user is internally referenced to that Jabber ID, so that whenever the client application logs in and asks for the user's Jabber ID, it is encrypted and sent to the mobile device. In turn, it is decrypted and gets ready to use for communication with paired nodes. This approach provides interesting advantages in the extent that, it is given the opportunity to the user to change every field of the login data

(particularly the email), without affecting the internal Jabber node relationships. An email address is also easier to remember and share with other users, for pairing purposes, just like it happens with the majority of instant messaging clients.

4.4. Device Pairing Mechanism

Users need a pairing system that allows them to add new devices to be monitored (or vice versa), in their application, or to remove them at any time. The iPhone has a set of communication components that can assist in implementing different pairing mechanisms and privacy features, either as a full alternative or just as an auxiliary method. However, it is necessary to compare the advantages with the negative aspects of each technology, in order to conclude whether it really justifies its use or not. This section evaluates each pairing method which was considered, during the application development.

4.4.1. GPRS/3G/Wi-Fi

This is the main communication technology that allows every single feature of the application to work properly. This means that no technology other than this one is required in order to use the application. Regarding devices pairing, this kind of technology requires Internet connection on both mobile devices to execute the pairing process and it may require more time when compared with other technologies, since such kind of communications usually have longer delays.

Advantages:

- The physical proximity of the equipment is no longer a barrier; the pairing process can be accomplished regardless of the distance separating both devices.
- Every single feature of the application can solely rely on this technology.

Drawbacks:

- Requires Internet connection to be present on both mobile devices.
- Consumes more bandwidth.
- The pairing process can become more time consuming due to congestion situations on the server or due to weak GPRS/3G/Wi-Fi signal reception.
- Increases the load on the XMPP/Jabber server.

Operating Mode:

It were considered two distinct ways of pairing for this type of technology, both related in the way in which each device is identified. In one case, it could be used a custom user identifier such an email address. On the other hand, it could be used the SIM card number. With this approach, we could easily see who from our contact list uses the application and directly ask for pairing without any other time consuming task.

Although it seems to be the best method, it raises several issues like authenticity problems as well as additional costs, not so much for the user, but for the company/developer owning

the application. Apple does not provide an API call that allows the SIM card number to be retrieved, which means it must be the user itself to enter its number manually. Besides that, in order to check if the phone number the user submits is authentic, there is a need for a mechanism that would send an SMS to the phone number with a validation code. That code would have to be subsequently entered, so that the validation process could be completed. This complicates the overall process and implies additional costs which may vary depending on the number of registrations.

4.4.2. Bluetooth

The use of this technology in pairing, cannot exclude the requirement to establish a connection with the XMPP/Jabber server in order to fully complete the process. However, it can act as an auxiliary method in many situations such as the server unavailability, Internet connection failures or even in situations where the user is not willing to spend too much time in the process.

Advantages:

- Improved pairing speed.
- Bandwidth saving.
- Only one device needs to communicate with the XMPP/Jabber server in order to acknowledge new entries to user's roster.
- Reduces traffic congestion in the XMPP/Jabber server.
- One of the devices may lack Internet connection at the time of pairing, since only one of them is responsible for consolidating and sending pairing information to the XMPP/Jabber server.

Drawbacks:

- Need for physical proximity to establish a connection between devices.
- Does not eliminate the need for communication with the XMPP/Jabber server to complete the pairing process, although it reduces the exchange of packets.

Operating Mode:

The application detects a list of devices in the proximity and offers the user the possibility to choose one of them for pairing. In turn, the application running on the chosen device receives the request, and then asks its user for permission, in order to accept the pairing process.

The device initiating the pairing, receives a confirmation along with the information required about the other device, and if it has Internet connection with the XMPP/Jabber server then submits the data, otherwise it finishes the Bluetooth connection with the other device by telling it that it does not have Internet connection, so that it may attempt to submit the pairing data itself. If none of the devices could send the data at this time, it is stored on both devices and the first one to establish a connection with the server will attempt to send the data.

It is important to note that, only when the XMPP/Jabber server is notified of the pairing process, devices will appear on each of the user's roster allowing then to perform monitoring actions over each other.

4.4.3. SMS

The interception of short messages by the application would be a way to save users' mobile data plan bandwidth, in addition to decongest the access to XMPP/Jabber server. Since the majority of the mobile operators are abolishing the SMS taxation in some of their price plans, for messages exchanged between costumers within the same network, this could be a suitable mechanism for the pairing process. However, this implementation raises several issues mainly related to safety, since text messages can easily be forged by any equipment/service enabled to send them. A way to overcome this problem could be, for example, implementing an encryption algorithm.

Advantages:

- Pairing may eventually be initiated by selecting a contact in the contact list of the device itself, eliminating the need to find surrounded devices or to know the User ID (email) of the target user.
- The physical proximity of the equipment is no longer a barrier; the pairing process can be accomplished regardless of the distance separating both devices.
- One of the devices may lack of Internet connection at the time of pairing, since only one of them is responsible to consolidate and sending the data to the XMPP/Jabber server.
- Reduces traffic congestion in the XMPP/Jabber server.

Disadvantages:

- Some operators may charge for sending SMS, especially if communication is made between different mobile networks.
- Does not eliminate the need for communication with the XMPP/Jabber server to complete the pairing process, although it reduces the exchange of packets.
- The device is always receptive to new messages, which may allow malicious people to easily take advantage of application security flaws.

Operation Mode:

The user chooses the desired user from its contact list and the application sends an encrypted SMS. The remaining process is similar to the Bluetooth mechanism.

4.4.4. Conclusion

The GPRS/3G/Wi-Fi connection is always needed to finish a pairing process, since Internet connection is always needed in order to interact with the XMPP/Jabber server. In both Bluetooth and SMS pairing mechanisms, they fail to successfully eliminate the need for communication with the XMPP/Jabber server to complete the pairing. But it is important to highlight that these two mechanisms may be used as auxiliary ones, as they are proven to provide some flexibility in many situations as stated above.

The only difference between the Bluetooth and SMS systems lies in the fact that SMS can transport information over any distance between the physical devices, whereas Bluetooth

cannot. This means that SMS, though its limitations, could very well replace the use of an Internet based mobile data plan to support communications between devices.

However, the application architecture would need to be completely different and would not bring any added value to it. In fact, some useful features like the possibility to see the current state of a user – if it is online or not – would be compromised.

Table 4 – Different Pairing Mechanisms Comparison

	Bluetooth	GPRS/3G/Wi-Fi	SMS
Physical Devices Location	Near	Any	Any
Bandwidth Consumed	None	High (considering the poor traffic limits currently imposed by operators)	None
Dependency on Internet Connection	No[1]	Yes	No[1]
Pairing Mode	Device Search and Selection	Unique User ID	User Phone Number
Involved Costs	Mobile Data Plan	Mobile Data Plan	SMS Fees[2]

Some considerations:

[1] Although the devices do not depend on an Internet connection to initiate the pairing between them, the process is only complete when the pairing data is submitted and validated by the XMPP/Jabber server, thus the requirement for Internet connection is not fully eliminated.

[2] Even though there are many mobile plans where the operator does not charge for sending SMS between its customers, these are often subject to a weekly limit and they do not apply when it comes to communication between different mobile networks. The point stated above also applies.

4.5. Bandwidth Saving Mechanism

The choices provided by mobile operators, with regard to data plans, are still too expensive and restrictive. The reasons for such to happen may be the weakness of their own network infrastructures, which may not be ready to support high traffic demands (the USB dongles that plug into laptops and allow accessing data through mobile networks are a cause of the service quality degradation, as in some cases, they benefit from unlimited data plans) or just an attempt to avoid the use of third party VoIP technologies. These technologies have been proven cheaper and may result in losses to the mobile operators. This fact induces the need to optimize applications in order to generate lower network traffic, which can bring visible benefits to the consumer, not only economically but also regarding performance.

Since the application needs to access information about other users (user profile, alerts, etc.), it needs to download all that data, every time it is launched. This has an impact not only

in the bandwidth used, but also in the application performance, especially regarding the initial synchronization times that it takes at the application launch. Therefore, a system has been studied to optimize the application, in these two aspects, by creating a cache mechanism. A diagram of the overall process is depicted in Figure 24.

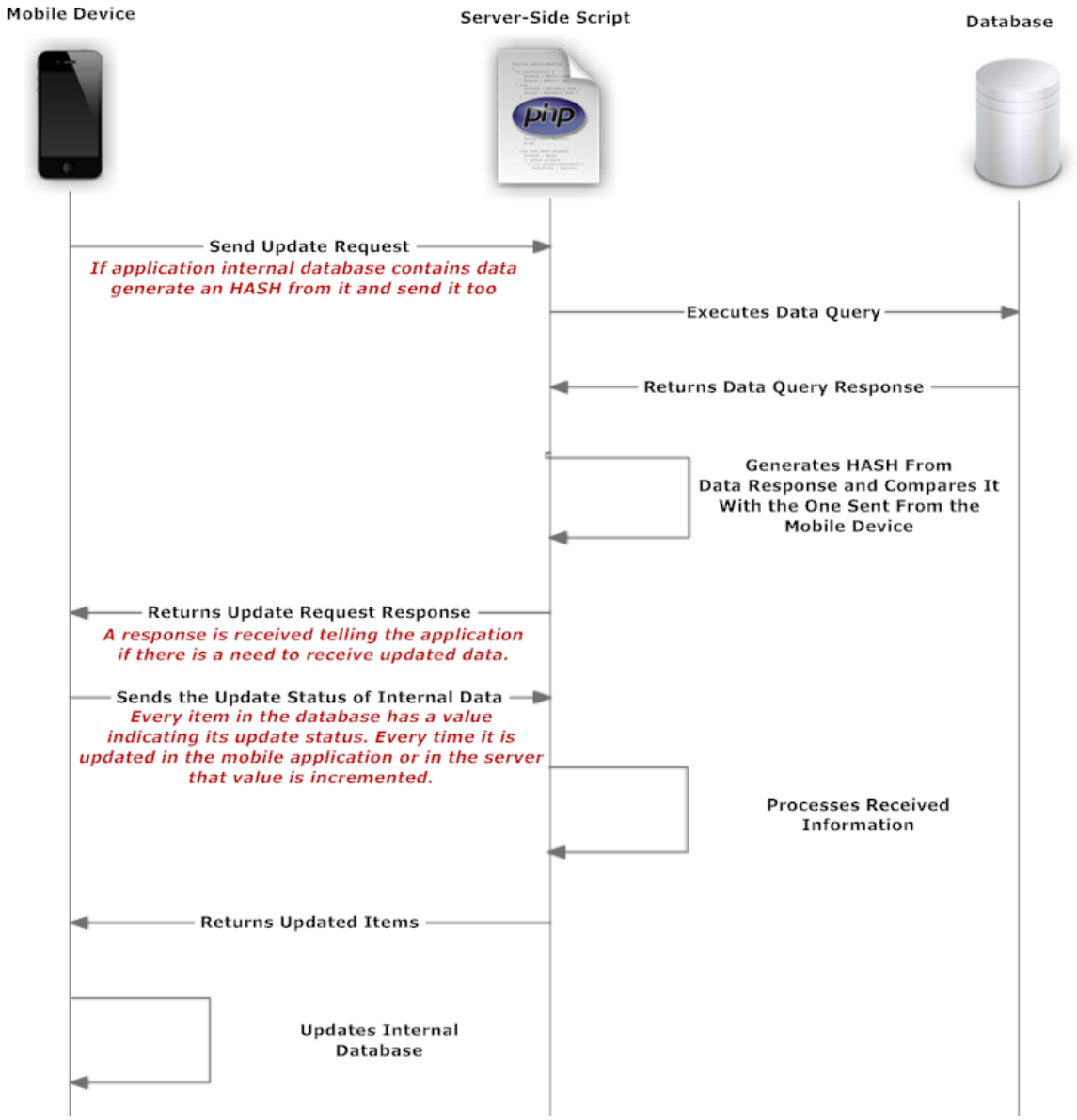

Figure 24 – Cache Mechanism Overview

To better understand the whole process, let's approach it in steps with greater detail.

1) The application running in the mobile device (it does not matter if it is a monitoring or a monitored device as this process occurs in both cases) checks its internal database for data and if it is not empty (that usually does only happens prior to the first pairing be made), the application generates a hash from the whole data. This means that any minimal change made to the internal database will result in a completely different generated hash, the next time this step is taken. Then a request is sent along with the generated hash to the server.

2) When the server-side script is executed, it queries the database data (all the returned data must be in sync with the one contained in the mobile device – the structure is the same, so when both sides are in sync, the generated hashes always match).

3) When the data is retrieved from the database, the server generates a hash from it using the same algorithm as it is used in the mobile application and then, it compares that generated hash with the one sent from the mobile. If both hashes match, then a response is sent to the mobile device telling that its data it is up-to-date. If that does not happens, a message is sent telling the application that it needs to receive updated data.

In order to proceed to the next steps, it was needed to evaluate solutions to determinate which part of the whole data is already up to date and thus, does not need to be sent by the server, so that the minimum bandwidth is used. Two solutions were approached, both relying on a basic principle, to make possible to determine which records should be downloaded by the mobile application, in order to keep the cache data up-to-date. This principle lies on the existence of a numerical field for each record in the tables.

Initially it was considered the use of a date/time field instead of a numeric one. However, it turned out that it was not feasible since our main goal is to minimize the waste of bandwidth and as we may realize, transmitting a date and time string for each record over the network will too much data. A simple numeric value, that never gets too much bigger due to the temporary nature of the records, is a better approach, since the expected number of changes to a single record is low. Even for a record that has been updated one million times, the string transmitted still being smaller rather than using a date/time formatted string.

For future reference on this chapter, let's call this update status field *updateIndex*. As explained before, for each new record created in a table, this field will have a numeric value initialized in zero. Therefore, each time a record is modified, its *updateIndex* field is incremented by 1. It is important to refer that, the magnitude of the value is not that important, because the need for update is considered if its *updateIndex* value is simply different from the value of the same record,that the mobile device contains in its internal database. This methodology allows keeping the value always low, for minimal bandwidth usage.

Let's consider the two different solutions to determine which data needs to be updated. Let's call them **Solution A** and **Solution B**.

In **Solution A** (Figure 25), the server sends each record ID along with its *updateIndex* value for every data, allusive to the user that is using the mobile application. Then the mobile device processes all the received data by comparing the *updateIndex* for each record in its cache, thus defining which records must be received from the server to update the internal data. The result of this processing is submitted to the server, using the minimal data as possible (just the type – *table name* – and record IDs). In turn, the server answers back by processing and sending only the full records that are outdated in the mobile device, allowing it to update its cache. Figure 25 illustrates the requests and responses between both devices and in which entity does data processing takes place.

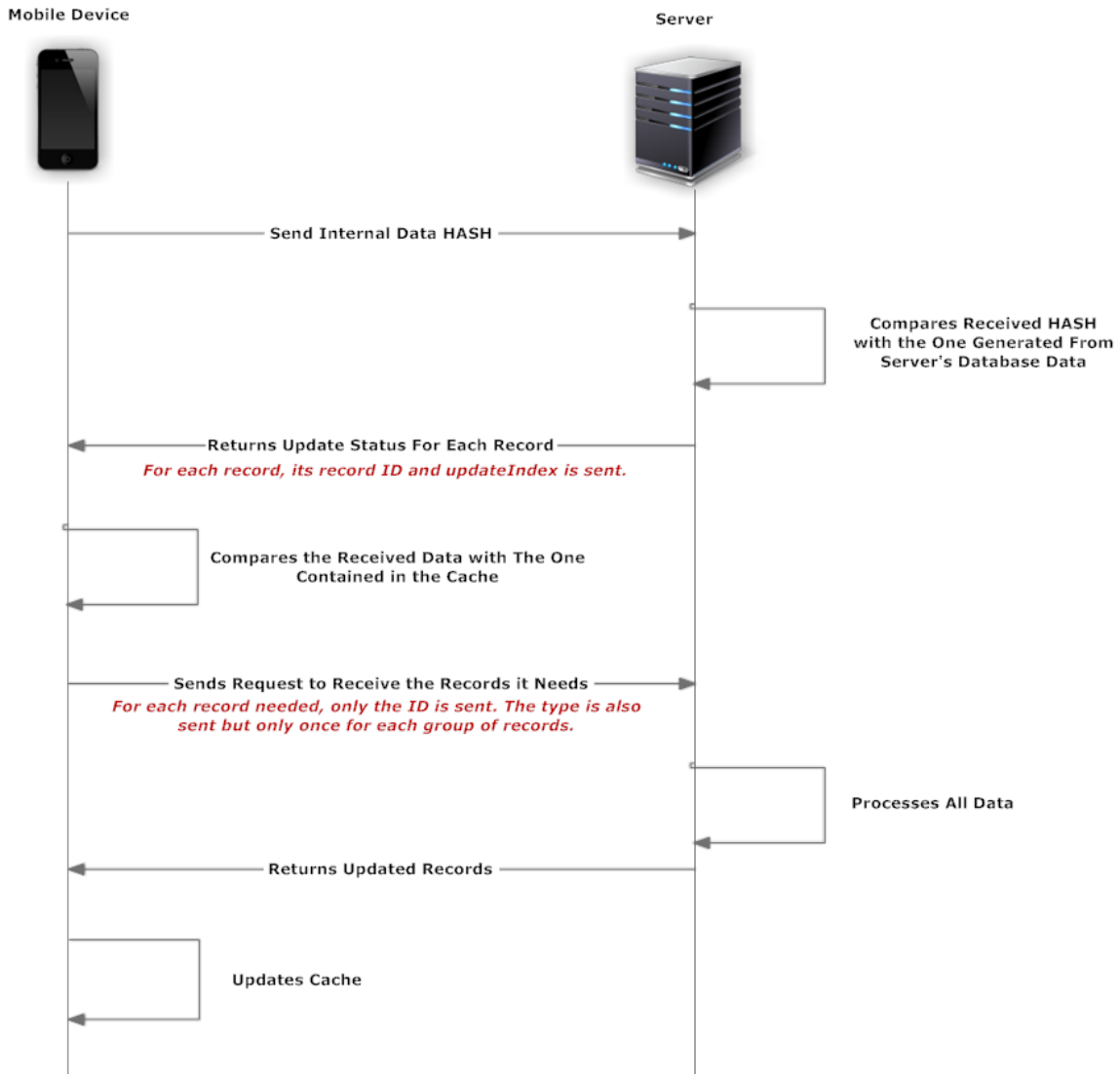

Figure 25 – Outdated Records Update Mechanism (Solution A)

For the **Solution B** (Figure 26), almost the same as above (Figure 25) applies with the only exception that, it would be the mobile device itself to send the record IDs and *updateIndex* fields to the server. In turn, the server would process that request and would return all the data in need of updating, just as is shown in (Figure 26).

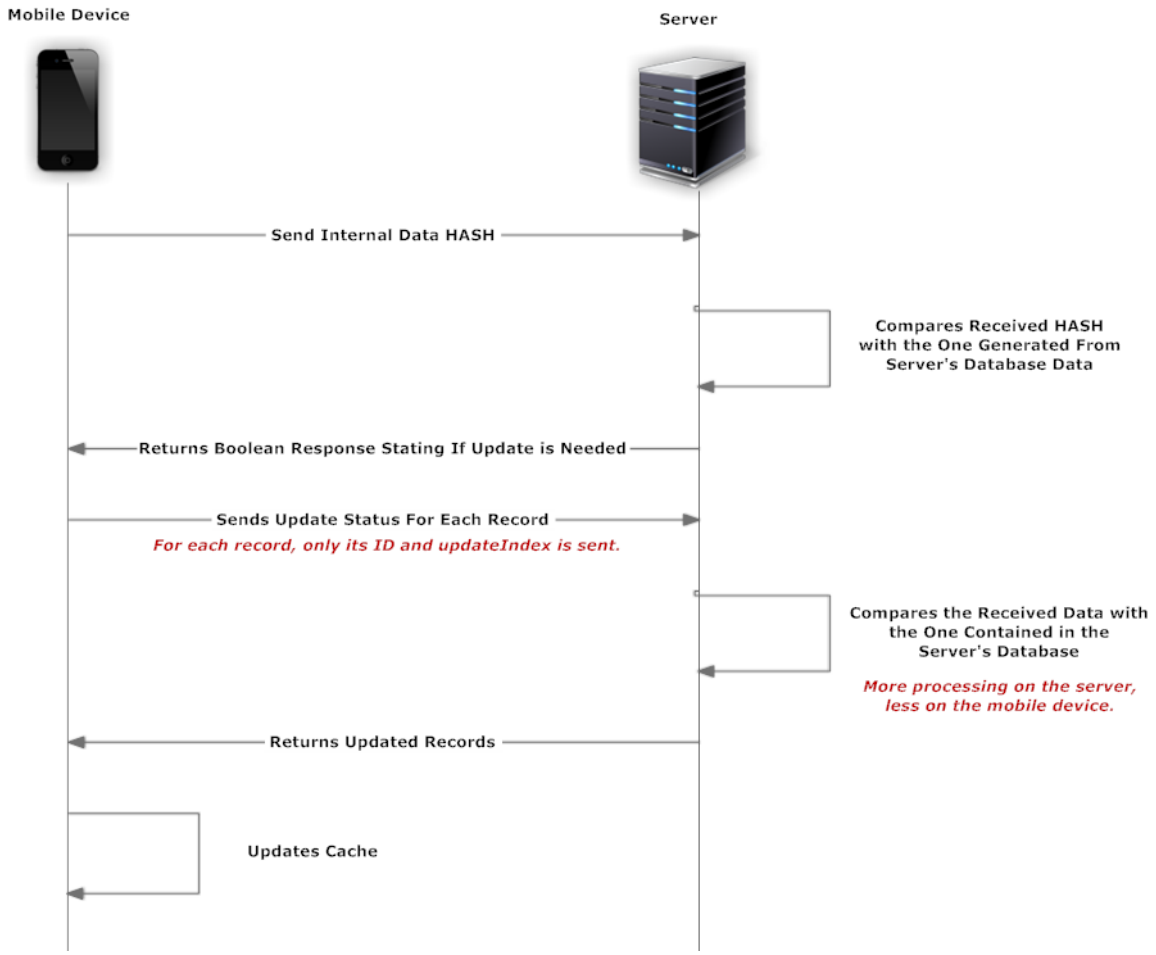

Figure 26 - Outdated Records Update Mechanism (Solution B)

Given these two assumptions, we have to consider two key factors: the server and mobile device processing needs and the bandwidth consumption between them. By analysing the two diagrams we can see that, although they may seem very similar at first glance, both have points of greater processing load in different entities and different amounts of data to be transferred as well.

Regarding the processing load, it is better opting for solutions that attribute the more extensive processing tasks to the mobile devices of the users (considering that the impact on autonomy will be insignificant and that the target devices have capacity to satisfy the processing requirements). This relieves some of the load imposed on the server because of its several parallel requests coming from other users that it may also have to fulfil.

Hence, regarding the computational load, **Solution A** seems to be definitely the best approach, given that all the processing work to accomplish the task of identifying which records needs to be updated is done locally, on the mobile device (and just for its user, not for many as it would be on the server). This leaves the server only in charge of simple database queries. In the **Solution B**, the server would be responsible for, in addition to executing queries to the database, doing all the processing which is done exclusively by the mobile devices in **Solution A**.

Regarding bandwidth consumption, **Solution B** proves to be the best choice. Figure 27 demonstrates that.

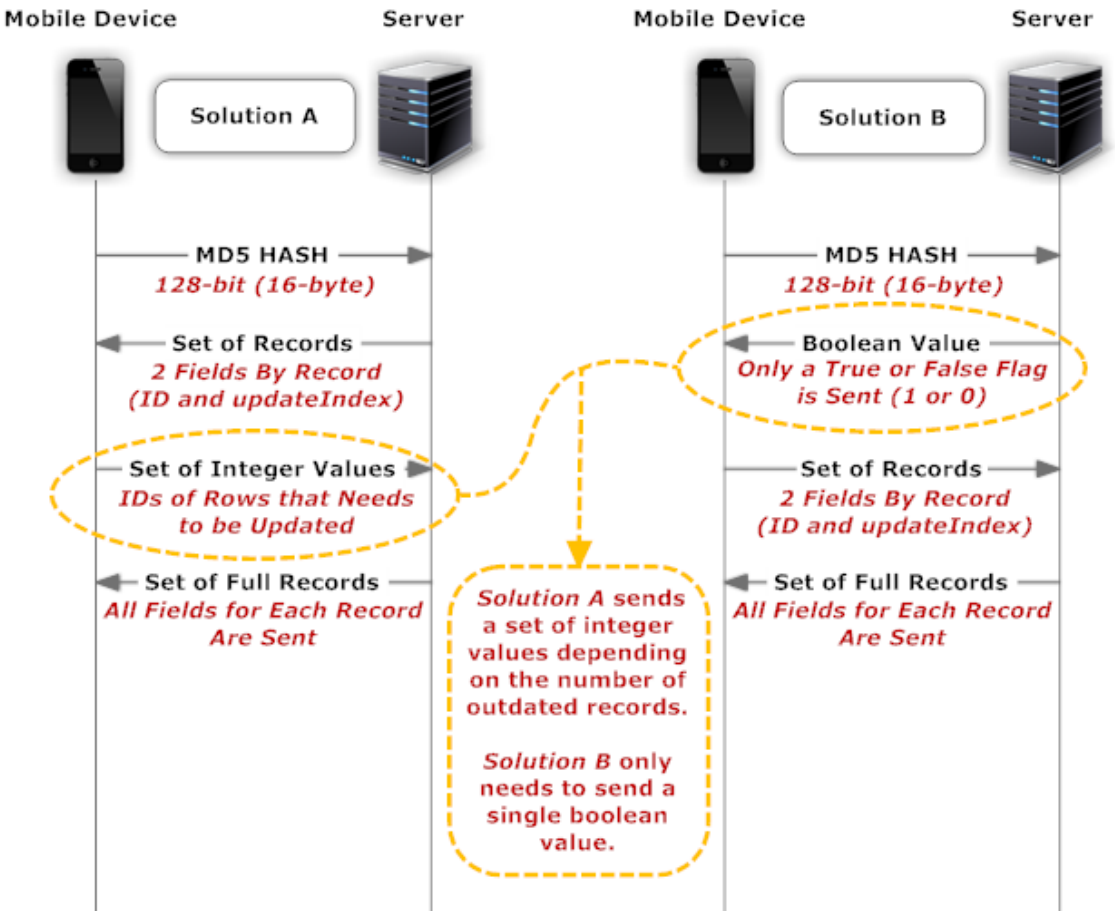

Figure 27 – Bandwidth Consumption Comparison Diagram

Whereas **Solution B** sends only a bit, which is a flag responsible by telling the mobile application that its cached data needs to be updated, **Solution A** has to send a set of integer values. Therefore, the bandwidth will vary depending on the number of outdated records and the size of the identifier for each record (which obviously tends to increase each time a new record is created). As the data is sent using JSON technology, these facts may be considered since they may affect the bandwidth usage in long terms.

Clearly, we ended up with a dilemma. Whereas the **Solution A** optimizes the processing load by relieving the load of the server distributing the processing requirements by the different mobile devices that accesses it (in a distributed computing fashion), **Solution B** is able to provide better usage of bandwidth, while not taking into account the processing load. Given the importance of the server scalability in medium/long term and that the impact between the processing load and bandwidth consumption is much worrisome in the first case, it was decided to adopt **Solution A**.

4.6. Power Saving Mechanism

A mobile device is composed by many hardware pieces, although some of those use more power than the others. One of the most useful features ever implemented in handheld devices was the GPS receiver. Unfortunately, this is one of the most power consuming

devices if not the most one, which can drain a mobile device battery relatively faster. Due to this fact and given the application needs to make use of GPS in order to detect users location, it was approached a mechanism that smartly coordinates the activation and deactivation of the GPS receiver. This allows saving a considerable amount of energy on the mobile device depending on the usage frequency of the application.

The application needs to be constantly monitoring the user's position in order to know when to launch an alert that he or other users may have set, which means the GPS receiver would need to be always on. In order to keep the GPS receiver activated only when needed, the next solution was approached.

Whenever the mobile application is launched, the user's current location is detected. Then, for each alert, the application tries to detect which one has its trigger location closer from the one the user is at the moment. With that information retrieved along with the direction and speed at which the user is moving, it is calculated an estimative for the time it takes in order for the user to reach the trigger location of the alert. This calculation must include a margin of error that considers both speed and target variations.

The final result is the interval of time between GPS receiver reactivations. When the application is launched, the margin error is high. It gradually decreases or increases as the distance the user is from the target location of the nearest alert decreases or increases, respectively. The faster the user gets closer to the target alert location and the more it moves towards its direction, the less the margin of error will be. This margin is then decremented to the estimated time to reach the trigger location of the nearest alert.

Let's now see how the margin of error is calculated.

Defining:

$ME = Margin\ of\ Error$
$DistanceW = Distance\ Weight$
$DirectionW = Direction\ Weight$

The margin of error considered for the mobile application takes into account the speed, distance and angle. It is calculated using the next formula (1):

$$ME = DistanceW.0{,}7 + DirectionW.0{,}3 \qquad (1)$$

As we can see, the distance is set to have more impact than the direction itself. The direction variation is always a factor that can predict reasonably if the user is subject to reach the target location. However, since it may have sudden changes constantly, one cannot rely so much on it. On the other hand, every time the distance to the alert location decreases, it is an undeniable indicator that the user is on the right track to trigger the target alert, thus explaining why its weight is so high in the formula above (1).

Calculating Distance and Direction Weights:

Assuming that every time the GPS receiver is reactivated and a new position is obtained, the last known position stills accessible:

$LP = Last\ Known\ Position$
$NP = New\ Position$
$TP = Target\ Position$
$D \underset{LT}{\rightarrow} = Distance\ from\ LP\ to\ TP$
$D \underset{NT}{\rightarrow} = Distance\ from\ NP\ to\ TP$

Figure 28 – Distance and Direction Weight Calculation Elements

Figure 28 depicts the different components needed to calculate the distance and direction weights. While LP and NP refer to the user's position, TP refers to the target alert location.

The distance from LP to TP ($D \underset{LT}{\rightarrow}$) and NP to TP ($D \underset{NT}{\rightarrow}$) as well, can be calculated by:

$$D \underset{LT}{\rightarrow} = \sqrt{(\mathrm{TP}x - NPx)^2 + (\mathrm{TP}y - NPy)^2} \qquad (2)$$

$$D \underset{NT}{\rightarrow} = \sqrt{(\mathrm{TP}x - NPx)^2 + (\mathrm{TP}y - NPy)^2} \qquad (3)$$

It is important to note that the target alert position has an assigned radius which may decrease the distance to the point of triggering, thus it must be considered and decremented to these two calculated distances.

So, knowing that we have both $D\underset{LT}{\rightarrow}$ (last known distance) and $D\underset{NT}{\rightarrow}$ (new distance) values, we can easily calculate the **Distance Weight**:

$$DistanceW = \begin{cases} \frac{D\underset{LT}{\rightarrow} - D\underset{NT}{\rightarrow}}{D\underset{LT}{\rightarrow}} \; for \; D\underset{LT}{\rightarrow} > D\underset{NT}{\rightarrow} \\ 0 \; otherwise. \end{cases} \qquad (4)$$

If the new distance is higher than the old one (the user is moving away from the target), the distance weight will be zero in order to avoid long GPS receiver reactivation timeouts. Now we need to calculate the **Direction Weight**. The first thing to do is to calculate the angle variation from the vector connecting LP and NP points (predicted current direction) and the vector connecting NP and TP points (predicted direction the user needs to follow in order to trigger the alert), which may be done as follows:

$$Angle\underset{LPNP}{\longrightarrow} = ATan2(NPy - LPy, NPx - LPx) \qquad (5)$$

The calculation in the equation 5 will result in a radian angle and should be converted into degrees. These steps need also to be done again in order to determine the angle of the vector connecting the NP and TP points. Considering that we have now the two angles, we must calculate the difference between them.

$$AngleDiff = \left| Angle\underset{LPNP}{\longrightarrow}° - Angle\underset{NPTP}{\longrightarrow}° \right| \qquad (7)$$

Now, the **Direction Weight** can be calculated:

$$DirectionW = |weight(AngleDiff) - 100| \qquad (8)$$

where the function *weight(x)* is defined as

$$wheight(x) = \begin{cases} \frac{x}{90}.100 \; for \; x < 90 \\ 100 \; otherwise. \end{cases} \qquad (9)$$

The function 9 states that for every *x* higher than 90 the result will always be 100. Since our objective is to have the margin of error lower when the angle between the current direction and the direction to the target is less than 90 degrees, the weight is inverted in the expression 7. This means that whenever the angle difference surpasses 90 degrees, the weight will always be zero.

At this time, we should now be able to calculate the estimated time to reach the trigger location of the alert. In addition to the margin of error, only two more parameters are needed. They are the current speed, which is easily obtained from the current GPS data and the current distance to the target alert, which has been already calculated and defined as $D_{\overrightarrow{NT}}$. Therefore, the estimated time to reach a location (in seconds) is defined by:

$$Estimated\ Time\ = \frac{D_{\overrightarrow{NT}}}{Current\ Speed}\ .\ (1 - ME) \qquad (10)$$

After the estimated time is known, the GPS receiver is deactivated and a timeout is set, based on that value, which reactivates it whenever it reaches zero. When the timeout is considerably low, the GPS keeps operating in order to avoid missing some alert triggering.

There is another problem that may happen often. This issue is related with the absence of GPS signal when the GPS is activated and a pair of coordinates of the user location is not obtained. An attempt to solve this problem is to have a timeout assigned to the GPS receiver which especially means that, if the timeout is reached and no coordinates were obtained, then it will be turned off and reactivated automatically some time later.

The time to reactivate will be dependent on estimated time to reach the trigger location of the nearest alert. So, the less this time is, the faster the GPS receiver will be activated in an attempt that no alerts will be missed (small estimated trigger location reach times does not justifies the deactivation of the GPS receiver though, thus this situation is considered as well). In the case of a location request sent by another user, if the timeout is reached (which means that no coordinates could be obtained), the application will ensure that the requesting user will receive the last known position of the mobile device.

On the other side, the requesting user will be able to know that the location sent is not accurate because whenever a location response is sent, the timestamp of the acquired coordinates is also sent along with many other parameters (accuracy, speed, direction, etc.). Through the timestamp examination, any user is able to know whether the received position data is recent or not.

CHAPTER 5

DEVELOPMENT

This chapter describes the most relevant steps which were taken when developing the application. It gives also an overall view of the different modules and presents the relationship between them. It refers to the design pattern used in the overall development architecture and also details the implemented mechanisms at a deeper level. The test phase is also discussed and its derived results are presented.

When the application first started to be developed, the current iOS - the iPhone's operating system - available was version 3.2.1 and the available device for testing purposes was an iPhone 3GS. Throughout the entire development process, the application has been updated regularly in order to support all the new released iOS versions to the date.

With the introduction of the iPhone 4 equipped with a "retina" display (an LCD featuring an high pixel density by putting a 960-by-640 pixel resolution into 3.5 inches compared to the 480-by-320 pixel of the iPhone predecessors), all the applications were required to be upgraded with an extra resolution image per each one already existent, in order to make applications look good in the new display. Along with iPhone 4 launch, the iOS 4 was also released. Although it did not bring any relevant feature to the application being developed, all the source code was upgraded in order to support devices running this version of the iOS.

Today, the application offers full support for the newly released iOS 5 and all the released iPhone models to the date, including iPhone 4, the current testing device. A fact that should be noted is that upgrading the development environment each time a new iOS version is released is not as convenient as it should be, since the SDK is not distributed as a standalone package (neither XCode is), forcing the developers to download and install the full development suite (XCode + iOS SDK) weighting more than 2 gigabytes in size.

The iOS development is based on an architectural pattern denominated ***Model-View-Controller*** **(MVC)** which isolates the domain logic from the user interface, thus separating both the parts which allow independent development, testing and maintenance for each one of them. In Figure 29 it is represented the overall development structure which relies on the MVC design pattern.

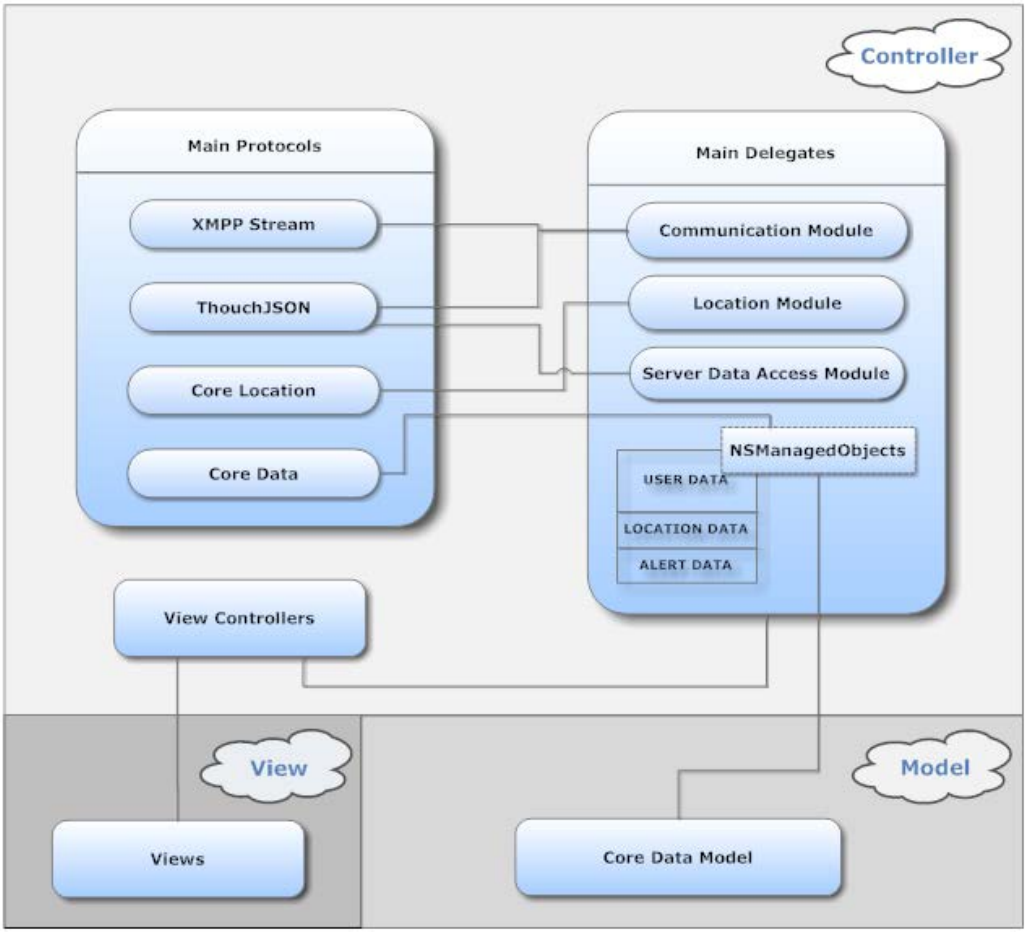

Figure 29 – Application's MVC Design Pattern Structure

In Objective-C, protocols declare methods that can be implemented by any class. They are particularly useful to declare the interface of an object while concealing its class and to declare methods that others are expected to implement. On the other hand, a delegate is an object that another object sends messages to, when certain events happen, so that the delegate can handle application specific details the original object was not designed for.

For instance, whenever a `UITableView` (interactive graphic table component) is implemented, its delegate object must be defined. That object implements the essential methods (defined in the `UITableViewDelegate` protocol) which are responsible for defining how data should be displayed, behavior and aspect details as well as responses to user interactions we want the `UITableView` element to have. XMPP Stream (module of the XMPP Framework for the iOS) and TouchJSON (TouchJSON, 2011) are part of third party frameworks whereas Core Location and Core Data are iOS SDK frameworks.

The **Communication Module** uses XMPP and JSON frameworks to implement user-to-user and user-to-server communication methods, respectively. It is important to refer that the user device may communicate with two separate (logically or physically) servers, the *XMPP server,* used to transport messages between user nodes and the *User Stored Data Server,* which is used, as the name implies, to obtain and update user information by the nodes. This module allows extra mobile device information to be added to the messages sent while keeping the remaining structure intact.

The **Location Manager Module** aggregates everything that has to do with self-positioning, that is, it is responsible for managing the GPS receiver activity and retrieving information about users' position. It is the delegate that implements iOS SDK Core Data framework methods defined in the Core Location protocol. The power-saving mechanism referred in section 4.6 is implemented in this module.

The **Server Data Access Module** handles both user information and also alerts data storing and retrieving in the server. It also implements the bandwidth saving techniques mentioned in section 4.5.

For the device internal stored data, there are three separate objects that implement the `NSManagedObject` (implements the basic behavior of a Core Data model) interface and other methods defined by the Core Data framework protocol. These objects are the ones referring to user, alert and tracking location stored data in the Core Data Model defined for each of them.

The Core Data framework is able to manage where data is stored, how it is stored, data caching, and memory management. It uses SQLite as database engine, but its major benefit is that Core Data API allows the creation and use of relational databases without the use of any kind of SQL conditions. Thus, it is possible to interact with SQLite right from Objective-C code, without needing to worry about managing the database schema or connections.

The views present the application content on the screen. They are defined in NIB files, which store information about the graphical components used in a common *XML-fashion* style. These files alone cannot define the responses to user interactions, thus coexisting with those files are the view controllers. These objects are the delegates for the views which implements `UIViewDelegate` protocol methods defining responses, appearance style and behavior the view should have.

The reader should note that, in spite of the reference to NIB files and the trivialization they assure in iOS and Mac OS X development, they are created by the interface builder assistant. It helps developers to create graphical interfaces without writing any code but they are not necessary needed in order to produce applications with graphical content. This is due to the fact that a view may be created programmatically as well as any other graphical component, because they all are an instance of the `UIView` class (or subclasses).

5.1. Views and Controllers

The application comprises many views. Most of them are `UITableView` objects, which are subclasses of the `UIView` component, part of the UIKit framework that supports most of the official iOS interface components. They are the easiest way, not only to show information in a consistent way over the whole application, but also to create option menus for user interaction. The application navigation structure is presented in Figure 30, which shows the possible user interactions since the *Splashscreen* view and thorough all the way till the most specific feature view.

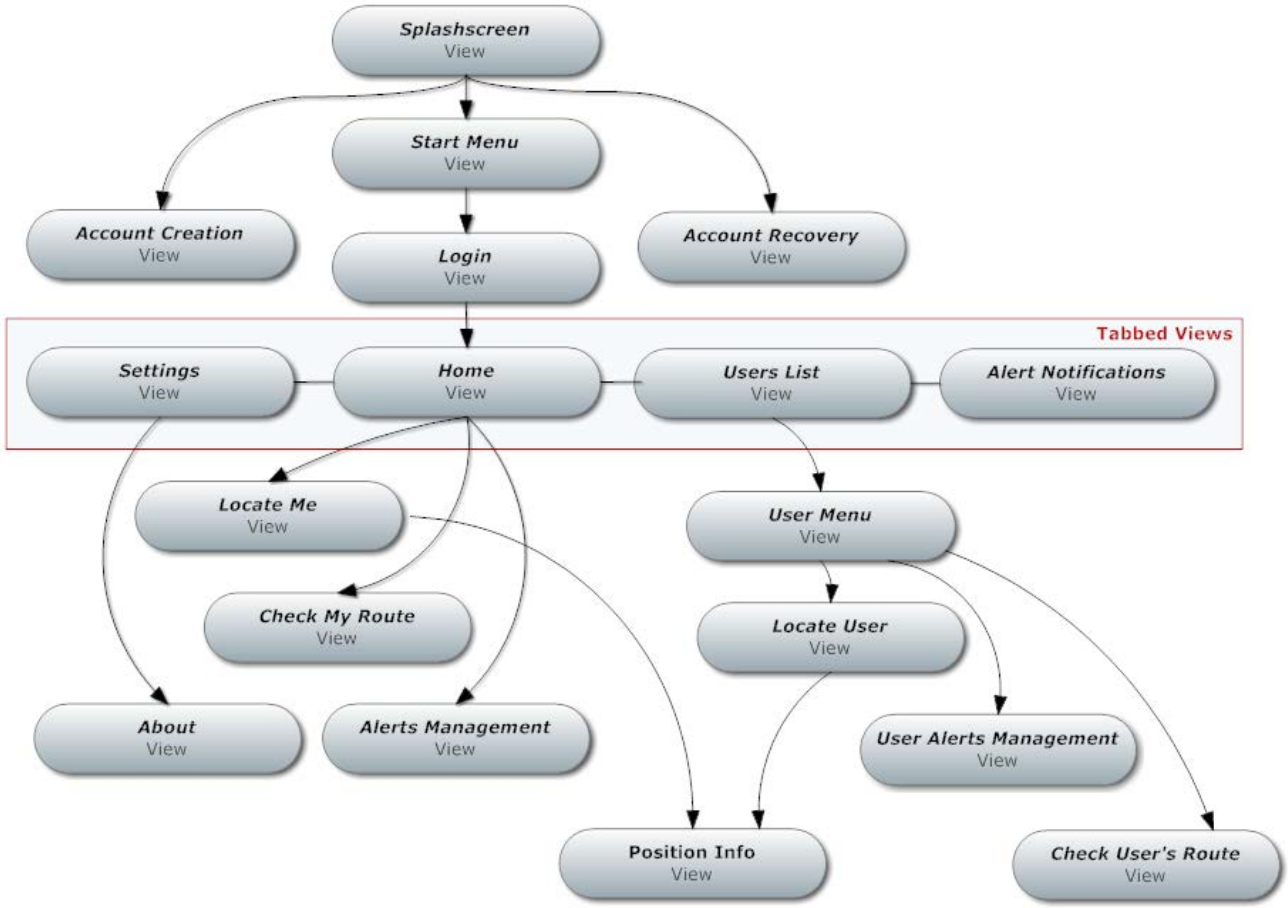

Figure 30 - Application Navigation Structure

As it is denoted on the figure above, The *Home*, *Users List*, *Alert Notifications* and *Settings* views are at the same level of hierarchy since they are supported by a `UITabBar` component, an interface element that gathers together a set of views for direct access through tabs.

There is a view missing from the diagram on purpose, as it is not considerably relevant and is temporarily presented to the user, though most of the views use it between some transitions, to tell the user that some kind of processing (getting location data, updating local or server data, etc.) is occurring at that moment. This view is the *Loading Activity* view (Figure 31b). Apart from this one, only the *Splashscreen* view is reused at the login process in order to tell the user that the internal data is being synchronized with the server (Figure 31a).

Figure 31 – a) Splashscreen view; b) Loading Activity view

The application starts by displaying the *Splashscreen* view without any activity indicator. Then the *Start Menu* view is displayed (Figure 32a). In this view, there are presented three options where the user may choose to create or recover an account or to just simply log in. When in the *Login* view (Figure 32b), there is also an option which allows automated login every time the application is launched.

For preference storing purposes, like for the automated login option, the `NSUserDefaults` class plays its role where it provides a programmatic interface for interacting with the *defaults system*, thus allowing applications to save preferences (login information, metrics, etc.) by assigning values to a set of parameters in a user's defaults database. Thus, both the login email, password and login automation preference are stored in the user's defaults database, with the exception that, while the email is stored as plain text, the password is encrypted prior to its storage. The login automation gets disabled every time a user logouts from within the application.

Both the view controllers for the login, account creation and recovery views, use the Server Data Access Module which is responsible by fetching and updating the server data through JSON. JSON support in the application is provided by the third party TouchJSON framework.

Figure 32 – a) Start Menu view; b) Login Menu view

The authentication system was made to be as simple as possible, but as secure as possible as well. By relying on a trivial login system, if any person with malicious intentions could obtain the login information, it could be a serious problem since it would not only affect the user from whom the information was stolen but also several persons. All the users of the mobile devices which were paired with a given person's mobile device would have their privacy compromised as they would be sending their positioning and tracking information to someone not legit.

With a simple XMPP client, any person with knowledge of the format of the messages exchanged by the application (which may be easy to get when not using SSL, though the developed application supports it) could generate fake requests for location, and send them to the any paired mobile device which would answer without any suspicion of what is really happening in the background. Hence, for security reasons the overall authentication system consists in two layers.

For convenience of this explanation, the *User Stored Data Server* will be just called data server, leaving the XMPP server denomination just as it is. In the first layer the user login information is sent to the data server, which is independent from the XMPP server.

Along with the login data goes the device UDID (already explained in detail in section 4.3). The data server then checks the information and if it is all correct, it grabs the user JID and Password (which are randomly created whenever a user account is created in the data server, thus the user never gets to know this information, only the server does) and encrypts it using the UDID provided. This assures that only the mobile device that sent the login request is able to know how to decrypt the information. This data is then sent to the mobile device and if it is successful in decrypting it (which should be if it is the right device), then it

can move forward to the second layer by logging in the XMPP server with the received *jabber* account details.

As already pointed out, the user never gets to know its *jabber* account information and as long as a secure connection is used, it will be very hard for anybody else to get to know this data but the server itself.

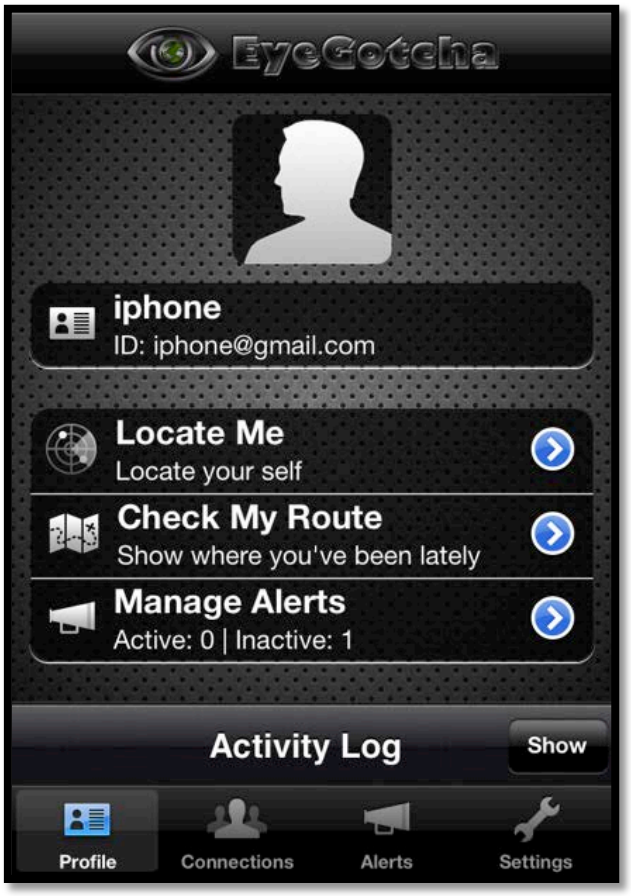

Figure 33 – Home view

When the login process is accomplished successfully, the *Home* view is presented to the user (Figure 33). It provides actions related to the user itself. From here the user can locate itself, check its route and manage all the alerts it may have defined for all of its connections (paired devices). Along with that, the user also has access to an activity log that shows all the inbound and outbound communications which were established (Figure 34).

Every time the user's location or traveled path data is requested and sent to anybody else, he is able to know that occurrence by checking his log. Information about the user to whom the information was sent, the direction and result of the communication, and also the kind of the operation are displayed.

Figure 34 – Activity log in Home view

All those activities which are displayed are notifications sent from other living objects. The great majority of them are sent by the instance of the `AppDelegate` class (the main class of the application which controls its main lifecycle events) because it is delegate of the *XMPPStream* protocol, therefore all the XMPP received messages are processed by a method defined in that class. Consequently, every location and traveled path data request among all any others received from other users, always goes through the `xmppStream:didReceiveMessage:` method present in the `AppDelegate` class.

In order to make all those request and response events to reach the activity log instance, it was used a publish-subscribe model since that instance is not the only one which may want to receive those notifications. The `NSNotificationCenter` and `NSNotification` objects are part of the Cocoa Touch framework and allow the subject (the object wanting to send the notification) and its observers (the objects that are observing for any thrown notification that matches what they are willing to receive) to communicate in a one-to-many fashion and without needing to know much about each other.

The following code is put on the subject so that, when executed, it creates a notification and posts it to the notification center. The notification center then determines the observers that are willing to receive alert notifications and sends to them via messages. The `alertInfoDictionary` parameter on line 4 represents a created `NSDictionary` that stores the received alert information to be send along with the notification.

```
1.  [[NSNotificationCenter defaultCenter]
2.          postNotificationName:@"xmppStream:didReceiveAlert"
3.          object:self
```

```
4.          userInfo:alertInfoDictionary];
```

On the observer objects the following code must exist. It states that every time a notification with the name `xmppStream:didReceiveAlert` is sent to the notification center, it should be received by the observer object in the `didReceiveAlert:` method also defined in the object.

```
1. [[NSNotificationCenter defaultCenter]
2.         addObserver:self
3.         selector:@selector(didReceiveAlert:)
4.         name:@"xmppStream:didReceiveAlert"
5.         object:nil];
```

This publish-subscribe model was widely used in the application as it was useful in several situations (to control connection and data states, for instance), where many objects needed to receive the very same notification.

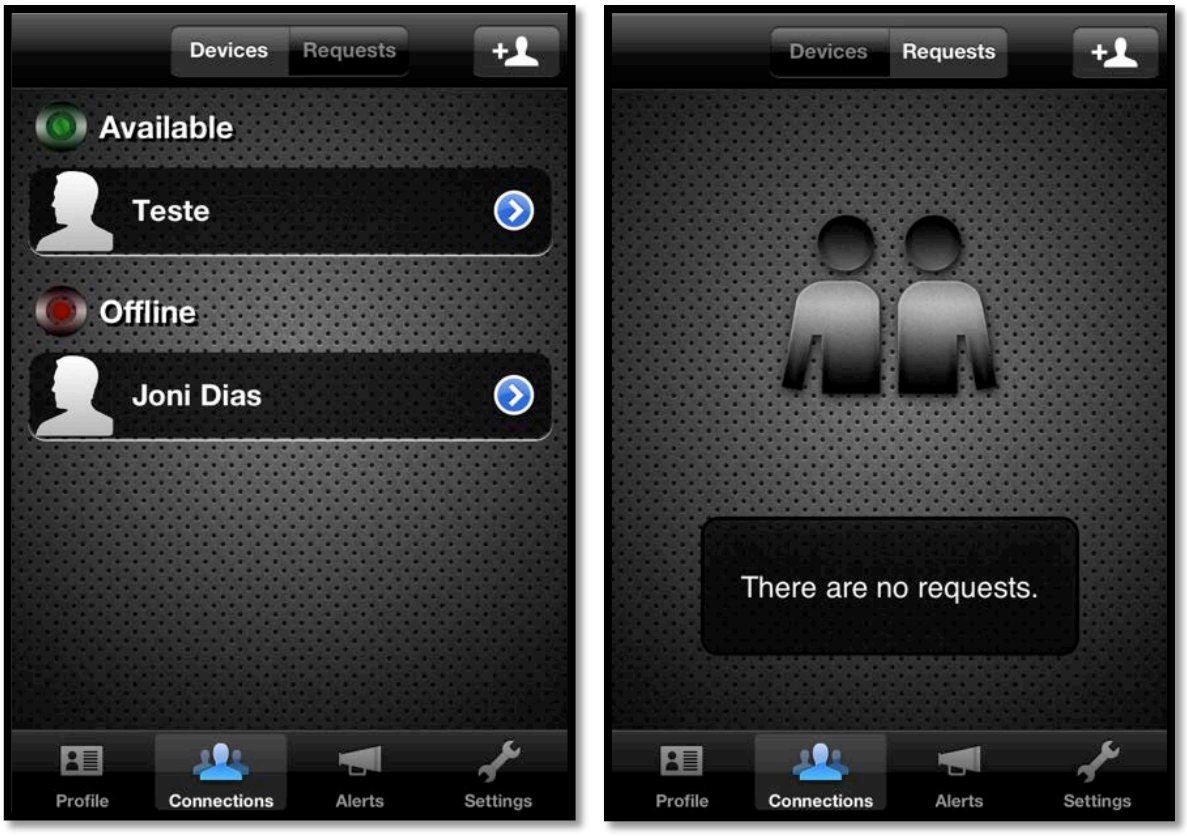

Figure 35 – Users List view

In the tab bar there is access to the *Users List* view known as *"Connections"* (Figure 35). It is on the controller for this view where most of the XMPP framework features are used. All the paired devices are shown with the XMPP presence feature implemented, which tells if a certain user is available or not. By touching in a cell of a certain user, the *User Menu* view is shown (Figure 36).

Pairing with a new mobile device is also possible through this view, by clicking on the upper right button and providing the other user's ID. There is also an option for Bluetooth pairing but is not fully implemented yet, due to the fact that there were only two testing targets, a real device and a simulator - an iOS emulator which comes packed with XCode and runs over Mac OS. It turns out that the simulator is very limited regarding hardware emulation, as it does not support GPS, Bluetooth neither accelerometers.

Whenever a pairing request is sent to another device, it is listed on the same view by selecting the "requests" segmented option present in the top bar, as it is shown in Figure 35. Then the user is able to accept or reject the request, just like in any instant messaging client. The list of users, also known as roster, is handled by a *Fetched Results Controller (*`NSFetchedResultsController` class*)*. This object assists in monitoring changes in objects and reporting them to its delegate. Those two features are very useful in order to handle the roster items presence, since they change very often and it is essential that those changes may be reflected immediately in the user interface. It also allows caching the results in order to avoid repeating computation steps, whenever the same data is displayed.

Figure 36 – User Menu view

The *User Menu* view (Figure 36) offers an overall view over the underlying user, providing a set of actions to be performed and some settings to manage privacy in an individual manner. From this view, the mobile device user may perform location and traveled path data requests (the application designates traveled path data requests as tracking requests; location requests still keeping its original designation) over another user as well as manage alerts for that same user. The privacy settings are useful to individually define whether a user is able to access our location or tracking data. Further in this chapter, we can see that both these privacy settings have lower priority over the ones present in the *Settings* view, as those last are global ones for every single paired user. All the failed attempts to retrieve data from our mobile device are logged in the activity log, so that we always know every single personal data transaction that occurs on our mobile device.

Figure 37 – a) Location view; b) Position Info view;

When the user selects the Location action from the user's menu, the *Location* view is loaded with the retrieved position as long as the current user account has permission to access the monitored user location data (Figure 37a). A pin added to a map represents the target user's current location with an overlay label. This label shows the username and the city or even more accurate address information, depending on how much the position is close to a street.

In order to show the correct address for the location displayed in the map, it was used a common technique called *reverse geocoding* through the use of the MapKit framework `MKReverseGeocoder` class. As the name implies, it allows getting an address, area and other useful information (zip code, state, etc.) based on the coordinates of the location itself.

The icon just at the right position on the label is interactive, thus the user may tap it in order to show the *Position Info* view which displays detailed information of the position data acquired (Figure 37b). Such details include coordinates, current speed, direction, horizontal accuracy of the retrieved coordinates and the timestamp which allows knowing how fresh the monitored device's location data is.

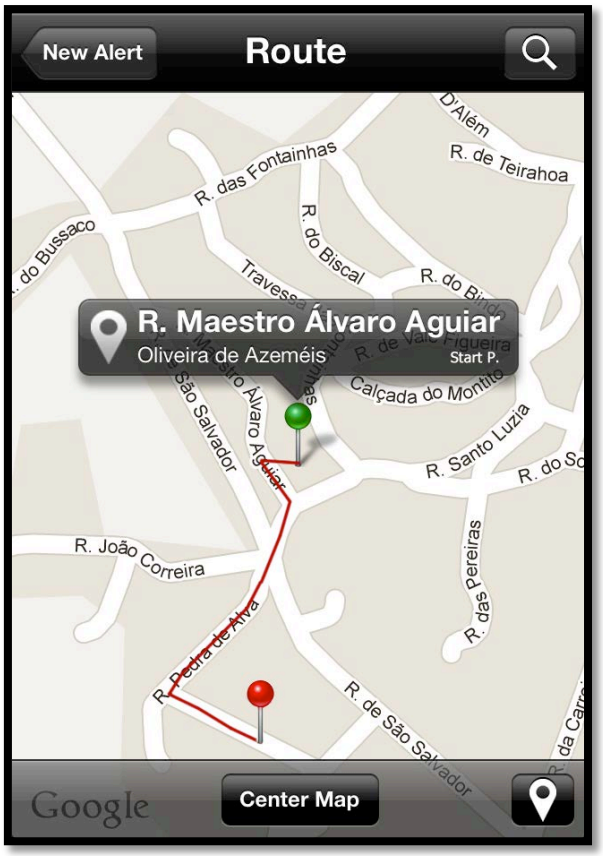

Figure 38 – Route view

The *Route* view (Figure 38) is similar to the *Location* view in almost every aspect. The two differences are in the number of pins displayed and in the line that links the ends. The green pin indicates the position where the user has begun to be tracked whereas the red one specifies the last position where the user has been, while the self-tracking mode was active. The path the monitored user followed is shown, as a red line, between the start and the end positions.

The MapKit framework in the iOS SDK does not support path overlays to be drawn over the map. Thus, it was necessary to implement this functionality. It is received a set of pairs of coordinates along with other information like timestamps for each of them and they are added to an `NSDictionary`, using a proper structure. Those items, which are sorted by timestamp, are then passed to a method that draws a line between every two points, forming the path the target user travelled. This path's accuracy varies depending on the time interval defined on the self-tracking settings menu to periodically store the user's current position.

Other kind of information like timestamps of the position is available just by tapping the label over the pins.

Figure 39 – a) Alerts Management view; b) New Alert view

The alerts management is another feature available from the user's menu. In the underlying view (Figure 39a), all the defined alerts for a certain user are shown, including their title as well as the time valid for their triggering. From this same view, it is possible to quickly enable and disable any alert, through the graphical switches available for each one of them. Every time a change is made to any of the alerts status, both the internal and server databases are updated and an update notification is sent to the mobile device of the user which the alert refers to. This will allow it to know that changes were made to the alerts that it is attending to, thus its internal database must be updated too in order to reflect the changes made.

It is in the *Alerts Management* view (Figure 39a) that new alerts can be defined for a user. At the top right corner there is a "plus" icon which will raise a *New Alert* view (Figure 39b). The alerts are very customizable, so this view offers a vast range of options in order to every alert meet the users' needs. In addition to the alert title, the user may define triggering time settings, including the time the monitoring begins and ends as well as how often it should be repeated or even not repeated at all. The repeating options that can be defined are grouped in a daily, weekly, monthly and yearly basis.

Regarding the monitoring settings, there are some more options that can be defined. In addition to the target position setting, comprising its latitude and longitude values, there are the radius (which can be expressed in meters or kilometers), minimal accuracy allowed

(selected from a predefined range of values), and trigger condition. The trigger condition defines if the alert should be triggered whenever the user reaches the target location within the defined radius, or on the other hand, whenever it leaves the target location area delimited by the radius value.

The minimal accuracy allowed is useful to delineate the margin of error for the retrieved coordinates from other users. There are plenty of users that work in areas where the location acquiring may not be as accurate as if they were in an open field. This lack of accuracy may lead to the trigger of unexpected and inaccurate alerts. By defining a minimal accuracy value, according to the environment characteristics of the target location, the alerts will be triggered with more reliability.

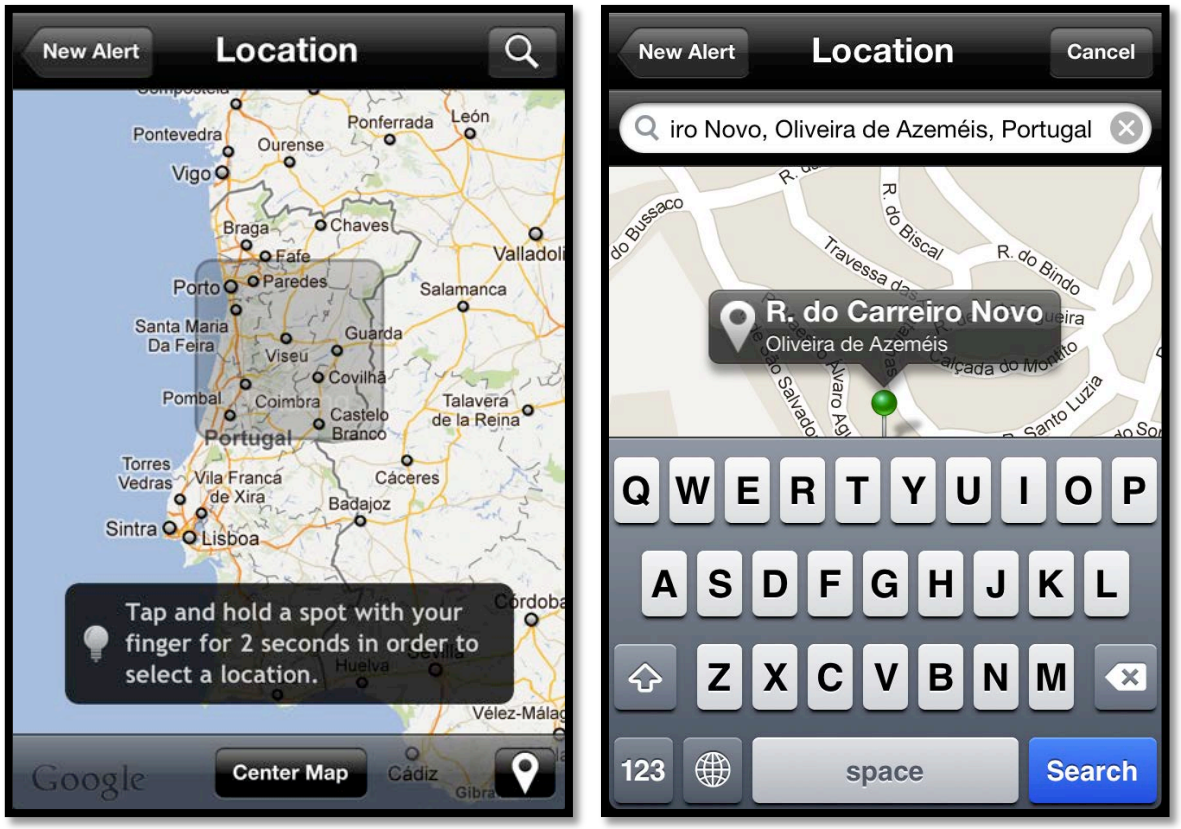

Figure 40 – Location view featuring Tap Selection and Address Search

When the location option in the *New Alert* view (Figure 39b) is tapped by the user, the *Location* view (Figure 40) is loaded but with a different behaviour as if it would be launched by the user's menu actions (Figure 36). In this mode, the view presents a searchable map with tapping support for choosing a location. Thus, in order to choose a location the user may opt by select the upper right magnifier button and entering the location address, or alternatively, it may want to just navigate through the map with finger gestures and then hold the desired position for just about two seconds. When doing this, a pin is dropped to the map assuring the user that the location is selected.

Another addressed situation is the one in which the user may want to just select his current location. This may be done by just tapping the lower right button.

The location search by address is not provided by any of the iOS SDK APIs. Thus, this feature was implemented using the Google Maps API and the TouchJSON framework. The query string is built as follows:

```
1. NSMutableString *queryString = [NSMutableString
2. stringWithFormat:@"http://maps.google.com/maps/geo?q=%@?outpu
3. t=json", inAddress];
```

The **inAddress** property is the text entered in the search text field. In the query string it is also possible to see that the output results will always be in JSON format. In the controller for the *Location* view there is the delegate method `connection:didReceiveData:` that handles the JSON formatted data received from Google servers. The output is then *deserialized* using the `CJSONDeserializer` class of the TouchJSON framework. Whenever a result for the entered address is not found, an *AlertView* is raised to alert the user for the fact. Otherwise, a pin representing the searched location is automatically dropped in the map.

Figure 41 – Settings view

The *Settings* view is accessible through UITabBar component (Figure 41). This view provides the user with options that helps to control both privacy and battery consumption. The global privacy settings allow full control of location and tracking data transmission to other users. Both these two options have higher priority over individual settings (refer Figure 36) when disabled and less priority when enabled. This means that, for instance, if the

location allowance is disabled in this view, any user's individual location allowance setting is not considered. Otherwise, the individual location allowance for a certain user will determine if it has the ability to either receive or not, this mobile device's location.

The self-tracking options allow defining its state and its updating interval. This interval is selected from a predefined list of values that goes up to 3 hours. The interval is used to define when the GPS receiver should be reactivated in order to aid in power saving. Hence, the lower this interval is, the higher the power consumption will be as the GPS receiver will reactivate more times in the same time space. The logout button is also present in this view, at the upper right corner, and it is the only way to return back to the *Start Menu* view, when the automated login option is enabled.

5.2. XMPP Communication

As mentioned in section 4.2, the elected communication protocol was the XMPP. Among the several reasons to use this protocol was the availability of an open-source XMPP framework, based in Objective-C, with full support for the iOS platform. This XMPP framework is divided into two parts, the XMPP Core package and the Extensions one (Figure 42).

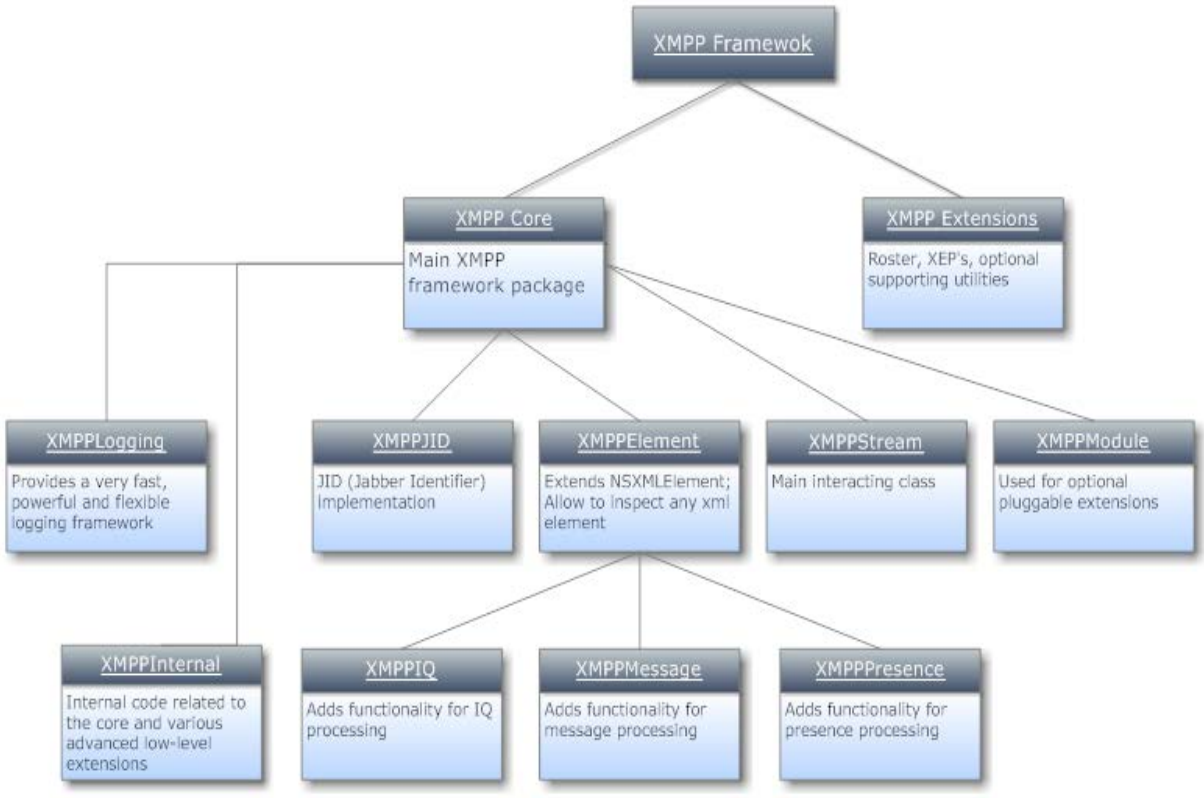

Figure 42 – XMPP Framework Architecture

The whole XMPP implementation attends to its RFC 3920 specification (IETF, 2004). Although it is a fully-featured framework, there were some simple features that were not supported like username management. However, the reason is simple. XMPP is not strictly a

chat protocol as many people uses to refer as to, but it is a generic protocol which can be used for several purposes. Thus, there were made some attempts to overcome the lack of this feature.

The first one was to change the message format of the protocol in the framework itself to include the username attribute. It turns out that it was not a wise choice. First of all, the modification of the message format automatically would imply that the server-side should be able to recognize all those extra elements added, thus demanding the change of the server application code and consequently requiring it to be open-source. In addition to that, the change of the message format would prevent interoperability with a set of server applications.

This approach was attempted using the *Openfire* XMPP server, a Java-based and open-source platform. This means that, the application would be dependable on an *Openfire* server, and not just any but our own custom *Openfire* server. Thanks to the decision of not following that path, nowadays the application is running over an *eJabberd* server which is much faster and lightweight to the system than *Openfire* will ever be (due to be *Java*-based), in addition to still being interoperable with any other XMPP server that follows the specification defined by the RFC 3920. Hence, the final approach consisted in keeping all the usernames in an external database used to store any other user related information.

As it happens with any data downloaded to the device, the usernames are stored in cache and whenever another user changes its username, a XMPP message is broadcasted to all his roster users, which in turn update their internal cached data. This broadcasting system allows the application to not be constantly monitoring the server database for any changes thus saving bandwidth. The same happens to the user that is broadcasting the message as it only needs to send one single message to the server which only then is distributed to all the other peers.

Every XMPP communication between two devices uses custom XML messages with a proper format for each of the required operations. These messages are created and handled by the communication module. These operations are grouped in requests and responses. Both the location and tracking data requests sent to another device use the same message format:

```
1. <message
2.       from="sourceJabberID@Domain/Resource"
3.       to="targetJabberID@Domain/Resource"
4.       type="requestType"
5. />
```

In order to a message reach the proper target device, the “to” attribute should be formed by the unique Jabber ID along with the XMPP “*domain*” and “*resource*” strings. While the *domain* property requirement may seem obvious, the *resource* one may be not as it is not so usual.

The *resources* are a Jabber feature that allows multiple clients to use the same user account concurrently. For instance, a user may have a resource “Home” at its XMPP client at home and a resource “iPhone” at its XMPP client in its mobile phone. In this situation, other Jabber users can choose whether to send messages to this user iPhone or its home client. The account is the same, the user is logged in both places but the resources to where the messages are delivered to are different.

The resources feature also allows setting different priorities for each of them. This allows, for instance, that a user defines its home client to have higher priority than its mobile device client. By setting priorities this way, whenever the home client gets inactive and set its availability state to “Away”, the mobile device client sets immediately its availability state to “Available”. Then, when the user returns back to its home client and its state is set again from “Away” to “Available”, the mobile device client gets back to its “Away” state because its low priority over the home client.

Unlike the requests, the responses’ elements may vary depending on the type of data received. A location response is sent using the following structure (example values are used to demonstrate the data types of each element):

```
<message
     from="sourceJabberID@Domain/Resource"
     to="targetJabberID@Domain/Resource"
     type="locationResponse">
      <location>
          <latitude>40.895625</latitude>
          <longitude>-8.416832</longitude>
          <speed>63.000000</speed>
          <horizontalAccuracy>162.787995</horizontalAccuracy>
          <course>18.000000</course>
          <timestamp>2011-10-19 14:28:04 +0000</timestamp>
      </location>
</message>
```

The larger response messages may be triggered to attend to tracking data requests depending on the number of stored coordinates the target device may have. However, the application was designed to have a limit defined in its configuration file to establish the number of tracking data items that may be stored in order to improve performance and keep the device from sending very old data.

There are other simple responses. Ones may be used to notify the target user for alert settings changes that the source device made in the online database, so that the target device doesn’t need to query the database periodically to check if any change was made (thus saving battery life). Others may be used to answer to unauthorized requests, the ones that the source user does not have permission to make. This kind of messages adds another attribute, the *operation* attribute, because the type of the response is defined as `unauthorizedOperation` and the source device of the request needs to know what type of operation is being answered to.

5.3. Application Configuration and Behavior

The application was developed keeping always in mind that it may serve several purposes in the future, as a framework to exchange hardware data between electronic devices. In order to allow other developers to adapt the application to meet their needs, it was created a configuration file that handles the most relevant settings which are required to be set, prior to the building process. In Table 5 there is a description of all the customizable parameters.

Table 5 – Application Configuration Parameters

Parameter	Type	Description
XMPP Protocol Parameters		
XMPP_SERVER_ADDRESS	String	Address for the XMPP server.
XMPP_SERVER_PORT	Integer	Port for the XMPP server.
XMPP_SERVER_DOMAIN	String	Domain name XMPP server.
XMPP_RESOURCE_NAME	String	Resource name for XMPP client.
XMPP_ALLOW_SELF_SIGNED_CERTIFICATES	Boolean	Defines if self-signed certificates are allowed.
XMPP_ATTEMPT_SECURE_CONNECTION	Boolean	Defines if a secure connection via SSL/TLS should be attempted first.
XMPP_ALLOW_SECURED_CONNECTION_ONLY	Boolean	Defines if the application should only connect through a secure SSL/TLS connection.
User Database Parameters		
USERDB_SERVER_ADDRESS	String	Address for the user database server.
USERDB_SERVER_PASSWORD	String	Password for the user database server.
Core Location Parameters		
MINIMUM_INTERVAL_BETWEEN_UPDATES	Double	Defines how recent is an event in order to turn off location updates to save battery (seconds).
IGNORE_CLOSEST_ALERT_FOR_ETA_CALC_IF_DISABLED	Boolean	Defines if the closest alerts which are disabled should be ignored for GPS reactivation time calculation.
REQUESTS_QUEUE_DISPATCH_INTERVAL	Double	Time interval between the processing of all the requests (seconds).
MAX_LOCATION_ENTRIES	Integer	Maximum number of location entries which are allowed to be stored in the internal database.
USER_LOCATION_REQUEST_TIMEOUT	Double	User's location data request timeout (seconds).
USER_TRACKING_REQUEST_TIMEOUT	Double	User's tracking data request timeout (seconds).
Alerts Parameters		
DELAY_BETWEEN_ALERT_RESEND	Double	Time delay within an alert cannot be resent to the same user (seconds).
MAX_ALERT_NOTIFICATIONS	Integer	Maximum number of received alert notifications which are allowed to be stored.

In addition to all the trivial server connection parameters, there are others that should be changed according to the purpose of the application. For instance, if there is a need for

monitor an extensive location data timeline for the users, there are a few parameters that should be adjusted. Among them, there is the MINIMUM_INTERVAL_BETWEEN_UPDATES parameter that should be lowered to the minimum time possible, and the MAX_LOCATION_ENTRIES that should be increased to a higher value (in order to store more location information over the time), considering the “factory defaults” the application has.

It is important to choose carefully the values for most of these parameters, mainly for those which belong to the core location parameters section, since they may have a strong impact, either in terms of performance or regarding used bandwidth.

5.4. Tests and Results

During development, a real device (iPhone 4) and a simulator were used in order to test the main functionalities of the application. This testing environment has limited the implementation of some hardware-accessing features.

Despite the fact the simulator performs well for applications that not requires access to many hardware capabilities, it turns out that it does not support some hardware features like Bluetooth, accelerometer and camera. Due to this fact, Bluetooth pairing mechanism could not be fully implemented.

However, in the final testing phase it was possible to deploy the application in five iPhones (two iPhone 3GS and three iPhone 4). The application performed very well even in 3G signal absence areas (where EDGE technology is used instead).

Considering the application running on EDGE, it was noticed that it can take several seconds in order to request the traveled path of a monitored device. In the performed tests, it takes an average of 7 seconds to send 10 pair of coordinates (which form a path) against the 2 seconds when running under 3G coverage.

Regarding the positioning accuracy, it was concluded that iPhone 4’s GPS receiver is more accurate and takes less time to first fix than the one present in iPhone 3GS. This means that location tracking requests sent to iPhone 3GS users may take more time to be answered.

The event-based notification system performed well in almost every tested situation. Initially, it revealed some issues for “leaving area” events. These issues were caused by the weakness of GPS signal when using a low accuracy value for the acquired location data. The following testing scenario was performed.

A monitored device was configured to send alert notifications whenever it leaves the Computer Science Department of Porto Superior Institute of Engineering (Porto, Portugal). The remaining alert options were set as follows:

- **Start Time:** 2011/02/07 10:00 AM
- **End Time:** 2011/02/07 10:30 AM
- **Location:** 41,177839, -8,607746
- **Repeat:** Every week
- **Radius:** 50 meters
- **Triggers when:** Leaving Area

It is important to notice that, whenever the accuracy of the acquired position drops, its representative value (distance in meters) increases. Therefore, it should be interpreted as a margin of error.

Attending to the scenario, whenever the GPS signal was weak and the positioning accuracy went down, the alert was triggered. It was happening even though the user was inside the radius delimited area for the location which was initially set for the alert. This is due to the radius being less than the predefined minimal accuracy allowed (radius = 50 meters; default minimal accuracy = 100 meters).

The issue was solved by providing the user with the ability to choose the minimal accuracy to be attended to and by ensuring that it is not allowed to choose a radius less than the minimal accuracy selected. After this new implementation, the testing results were the expected since the application did not sent false alert notifications any more.

It should be noted that in every test performed, the schedule and all the repetition settings were always attended and no alerts were sent, out of that time. In order to test the repetition settings effectiveness, the mobile device’s calendar was adulterated.

CHAPTER 6

BUSINESS MODEL

This chapter will take the reader through a set of market related aspects on how to monetize the application efficiently. The initial investment is approached as well as the payback times for all the spending. It is also discussed the available business models and which ones would bring more profits when applied to this application. A comparison between rival applications' features is made. It is presented an annual profit projection for each business model approached. Promotion related aspects are also referred.

The mobile market has grown, and continues to grow, due to the interest shown by entrepreneurs and companies for the amazing and crowded mobile platforms that have become *mainstream* in the most reputable mobile devices' brands (BuySellAds, 2011; Lookout, 2011; ABI Research, 2011).

Apart from Apple that, despite the overall success it had and continues to have, only deploy its iOS platform into their proprietary devices, Android and Windows Phone are two platforms that have shown a steady advance in the mobile market field, being adopted by a high number of brands and devices. This higher variety when comparing with Apple devices, allows the applications developed for those platforms to reach a greater set of market niches, as there are low priced devices which meets the budget of a greater number of customers.

While some years ago it did may seem easy to get going with the mobile applications market, today it takes more time, additional visibility and a greater idea to really make feasible profits from an application. Due to the extremely congested market, full of applications belonging to the same category of the one we are developing, it takes extra steps to make it go through all the rival applications to win a place on the podium. Innovative features or services, greater and easy-to-use interfaces and lowered-priced solutions, regarding the competition, may be the path to follow.

The application developed in this work belongs to the location-aware category. It has been proven already that the location-aware economy is one of the ones growing at a faster rate over the past few years and that it is expected to grow over the ones to come. To support this assertion, one may refer to the amount of cash received as of the last year (reporting the past three years of revenues) by the companies that make only location-based applications. In just 67 deals, earnings were of $656 million (Spiro, 2010).

6.1. Spending Assumption

Considering the XMPP/Jabber server hosting, there are two solutions with different implications at first sight: either to consider renting a server or to set up our own. In terms of costs, the last one seems to be the more feasible. It also allows full access to the machine, thus easing the maintenance and providing faster service reestablishment in case of failure.

On the other hand, the scalability factor has an enormous weight on the decision. Whereas running our own server may be a good way to start collecting costumers, it can become a

serious problem in medium or long term (considering the application becomes successful) when the server gets unable to handle a high amount of connections. In addition to that, the scalability regarding bandwidth needs also applies. If we consider also the need of a secured connection, it turns out that the final expense does not justify the lack of scalability.

Some commercial XMPP/Jabber and PHP/MySQL server hosting companies were taken as reference (refer Table 6) in order to get to know an average monthly cost that one should be willing to pay (JabberHosts, 2011). The main deference between the different packages offered by a Jabber hosting company is the number of supported Jabber accounts. Since the number of accounts supported is one of the main concerns, there are only considered the plans that support the highest number per each company. Two companies were selected among many others in order to get some reference values.

Table 6 – XMPP/Jabber and PHP/MySQL Hosting Plans

Company	Main Features	Price
ITONIX Jabber Hosting[1]	Jabber IM server including up to 1,000 user accounts; 1,000 GB monthly instant messaging bandwidth.	$240/year
	Additional Secure Digital Certificate.	$150/year
IPower[2]	Unlimited bandwidth and disk space; CGI-BIN + PHP 4 & 5 Support; support up to 25 MySQL Databases.	$5.00/month

Table 7 depicts the total costs regarding the ones involved with the application development and maintenance. Since in the application's early stage, we are only seeking to know how the targeted audience reacts to its appearance on the market, there are some expensive features that are not yet considered. The purchase of secure digital certificates to support secure connections is an example of such features.

In an experimental phase, the server will support 1000 accounts which should be constantly monitored in order to know when to rent another one. There are also a few assumptions that were taken regarding the user data server. We assume that, at the beginning, there is no need for a dedicated server, so a shared one could be rented.

Over time, if the application is successful and gets too many user connections or the database gets extensive (which is unlikely due to the very limited amount of information that is stored about users), choosing a dedicated server should be considered. Since the application was developed under the scope of this work and not by third parties, no software development costs are added.

[1] Detailed hosting plan at http://itonix.com/services.
[2] Detailed hosting plan at http://www.ipower.com.

Table 7 – Total Costs to Get Started

Costs	
Apple Developer Program	$99/year
XMPP/Jabber Hosting Service	$240/year
PHP Hosting with MySQL Support	$5,00/month
Domain name (.com)	$9,00/year
	$353

Taking as reference average values from multiple hosting services and domain name providers, the total cost needed to set the application running and suitable to be commercialized is $353. The domain name was considered, mainly for maintenance purposes, since it would facilitate hosting switching with minimal service interruption time (no dedicated IP addresses are attached to the application, thus no software updates need to be deployed to the costumers reflecting IP address changes). Following the same values, the 11 months left in order to complete a year would cost $55, the cost related to the user data server hosting. Annually, the cost to keep the application running would be $408/year.

The Apple developer program should always be considered in the yearly costs. Even after the application is submitted on App Store and the developer does not intend to update it anymore, the developer program should be renewed. Else, the application will no longer be present in the App Store and the ability to submit updates or other applications is lost (Apple Inc., 2011c).

6.2. Revenue Projection

There are many ways to monetize the application. Some may be taken as a way to start distributing the app and others may be seen as the next step in the exploitation field.

Distributing this application for free can be the best way to give it some visibility because customers are usually afraid to buy something which they think there are already alternatives to. The users may be able to explore the application with no commitment in order to get to know what really distinguishes this one from the remaining applications within its category. This, however, does not mean that one cannot make profit of it. Since the intention is to earn cash from this application, the advertising system is the way to go.

As this application is iOS based, thus directed to Apple devices' costumers, the Apple's iAd Network is the best way to monetize the application at an initial stage. Apple sells and hosts the ads, while 60% of the revenues go directly to the developer. In addition to the fact that it is easily implemented in iOS applications through the provided SDK, it is also a really optimized advertising solution, since the ads provided are interactive and do not need the user to get out of the running application (a critical factor, when running games, for instance).

The iAd technology relies on a set of standard metrics that allows the advertiser to measure its adopted market strategy and adapt it to the new knowledge discovered about its audience (Table 8).

Table 8 – Standard Apple's iAd Network Metrics

Metric	Description
Impressions	Equals to the number of exposures of an ad or commercial to people or households in your audience.
Taps and tap-through-rate	Is the ratio of the number of times an ad is tapped, divided by the number of times an ad is viewed (or impressions).
Unique visits	Represents the number of unduplicated (counted only once) visitors the subject of your ad over the course of a specified time period.
Average time spent	The amount of time that a visitor spends on an advertisement subject.
Views and views per visit	Number of total views of the ad
Fill rate	Percentage of the time an ad is displayed.
Interactions (videos viewed, games played etc.)	The number of times a media is opened and viewed from the application.
Conversions and downloads	Many advertisements may be to promote other applications in App Store. The number of downloads and conversions to another app is thus considered.

Regarding the paid model it may be profitable at first but over the time, it will be impossible to maintain the service running due to the fact that there are regular extra costs with servers which are fixed for each month/year. The average price for an application is between $3.99 and $4.99, which is a price that almost anyone is willing to pay for it, if it is useful and meet their needs.

The price, however, is sometimes target of bad reviews, and not allowing the user to first trial and then buy, is a way to foster this kind of situations, as he may expect more than the application offers. If the application passes through a cycle where it starts by being free (even for a short period of time which is enough to get some good reviews) and eventually ends priced, most of the target users of the application will eventually pay for it without double thinking.

Another alternative should be by releasing lite (limited functionality) and pro (full functionality) versions of the application. Where the first one attends to the advertising system, the pro version is paid but offers all the functionality without any kind of restriction to the user. In this way, it is gathered the best of both worlds, considering that there is no need for too much effort in order to adapt the same developed application to this two different situations.

The subscription model is another way to get profits from this application. By charging the customers a recurring fee (monthly, for instance) for using some features/services, the revenues may be enough to cover the underlying server costs and the remaining is all profit. This application has a set of features that can be completely restricted or partially limited.

Regarding limitations, there are some important parameters in this application that can be set as premium features. The most important one is the number of roster items allowed. The free package allows the user to pair with up to another 5 users. In order to pair with more, the user has to buy a monthly premium subscription allowing that. The same goes to other features like the ability to send alerts, number of alert monitoring items allowed to be concurrently enabled, number of tracking times per day, etc. The subscription can be acquired per feature or as a full-package, which unlocks every limitation the application has.

In-app purchases could be an alternative to subscriptions, since users tend to prefer life-time acquisitions, even though for an increased price. This model alone, however, may not be able to cover the server costs in medium or long-term, as the subscription system does. Considering a mix between in-app purchases and subscriptions would be ideal. This can be done through a credit system.

A credit system should give the consumer the freedom to choose when to extend the features without being forced to pay, regularly, a fixed fee once a month or whatever the billing period is. Each credit represents a certain amount in cash. The user realizes that it needs 10 more slots in its roster, and that each slot costs, for instance, 2 credits. The user then buys a 30 credit package and uses them for whatever it wants, which in this case would be used to increase the roster limit by 10 slots and still have left 10 credits. These 10 credits may be spent in other application's features which have a credit cost associated.

This approach reveals itself very flexible as it also allows the developer to distribute free credits, in order to get users testing the application features without being forced to spend any cent. This credit system may even be combined with the advertising model, extending the chances of profiting. Not all people likes to see advertising while interacting with their applications, so the customers who feel annoyed by the presented advertisements (ads) may eventually use credits to extinguish them, which is another great way to maintain various ways to profit in one single application.

6.3. Payback Times

For each business model applied, the time it takes to carry back the initial investment differs. This section explains what variables are involved in payback time calculations for each business model and presents some estimated times.

6.3.1. Advertising Business Model

Considering the application distribution using the advertising business model, the payback time can be calculated by answering the following questions:

1. How many times a user will launch this application every day?
2. How many average advertisements users will see every time they use this application?
3. What would be the estimated cost per thousand ads impressions (CPM), which an advertiser would pay for?

Having these questions answered will give us the required quantity of downloads to cover the spending of the initial investment within 1 month. Let's then consider some realistic answers:

1. The user will launch the application 3 times a day (L=3).
2. Users will see an average of 4 advertisements (impressions) whenever they use this application (I=4).
3. The estimated CPM is $7.90.

$$\textbf{\textit{Required Downloads}} = \frac{\text{Spendings x } 1000}{\text{CPM x L x I}}$$

$$= \frac{353 \text{ x } 1000}{7{,}90 \text{ x } 3 \text{ x } 4}$$

$$\cong 3724$$

According to the result, when following the advertising business model, there are needed 3724 downloads approximately in order to cover the initial investment within 1 month.

Despite not being considered for the calculations due to its high randomness, there is an important factor that is usually associated with the CPM. It is the Fill Rate (Table 8). The Fill Rate almost never is 100%. For example, if the $7.90 ad only fills 10% of the time the application is launched, we are going to make only $0.79 per thousand ads impressions (10% of $7.90) (JQMedia, 2010).

6.3.2. Paid Business Model

Considering the paid business model, the payback time can be calculated through the answer to these questions:

1. How much will cost the application in the App Store?
2. What is the download average per month for each application?

The first question should be answered based on the developer goals and expectations but it is also important to give it an attractive price at this initial stage. The second question is a little bit difficult to answer as it varies too much considering the application and the month.

1. The application will cost $1.50.
2. Each application is downloaded 150 times per month worldwide.

If the application costs $1.50 and considering that Apple takes 30% of the profit, the earnings will be of $1.05 per application.

In order to cover the initial investment, there will be needed 235 users to buy the application. Assuming that each application is downloaded around 150 times per month, the payback time will be little more than one month and a half approximately.

6.3.3. Subscription Business Model

Considering the subscription business model (recurring to in-app purchases) adapted to the credit system, there are also some questions to answer:

1. How much money will cost 1 credit?
2. What will be the minimum amount of credits which will be allowed to be acquired?
3. How many features will use credits to be unlocked for life and how many will spend credits per use rate?
4. What is the download average per month for each application?
5. The application will also be supported by advertising with an option to exclude it by using credits? How many credits will it cost?

Assuming that the answers to those questions are:

1. $0.02 is the price for just 1 credit.
2. The credit packages available are 50 credits for $1 and 200 for $3.
3. The location and tracking features use 1 credit per use; 10 more slots added to the roster costs 10 credits; alert monitoring features are free.
4. Each application is downloaded 150 times per month worldwide.
5. Yes. The advertising removal option will cost 50 credits.

We assume an average of 150 downloads per month and that only 60% of the users will find it useful for their needs and buy credit packages. From those 60% (90 users) that bough credit packages, we assume that they will buy the lowest credit package since they do not know the application, it is profited $90.

Yet considering the same percentage of users (60%), 25% finds the application useful in their daily life and makes intense use of it, thus they buy the higher credit package available. So, $67.5 is added to the $90 resulting in a total of $157.5.

Those who also make intense use of the application also use credits in order to remove advertisements from the application. At 1$ each ad removal, the end result is $180.

By resorting to this business model only, the developer will take almost 2 months to recover its initial investment. However, this model working along with the advertising one will reach higher profits, thus decreasing that time.

6.4. Profits Over Time

This section discusses which business models are more profitable considering the application's purposes and its target audience.

6.4.1. Advertising Business Model

Considering the assumptions mentioned in section 6.3.1, it was concluded that it is required 3724 downloads in order to cover the initial investment within 1 month. However, this number is relatively high considering that the application is new in the market. To make this value more realistic, we are going to assume the required number of downloads to cover the initial investment in 6 months.

$$\boldsymbol{Required\ Downloads} = \frac{3724}{6} = 621$$

Assuming that the application has an average of 621 downloads per month, the user will launch it 3 times a day and it will see an average of 4 advertisements per each time he uses the application:

$$\boldsymbol{Total\ Ad\ Impressions} \quad = \quad 621 \text{ x } 3 \text{ x } 4 \quad = \quad 7452$$

Still considering the same estimated CPM as in section 6.3.1 (CPM = $7.90):

$$\boldsymbol{Profits\ per\ Month} \quad = \quad 7452 \text{ x } \frac{\$7.90}{1000} = \$58.87$$

$$\boldsymbol{Profits\ per\ Year} \quad = \quad \$58.87 \text{ x } 12 \ = \ \$706.45$$

Subtracting the yearly expenses ($408.00) to the profits per year ($706.45), it results in annually earnings of $298.45.

Table 9 illustrates the profits for the first five years of commercialization. Assuming that the application becomes better known over time, it is considered that the number of application downloads per month increases an average of 30 percent each year until the third one. After that, the amount of downloads increases less 10 per cent each year.

Table 9 – Annual Profits Projection for Advertising Business Model

Year	Downloads (month)	Total Profits (year)	Gross Profits (year)
1	621	$706.45	$298.45
2	807 (+30%)	$918.04	$510.04
3	1049 (+30%)	$1193.34	$785.34
4	1259 (+20%)	$1432.24	$1024.24
5	1385 (+10%)	$1575.58	$1167.58

6.4.2. Paid Business Model

Regarding the paid business model, the profits coming from its employment varies according to the application price but it essentially depends on the number of downloads per month. In section 6.3, it was assumed an average of 150 downloads per month in order to calculate the payback time. It was also considered that the application costs $1.50, resulting in only $1.05 after Apple takes 30 per cent of the profit. Table 10 shows the annual profits projection regarding this business model.

Table 10 – Annual Profits Projection for Paid Business Model

Year	Downloads (month)	Total Profits (year)	Gross Profits (year)
1	150	$1890.00	$1482.00
2	195 (+30%)	$2457.00	$2049.00
3	254 (+30%)	$3200.40	$2792.40
4	305 (+20%)	$3843.00	$3435.00
5	336 (+10%)	$4233.60	$3825.60

This business model compared to the advertising business model is more profitable. In addition, if we apply this model's base monthly downloads in the first year (150) to the advertising business model, we notice that its profits are not even enough to cover the annual expenses ($170.64).

6.4.3. Subscription Business Model

In order to provide an annual projection for the subscription model, the same assumptions mentioned in section 6.3.3 are followed.

Table 11 - Annual Profits Projection for Subscription Business Model

Year	Downloads (month)	Total Profits (year)	Gross Profits (year)
1	150	$180.00	-$228.00
2	195 (+30%)	$234.00	-$174.00
3	254 (+30%)	$304.80	-$103.20
4	305 (+20%)	$366.00	-$42.00
5	336 (+10%)	$403.20	-$4.80
6	353 (+5%)	$423.60	$15.00

By examining Table 11, we notice that only after the sixth year the application would be profitable by a small margin. Just like the advertising business model (section 6.4.1), this subscription based model requires higher number of application downloads in order to be profitable in the first commercialization times.

6.4.4. Mixed Advertising and Subscription Business Model

As it was said in section 6.3.3, both the advertising and subscription business models working together allow us to reach higher profits. Table 12 considers both business models

gross profits. The advertising model values already assume those users which pay for advertising removal, thus the values are lower than in the projection made in section 6.4.1.

Table 12 - Annual Profits Projection for Mixed A+S Business Model

Year	Downloads (month)	Advertising Model Gross Profits (year) *(Considering Users Which Paid To Remove Advertisements)*	Subscription Model Gross Profits (year)	Total Profits From Both Models
1	621	$143.20	$337.20	$480.40
2	807 (+30%)	$308.29	$560.40	$868.69
3	1049 (+30%)	$523.09	$850.80	$1373.89
4	1259 (+20%)	$709.49	$1102.80	$1812.29
5	1385 (+10%)	$821.33	$1254.00	$2075.33

Although it is not as profitable as the paid business model and it requires more downloads in order to be feasible, this mix between the advertising and subscription business models is the most rewarding solution, next to the paid model.

6.5. Rival Applications

The mobile market is full of location-aware applications. They serve several different purposes, ones better than others. In order to the application succeed, it is important to analyze the most competitive applications that provide the same features as *EyeGotcha* does. This section approaches the most successful applications which are aimed to the same purposes of the developed application. Thereby, it is possible to know what *EyeGotcha* can offer to the mobile audience which other applications of the same segment cannot.

6.5.1. Find My Friends

Find My Friends is an application for the iOS platform that allows locating people. Users are able to share their current location with a group of persons. This group of people is managed through requests sent to them through the iCloud, the Apple's Cloud Service (iCloud, 2011).

The exclusive identifier for each user is their email address and it is possible to invite up to four persons at a time, by just entering the email address or by searching directly for a contact in the iPhone's contact list. For each contact added, the user can set a privacy rule which defines if the contact can actually locate him. When a contact is located, its position is shown on the map and the user has shortcuts to some iOS actions, related with contact conversation. This allows a user to rapidly call a contact through *Phone* or *FaceTime* iOS stock applications or to just send him an email quickly.

The application also allows privacy options to be password protected. This is an interesting feature for parental controlling purposes. No location tracking history (traveled path) feature is present and it is only available to iOS 5 and versions above.

6.5.2. Google Latitude

Google Latitude (Google, 2011) is a multi-platform (iOS, Android, RIM, Symbian S60 and Windows Mobile) tool with location tracking features. Users are paired, based on their Google's email address, using the usual request approving/rejecting system. In the monitored users' list, their current distance from the monitoring user is displayed and they are sorted by the lowest distance, so that the monitoring user can know who is closer to him. It supports detailed privacy control in which the monitoring user can define what location information can be shared (city-level data only, for instance).

6.5.3. Glympse

Glympse is another location tracking but it works on a different way than usual. Rather than being the monitoring user to ask for the monitored user's location, it happens on contrary instead. Thus, it is the user itself that may want to share his location with others, not the one who asks anybody else for their location. Therefore, the user selects numbers or emails of people he wants them to know his location and also selects the time for how long the application will be sending his position to them (indefinitely is also a possible option).

When the user proceeds in sending location, a link to the *Glympse* site is sent to all the people the user selected. From here, those persons can track the user's location even without a mobile phone since it only requires a web browser. The path a user travels can also be drawn in the map with high accuracy.

Table 13 – EyeGotcha Competitors' Strengths and Weaknesses

	Strengths	Weaknesses
Find My Friends	• Ability to invite up to 4 persons at a time; • May be configured to be always running even after phone restarts as it uses iOS private features; • Allows calling or mailing a monitored user quickly through shortcuts.	• Does not show the path a monitored user travels; • Does not have the ability to send event-based alerts.
Google Latitude	• Multi-platform. • Allows calling or mailing a monitored user quickly through shortcuts; • Allows revealing only certain details about the user's location (i.e.: city,	• Only Google email accounts can be used for sign in and pairing purposes; • Does not show the path a monitored user travels; only does for the monitoring

	etc.).	device; • Does not have the ability to send event-based alerts.
Glympse	• Allows sharing user location with someone who does not have a phone; • Retrieved location data is more accurate as location updates are continuously sent.	• Has to be the other party (monitored user) to initiate the location sharing with the monitoring user; • Updates are sent constantly, thus draining the battery life. • Does not have the ability to send event-based alerts.

Table 13 highlights the strong and the weak points of applications that directly compete with *EyeGotcha*. It turns out that *EyeGotcha* misses some useful features like location-detailed privacy options, password protection and interruptibility (the application should not be terminated by any user). It should also be noted that all the mentioned applications are both free. However, it has some key features that most of the considered applications misses: the ability to track monitored and monitoring users' traveled path, the ability to send event-based alerts and also good energy and bandwidth saving mechanisms.

There are other applications in the market which includes the traveled path feature like *GeoTwitts* (GeoTwitts, 2011) but none of those which were analyzed offers event-based alerts. The closest feature found which is similar to that, is the one in which a monitoring user can get alert notifications whenever a monitored user comes close to him by a certain distance. *Friendjectory* (Friendjectory, 2011) is an application that has that feature. However, it is not as complete and flexible as the event-based alert system which *EyeGotcha* can offer.

6.6. Promotion Strategy

The social networks like Facebook and Twitter are, with no doubt, the best way to advertise this application. Promoting the application on blogs or writing reviews about it may help to disseminate it over the general Internet users. As the cost for the user data server is already beheld, it is feasible to create a good-looking site without spending any more money, explaining in detail what the application is and does, and who the target audience is.

In order to get this application promoted it does not need to be released to the market just yet. Contacting review sites and offering them early access to the application is a great way to get started in attracting a new interested audience that, otherwise, may just be too busy with other similar applications to see the great new ideas or concepts this one has over them all.

Finding promoting entities and companies that may sponsor the project is also a good option in several senses. It is not only a good manner to cover and exploit a wider crowd, but also a way to explore different resources and implement expensive features that, otherwise, would be economically impossible to put in practice.

The last resort, which is most expensive but works great, is the advertising system. This includes the Apple's iAd system itself since it uses application downloads and conversion as a metric, thus being one of the best choices in the advertising field.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

This chapter presents the conclusions about this work. It takes the reader through the research context and the application objectives, identifying those which have been met. This chapter also discusses some future work which may enhance and increase the application value.

7.1. Research Context and Objectives

The main purpose of this project was to develop a mobile application, able to take advantage of the regular and limited mobile devices' capabilities, in order to provide an automated and optimized device monitoring system.

The solution was projected to be used in both the monitoring and monitored devices, providing the users with location tracking and traveled path checking features. However, the envisioned key-feature of this project was an event-based alert notification system, which is an innovative capability in relation to rival applications.

One of the main concerns approached were the hardware limitations of mobile devices considering the location-aware requirements for the monitoring solution. Mobile devices' battery life still is the major obstacle for the evolution of the remaining hardware components. GPS is the hardware element that uses more energy, thus designing and implementing a power saving mechanism was one of the main goals for this project.

Other limitation which has been considered was the strict bandwidth limits which mobile operators submit costumers to. A monitoring solution requires data communication between many different entities, therefore, designing a bandwidth saving solution was considered in order to reduce costs and energy use.

Another settled objective was to ensure that the application would be interoperable with future ports of it to other platforms and devices, by using standardized communication protocols. Since the application has only been developed for iOS based devices, it would be important to have the possibility of extending the application to other mobile platforms, such as Android and Windows Phone, which are deployed on more affordable mobile devices.

The resulting product from this project is thought to be commercialized since its architecture involves online services which the application rely on to keep working. In order to make enough profits to cover maintenance and service hosting expenses, it was considered to define a proper business model.

7.2. Accomplishments

The application fulfills all the pre-established requirements. The user is able to track anyone in the world and to be also the tracking target, without bleaching any privacy aspects. The user shares its private information with who he wants and voids that “contract”, temporarily or permanently, whenever he desires.

The automation purpose of this project was met in the event-based alert notifications feature, where the user does not need to be constantly and manually monitoring a certain device, in order to be up-to-date with its latest events. This methodology contributed to the efficient power and bandwidth management.

The event-based alert notifications feature was further enhanced by including battery and bandwidth saving techniques. Those mechanisms allow the user to use the application longer without awfully interfere with external factors such as the debilitated phones battery and the limited and expensive traffic plans provided by mobile networks.

During development, standard protocols have been used in order to assure interoperability with future devices and platforms this application may be ported to.

A profitable business solution was found through the analysis of the annual profit projections for the different business models available. It was concluded that the paid business model is the more profitable one whereas the mixed advertising and subscription model is the more appellative to the consumer.

Through the profit projections, it was learned that the advertising model or the subscription model alone would not be profitable since it does not generate enough revenue to cover the maintenance of the online services that supports the application.

As result of the whole process, an optimized, secure and graphically appealing solution was created for one of the most successful and profitable mobile platforms ever, the iOS.

7.3. Future Work

There are other possibilities that may be explored in the future. The application has a consistent platform capable of retrieving GPS data from a certain device. It would be interesting to be able to retrieve some more useful information from that device. That information would include the accelerometer axes, battery info, surrounding Wi-Fi networks information and other personal data such as call and message history.

Unfortunately, the Apple’s iOS SDK is very restrictive and does not allow many of those features to be implemented. Even if they would do, the application would not be accepted in App Store due to privacy concerns those features raise. On the other hand, the Android platform is more receptive to this kind of applications as its SDK provides more freedom to the developer regarding hardware and personal information access.

The Android operating system is deployed in a vast range of mobile devices, including low cost ones. When porting the application to this platform, the different family budgets are taken into account. This is a crucial factor regarding the application context, since parental control purposes are one of the strands for its use and some parents may not be able to

afford expensive devices per each child they may have. Thus, the multi-platform porting of the application would be a good way to reach several and bigger market segments.

Since the application is already able to transmit location data from devices that can follow people wherever they go, the possible recipients of that information can be extended to other kind of devices. Those devices do not necessarily need to be portable. Hence, accessing all the information retrieved by the mobile devices through a web browser on a computer can be quite appealing to some users. In addition, not both parts need to have a supported mobile device in order to monitor any other, which would cover another market segment. Therefore, developing a web portal for mobile device tracking purposes should be considered.

REFERENCES

[121282.com, 2011] 121282.com. “iPhone UDID SHA1 Script”. Retrieved September 3, 2011 <http://121282.com/iphone_udid.txt>.

[Abandoned, 2011] The Form Group. Abandoned. Retrieved September 29, 2011 <http://www.theformgroup.com/work/form-apps/abandoned>.

[ABI Research, 2011] ABI Research. “Mobile Device Market Share Tracker”. Retrieved October 22, 2011 <http://www.abiresearch.com/research/1003402-Mobile+Device+Market+Share+Tracker>.

[AdMob, 2011] AdMob.com. “AdMob Enriching Mobile”. Retrieved April 24, 2011 <http://www.admob.com>.

[Allan, 2011] Allan, Alasdair. “Basic Sensors in iOS”. O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. July 2011, First Edition.

[AMQP, 2011] AMQP Working Group. Advanced Message Queuing Protocol. Retrieved September 29, 2011 <http://www.amqp.org/about/what>.

[Apple Inc., 2008a] Apple Inc. “iPhone App Store Downloads Top 10 Million in First Weekend”. Press Release. Apple Inc. 2008-07-14. Retrieved February 25, 2011 <http://www.apple.com/pr/library/2008/07/14App Store.html>.

[Apple Inc., 2008b] Apple Inc. “Apple Introduces the New iPhone 3G”. Apple. July 9, 2008. Retrieved February 25, 2011 <http://www.apple.com/pr/library/2008/06/09iphone.html>.

[Apple Inc., 2010a] Apple Inc. “The iOS Arquitecture”. iOS Developer Library. November 15, 2010. Retrieved August 12, 2011 <http://developer.apple.com/library/ios/#documentation/MiscellaMisce/Conceptual/iPhoneOSTechOverview/IPhoneOSOverview/IPhoneOSOverview.html#//apple_ref/doc/uid/TP40007898-CH4-SW1>.

[Apple Inc., 2010b] Apple Inc. “UIDevice Class Reference”. October 28, 2010. Retrieved September 18, 2011 <http://developer.apple.com/library/ios/#documentation/uikit/refererefer/UIDevice_Class/Reference/UIDevice.html>.

[Apple Inc., 2011a] Apple Inc. “iAd”. Developer Support Center. Retrieved July 8, 2011 <http://developer.apple.com/iad>.

[Apple Inc., 2011b] Apple Inc. “iOS Developer Program: Distribute.” Retrieved July 8, 2011 <http://developer.apple.com/programs/ios/distribute.html>.

[Apple Inc., 2011c] Apple Inc. “Program Renewals.” Developer Support Center. Retrieved September 28, 2011 <http://developer.apple.com/support/ios/program-renewals.html>.

[Arthi, 2010] Arthi, L. Latest Trends In IT. Hybrid Positioning System (XPS). K. S. RANGASAMY College Of Technology Tiruchengode. Register no.: 0714109, February 2010.

[Bahl *et al.*, 2000] Bahl, P.; Padmanabhan, V.N.; “RADAR: an in-building RF-based user location and tracking system”, INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE , vol.2, no., pp.775-784 vol.2, 2000.

[Barrett, 2009] Barrett, Kathryn. “What Can You Do with XMPP?”. O'Reilly FYI. Retrieved September 29, 2011 <http://fyi.oreilly.com/2009/05/what-can-you-do-with-xmpp.html>.

[BuySellAds, 2011] BuySellAds. “Attack of the In-App Ad”, July 15, 2011. Retrieved August 11, 2011 <http://blog.buysellads.com/2011/07/attack-of-the-in-app-ad>.

[CASA, 2006] Global Navigation Satellite Systems. Australian Government. Civil Aviation Safety Authority, 2006.

[Clarinox, 2009] Clarinox Technologies Pty Lt. “Real Time Location Systems”. Nov 2009. Retrieved January 12, 2011 <http://www.clarinox.com/docs/whitepapers/RealTime_main.pdf>.

[Dailyator, 2011] Dailyator. “US Declares iPhone *Jailbreaking* Legal Over Apple’s Objections”. Retrieved August 23, 2011 <http://dailyator.com/u-s-declares-iphone-jailbreaking-legal-over-apple%E2%80%99s-objections/27580>

[Das, 2010] Das, Debasis. “What is GBAS (Ground Based Augmentation System)?”. June 2010. Retrieved February 14, 2011 <http://www.brighthub.com/electronics/gps/articles/76030.aspx>

[Educause, 2009] Educase. “7 Things You Should Know About Location-Aware Applications”. Educase Learning Initiative. March 2009.

[Eissfeller *et al.*, 2007] Eissfeller, Bernd; Ameres, Gerald; Kropp, Victoria; Sanroma, Daniel; München; “Performance of GPS, GLONASS and Galileo”, pp. 185-199.

[eJabberd, 2011] eJabberd. “Community Site”. Retrieved June 12, 2011 <http://www.ejabberd.im>.

[Ekahau, 2011a] Ekahau.com. “Ekahau Real-Time Location System (RTLS) Overview”. Retrieved February 25, 2011 <http://www.ekahau.com/products/real-time-location-system/overview.html>.

[Ekahau, 2011b] Ekahau. “The Science of High-Performance Wi-Fi Real Time Location System”. Retrieved February 25, 2011 <http://www.ekahau.com/products/real-time-location-system/overview/technology.html>.

[FAA, 2011] Federal Aviation Administration. GBAS Architecture. Retrieved March 16, 2011 <http://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/navservices/gnss/laas/view/dspLAAS_Architecture.cfm>.

[Friendjectory, 2011] Friendjectory. “See your friends on map in real time!”. Retrieved September 28, 2011 <http://friendjectory.com>.

[Geotwitts, 2011] GeoTwitts. “Post your GPS location to Twitter from your iPhone”. Retrieved October 5, 2011 <http://www2.ladelo.com:8080/geotwitts>.

[Gezici, 2008] S. Gezici, “A survey on wireless position estimation”, *Wireless Personal Communications*, vol. 44, no. 3, pp. 263-282, February 2008.

[Gezici *et al.*, 2005] Gezici, S.; Zhi Tian; Giannakis, G.B.; Kobayashi, H.; Molisch, A.F.; Poor, H.V.; Sahinoglu, Z.; , “Localization via ultra-wideband radios: a look at positioning aspects for future sensor networks”, *Signal Processing Magazine*, IEEE , vol.22, no.4, pp. 70- 84, July 2005.

[Google, 2011] Google. “Google Latitude”. Retrieved September 26, 2011 <http://itunes.apple.com/app/google-latitude/id306586497?mt=8>.

[Grigorik, 2009] Grigorik, Ilya. Advanced Messaging & Routing with AMQP. September 2010. Retrieved September 29, 2011 <http://www.igvita.com/2009/10/08/advanced-messaging-routing-with-amqp>.

[GU *et al.*, 2009] Gu, Yanying; Lo, Anthony; Niemegeers, Ignas; Senior Member; IEEE. “A Survey Of Indoor Positioning Systems For Wireless Personal Networks”. IEEE Communications Surveys & Tutorials, Vol. 11, No. 1, First Quarter 2009.

[Hallauer, 2010] Hallauer, Thomas. “Wi-Fi positioning is about to change gear”. March 16, 2010. Retrieved February 22, 2011 <http://news.thewherebusiness.com/content/wi-fi-positioning-about-change-gear>.

[Han *et al.*, 2010] Ooi Wei Han; Nurfarhana Md. Noor; Noor Azawani bt. Wahap; Dr. Noordin Ahmad, “GNSS: Technology and Status Updates”, Space Application and Technology Development Division National Space Agency (ANGKASA), Workshop on GNSS Technology and Application, 27-28 September 2010, PICC.

[Hegarty and Chatre, Hegarty and Chatre. “Evolution of the Global Navigation

2008] Satellite System (GNSS)". Proceedings of the IEEE (2008) vol. 96 (12) pp. 1902 – 1917.

[ICAO, 2005] International Civil Aviation Organization. "Global Navigation Satellite System (GNSS) Manual". Doc 9849 AN/457. First Edition, 2005.

[iCloud, 2011] Apple Inc. "iCloud". Retrieved October 3, 2011 <http:// www.icloud.com>.

[IDC, 2011] IDC Corporate USA. "Analyse the Future". Retrieved June 23, 2011 <http://www.idc.com>.

[IETF, 2004] IETF. Extensible Messaging and Presence Protocol RFC 3920. Network Working Group. Jabber Software Foundation. October 2004 < http://www.ietf.org/rfc/rfc3920.txt>.

[IgniteRealtime, 2011a] IgniteRealtime. "Openfire". Retrieved September 24, 2011. <http://www.igniterealtime.org/projects/openfire>.

[IgniteRealtime, 2011b] IgniteRealtime. "Openfire Connection Manager Module". Retrieved September 24, 2011. <http://www.igniterealtime.org/projects/openfire/connection_manager.jsp>.

[IpodNN, 2010] IpodNN.com. "Apple quietly switches to first-party location services". July 30, 2010. Retrieved February 23, 2011 <http://www.ipodnn.com/articles/10/07/30/ditches.google.skyhook>.

[JabberHosts, 2011] JabberHosts. "Jabber IM Hosting Services". Retrieved September 28, 2011 <http://www.jabberhosts.com>.

[Jones, 2009] Jones; M. Tim. "Meet the Extensible Messaging and Presence Protocol (XMPP)". Retrieved September 18, 2009 <http://www.ibm.com/developerworks/webservices/library/x-xmppintro/index.html>.

[JQMedia, 2010] JQ Media. "Display Ad Performance: CPM and Fill Rate". Retrieved September 28, 2011 <http://jquaglia.com/?p=34>.

[Kalmer, 2011] Kalmer, Kenneth. "To AMQP or to XMPP, that is the question." Retrieved February 12, 2011 <http://www.opensourcery.co.za/2009/04/19/to-amqp-or-to-xmpp-that-is-the-question>.

[Kofahl *et al.*, 2004] Kofahl, M.; Bill; C Cap; Mundt, T, "Indoor and Outdo or Positioning in Mobile Environments—a Review and some Investigations on WLAN-Positioning." Traffic, 2004.

[Lookout, 2011] Lookout Mobile Security. "App Gnome Report". February 2011. Retrieved September 22, 2011 <https://www.mylookout.com/appgenome >.

[Marioli *et al.*, 1992] Marioli, Daniele; Narduzzi, Cláudio; Offelli, Carlo; Petri, Dário;

Sardini, Emílio; Taroni, Andrea; “Digital Time-of-Flight Measurement for Ultrasonic Sensors”. IEEE Transactions on Instrumentation and Measurement. 1992; 41(1):93-97.

[Millennial Media, 2011] Millennial Media. “Mobile Mix: The Mobile Device Index”. October 2011. Retrieved October 10, 2011 <http://www.millennialmedia.com/research>.

[Moore, 2009] Moore, Terry. GPS, “Galileo and the Future of Satellite Positioning Systems”. Institute of Engineering Surveying and Space Geodesy, The University of Nottingham, 2009.

[Morgan-Owen *et al.*, 1995] Morgan-Owen, G.J.; Johnston, G.T.;, “Differential GPS positioning”, Electronics & Communication Engineering Journal , vol.7, no.1, pp.11-21, Feb 1995 doi: 10.1049/ecej:19950104.

[Murphy and Imrich, 2008] Murphy, T; Imrich, T., “Implementation and operational use of ground-based augmentation systems (GBASs) VA component of the future air traffic management system”, Proc. IEEE, vol. 96, no. 12, pp. 1936–1957, Dec. 2008.

[MyRepoSpace, 2011] MyRepoSpace. UDIDFaker. Retrieved March 23, 2011 <http://www.myrepospace.com/profile/iphonepk/8574/UDIDFaker>.

[Navizon, 2011] Navizon. “The Peer-to-Peer Wireless Positioning. Navizon For Businesses”. Retrieved February 21, 2011 <http://www.navizon.com/businesses.php>.

[Nokia, 2011] Nokia. “Nokia and Microsoft announce plans for a broad strategic partnership to build a new global ecosystem”. Retrieved February 25, 2011 <http://press.nokia.com/2011/02/11/nokia-and-microsoft-announce-plans-for-a-broad-strategic-partnership-to-build-a-new-global-ecosystem>.

[O’Dell, 2011] O’Dell, Jolie. “The Rise of Mobile In-App Adds”. July 16, 2011. Mashable.com. Retrieved June 16, 2011 <http://mashable.com/2011/07/16/in-app-ads>.

[Priyantha *et al*, 2000] Priyantha, Nissanka B.; Chakraborty, Anit; Balakrishnan, Hari, “The Cricket Location-Support System”, Proc. 6th ACM MOBICOM, Boston, MA, August 2000.

[Paul and Wan, 2008] Paul, A.S.; Wan, E.A.; , “Wi-Fi based indoor localization and tracking using sigma-point Kalman filtering methods”, Position, Location and Navigation Symposium, 2008 IEEE/ION , vol., no., pp.646-659, 5-8 May 2008.

[Porrmann, 2009] Porrmann, Meike (Ubisense), “Ubisense RTLS – an Overview, Presentation unpublished”, Ubisense Limited, 2009.

[Premchaisawasdi *et al.*, 2009] Premchaisawasdi, Wichian; Romsaiyud; Walisa, “Intelligent switching method using Cell-ID/GPS positioning on mobile

application.” 2009 7th International Conference on ICT and Knowledge Engineering. December 2009.

[ProcessOne, 2010] ProcessOne. “Ejabberd Installation and Operation Guide”. 2010. Retrieved September 26, 2011< http://www.process-one.net/docs/ejabberd/guide_en.html>.

[Rovio, 2011] Rovio Entertainment ltd. Retrieved July 8, 2011 <http:// http://www.rovio.com>.

[Saint-Andre, 2005] Saint-Andre; P. "Streaming XML with Jabber/XMPP," *Internet Computing, IEEE* , vol.9, no.5, pp. 82- 89, Sept.-Oct. 2005 doi: 10.1109/MIC.2005.110

[SCVNGR, 2011] SCVNGR. Retrieved September 29, 2011 <http://www.scvngr.com>.

[Smith, 2010] Smith, Eric. “iPhone Applications & Privacy Issues: An Analysis of Application Transmission of iPhone Unique Device Identifiers (UDIDs)”. October 1, 2010.

[Spiro, 2010] Spiro, Josh. “Mobile Application Design”. 2010. Retrieved September 21, 2011 <http://www.inc.com/best-industries-2010/mobile-app-design-industry.html>.

[SkyhookWireless, 2011] SkyhookWireless. “Core Engine Overview”. Retrieved February 21, 2011 <http://www.skyhookwireless.com/howitworks>.

[STMicroelectronics, 2011] STMicroelectronics. “STMicroelectronics Debuts the World’s First Single-Chip Positioning Device for Multiple Global Navigation Systems”, January 2011. Retrieved February 18, 2011 <http://www.st.com/Internet/com/press_release/p3113.jsp>.

[Task Ave, 2011] Task Ave. “Remember What You Need To Do, Where You Need To Do It”. Retrieved September 25, 2011 <http://www.taskave.com>.

[TheAppleLounge, 2011] TheAppleLounge. “Applicazioni iPhone: In-App Purchase non decolla”.Retrieved April 25, 2011 <http://www.theapplelounge.com/hardware/iphone/applicazioni-iphone-app-purchase>.

[TheCoffeeDesk, 2009] TheCoffeeDesk. “An Overview of the iPhone Architecture”. May 17, 2009. Retrieved August 12, 2011 <http://thecoffeedesk.com/news/index.php/2009/05/17/iphone-architecture>.

[Thomson *et al.*, 2010] Thomson, Martin; Harper, Neil; Tran, Khiem, “System and Method for A-GPS Positioning of a Mobile Device”, United States, ANDREW, LLC (Hickory, NC, US), 2010.

[TouchJSON, 2011] TouchJSON. “A human JSON Objective-C un-framework”. Retrieved February 12, 2011 <https://github.com/TouchCode/TouchJSON>.

[Ubisense, 2011] Ubisense.net. “Precise real-time location”. Retrieved February 25, 2011 <http://www.ubisense.net/en/products/precise-real-time-location.html>.

[UNOOSA, 2010] United Nations Office For Outter Space. “Current and planned global and regional navigation satellite systems and satellite-based augmentation systems”. International Committee on Global Navigation Satellite Systems Provider. United Nations publication, June 2010.

[Neowneow, 2011] Neowneow. “iPhone In-App Purchase”, January 14, 2011. Retrieved August 11, 2011 <http://www.neowneow.com/2011/01/iphone-in-app-purchase/>.

[Venture Beat, 2008] Venturebeat.com. “There’s a great future in iPhone apps", Venture Beat, June 11, 2008. Retrieved 25 February 2011 <http://venturebeat.com/2008/06/11/analyst-thees-a-great-future-in-iphone-apps>.

[Wirelesswerx, 2011a] Wirelesswerx. “Turning Locations into Destinations”. Retrieved February 24, 2011 <http://www.wirelesswerx.com>.

[Wirelesswerx, 2011b] Wirelesswerx. “MobiWERX”. Retrieved February 24, 2011 <http://www.wirelesswerx.com/MobiWERX.php>.

[Wirelesswerx, 2011c] Wirelesswerx. “SiteWERX”. Retrieved February 24, 2011 <http://www.wirelesswerx.com/SiteWERX.php>.

[XMPP, 2004] XMPP Standards Foundation. “Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence”. Retrieved October 1, 2011 <http://xmpp.org/rfcs/rfc3921.html#intro-requirements>.

[XMPP, 2011a] XMPP Standards Foundation. “Extensible Messaging and Presence Protocol”. Retrieved October 2, 2011 <http://xmpp.org/about-xmpp>.

[XMPP, 2011b] XMPP Standards Foundation. “Microsoft Adds XMPP Support to Windows Live APIs”. Retrieved October 2, 2011 <http://xmpp.org/2011/09/microsoft-adds-xmpp-support-to-windows-live-apis>.

[XMPP, 2011c] XMPP Standards Foundation. “Skype Adds XMPP Support”. Retrieved October 2, 2011 <http://xmpp.org/2011/06/skype-adds-xmpp-support>.

[XMPPFramework, 2011] XMPPFramework. “XMPP Framework for Cocoa”. Retrieved February 14, 2011 <http://code.google.com/p/xmppframework>.