GlucoServer

Design, Development and Exploitation Plan of a Health Monitoring Network

Victor Bernet Palacin

June 14, 2011

Acknowledgements

It has been a while since I started my degree dreaming about the day I could present my Final Project and be an Engineer. This day has finally come and I would like to thank all the people who has walked hand by hand with me.

First of all I have to thank all my family for their unconditional support, specially my parents, Victor and Pilar, who have always encouraged me to continue whatever it happened, and have helped me to overcome any problem that occurred.

Secondly, I would like to thank all my Spanish friends that despite being away from home this late months have made me feel like if they were here.

Also, I would like to thank all the people I have met during this experience in Paris. The Foyer Naples family, all the Spanish crew and all the people that have gave me some support during my stage in this amazing city.

Finally I would like to thank both my tutor from UPC, Pere Losantos, and my tutor from ECE, Laurent George, for being so patient with me and always helping me to put my project on the right track until it has been finished.

Contents

1 Introduction

In recent years the world has been experiencing a great change in the framework of Health. Changes such the increase in the number of people affected by chronic diseases aims us to be in continuous search for solutions to improve both the provision of medical services and the life of these patients.

Faced with this problem, we can find in eHealt a good solution. eHealth is the discipline that uses information technology in any variant applied to medical services. This tool allows us to innovate and thus improve the quality and efficiency of these services.

1.1 Objectives

In this context, our purpose is to introduce us in the eHealth market by creating a Health Monitoring Network conceived as a service addressed to hospitals. This system will help them to innovate in providing services to patients with chronic diseases, specifically diabetes.

The main objective of this system is to be in charge of sending patients' health information to medical services in order to track the patient day by day, and to give doctors more details about the patients' status. In order to do this, whenever a measurement will be made by a patient, the result will travel directly to a database where the doctor would be able to enter whenever he wants to look for his patients information.

In addition, the system should be able to send an alarm SMS when the patient's life is in danger allowing the intervention of medical services. In order to make this intervention fast and efficient the message will contain relevant information such as the patients glucose level or his latitude and longitude localisation.

Moreover, another objective is to make it easier for the patient to contact his doctor to report some anomalies or just to ask about a doubt whenever he wants.

1.2 Structure

In order to do this, we will start by introducing a theoretical innovation framework, explaining its types and more specifically how can we innovate in the service providing system, analyzing each of the dimensions necessary to provide a service.

Then, we will study more thoroughly the eHealth market, introducing its current state, analyzing its future expectations and what are the inhibitors that can be found at end of its full implementation. We will also place our project in this market, finding tools, among all those that make up this discipline, we will use and why. Moreover, we will overview some interesting innovations in this market.

Secondly, we will find the exploitation plan of the service which will consist of a first analysis of the environment, properly defining what kind of service we are offering, what will be our potential customers and what could be our competitors in the market exploiting this technology. We will also go a little further into explaining both the system architecture and the features it offers to its users. To complete this business plan, we will make a SWOT analysis on the service to better understand the real possibility that this could have to success on the market and, based on the facts about innovation studied in the first section and once we know more about our project, we will discuss our technology from the innovation point of view to analyze why our service is innovative and how hospitals can use it as a tool to innovate in their services.

Thirdly, we will study in depth all the technical development process of the application that provides the basis of our communication system. To do this we will see both stages of programming and simulation, to verify proper operation in a local network scenario.

Finally, in order to sum up, we will find the conclusions about the project and at a personal level.

2 Innovation in Service Providing System

'Innovation is the process of creating new ideas and turning them into new business and social value.' [26]

Innovation is a vast topic, one with many definitions and ways to understand them. In 2005, the European Commission defined innovation as 'the implementation of a new or significantly improved product (good or service), or process, a new marketing method, or a new organisational method in business practices, workplace organisation or external relations'. [28]

Colloquially, the word "innovation" is often synonymous with the output of the process. Moreover, to lead to the implementation of innovations, we have to follow some scientific, technological, organisational, financial and commercial steps called innovation activities. Some innovation activities are themselves innovative, others are not novel activities but are necessary for the implementation of innovations.

In all companies, innovation has become the main tool to be more competitive seeking new ways of being, new activities and in general one way to differentiate in an environment increasingly uncertain and changing.

2.1 Kinds of innovation

There are several types of innovation and ways to sort them. Thus, we can classify the different types of innovations depending on what are we innovating and the way that this innovation happen.

2.1.1 According to what we innovate

Four types of innovations are distinguished: product innovations, process innovations, marketing innovations and organisational innovations. Product innovations and process innovations are closely related to the concept of technological product innovation and technological process innovation.

A **product innovation** is the introduction of a good or service that is new or significantly improved with respect to its characteristics or intended uses. This includes significant improvements in technical specifications, components and materials, incorporated software, user friendliness or other functional characteristics.

Product innovations can utilise new knowledge or technologies, or can be based on new uses or combinations of existing knowledge or technologies. The term “product” is used to cover both goods and services.

The development of a new use for a product with only minor changes to its technical specifications is a product innovation. Significant improvements to existing products can occur through changes in materials, components and other characteristics that enhance performance.

Product innovations in services can include significant improvements in how they are provided (for example, in terms of their efficiency or speed), the addition of new functions or characteristics to existing services, or the introduction of entirely new services.

A **process innovation** is the implementation of a new or significantly improved production or delivery method. This includes significant changes in techniques, equipment or software.

Process innovations can be intended to decrease unit costs of production or delivery, to increase quality, or to produce or deliver new or significantly improved products. Production methods involve the techniques, equipment and software used to produce goods or services.

Moreover, process innovations also cover new or significantly improved techniques, equipment and software in ancillary support activities, such as purchasing, accounting, com-

puting and maintenance. The implementation of new or significantly improved information and communication technology (ICT) is a process innovation if it is intended to improve the efficiency or quality of an ancillary support activity.

A **marketing innovation** is the implementation of a new marketing method involving significant changes in product design or packaging, product placement, product promotion or pricing.

This kind of innovations are aimed at better addressing customer needs, opening up new markets, or newly positioning a firm's product on the market, with the objective of increasing the firm's sales.

An **organisational innovation** is the implementation of a new organisational method in the firm's business practices, workplace organisation or external relations. Organisational innovations can be intended to increase a firm's performance by reducing administrative costs or transaction costs, improving workplace satisfaction and in consequence labour productivity, gaining access to non- tradable assets such as non-codified external knowledge or reducing costs of supplies.

Organisational innovations in business practices involve the implementation of new methods for organising routines and procedures for the conduct of work. These include, for example, the implementation of new practices to improve learning and knowledge sharing within the firm.

2.1.2 According to how innovation occurs

On the other hand, if we look at the way that innovation occurs, we can distinguish three types: Incremental innovation, radical innovation and open innovation.

Incremental innovation refers to the creation of added value on an existing product, adding some improvement.

In contrast, **radical innovation** or rupture refers to a change or introduction of a new product, service or process that was not known before.

Finally, **open innovation** refers to those developments coming from a general public, and is characterized by open participation of individuals from all over the world.

2.2 Innovation in services

Although there are many definitions of the term 'service', the following is probably the closest to what is essential in business: "Produce a service is to organize a solution to a problem that mainly involves providing a good. Is to put a set of capabilities and competences (human, technological, organizational) available to the client and arrange a solution ".[15]

The main characteristics of services are the following:

- Intangibility: they cannot be touched, gripped, handled, looked at, smelled, tasted or heard. Thus, there is neither potential nor need for transport, storage or stocking of services.
- Heterogeneity: they depend on the interaction between buyer and supplier.
- Inseparability: The service provider is indispensable for service delivery as he must promptly generate and render the service to the requesting service consumer. Additionally, the service consumer is inseparable from service delivery because he is involved in it from requesting it up to consuming the rendered benefits.
- Simultaneity: Services are rendered and consumed during the same period of time.
- Variability: Each service is unique. It is one-time generated, rendered and consumed and can never be exactly repeated as the point in time, location, circumstances, conditions, current configurations or assigned resources are different for the next delivery.

Many service provider companies are faced with the need to innovate to remain competitive and constantly introduce new services or improving existing. However, despite being aware of the need for innovation in order to bring competitive advantages through new or improved services, the lack of resources and time for innovation is still a challenge for many of these companies.

2.2.1 Innovation dimensions in service providing system

According to a guide for innovation in services made by CIDEM (Centre d'Innovació i Desenvolupament Empresarial), service providing system is a set of activities or dimensions needed by an enterprise in order to be able to provide a service; therefore any innovation will mean redefining any of this activities. In Figure 2.1 we can see a diagram of the different dimensions were the enterprise can innovate adding a great value for the client. [10]

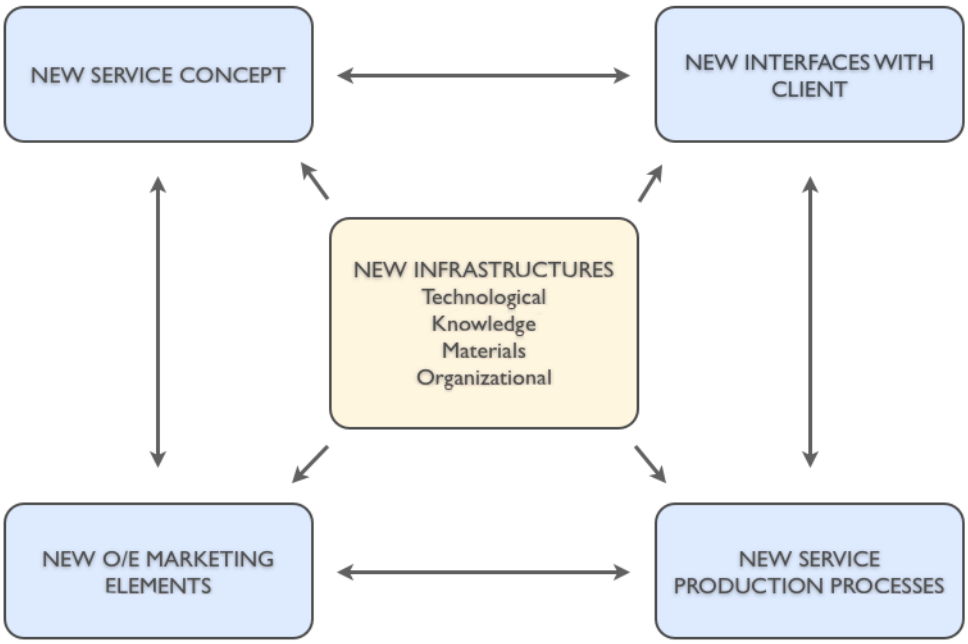

Figure 2.1: Innovation dimensions in service providing system

Despite this dimensions are defined as separate activities, it is frequent to find innovation projects meaning the appliance of innovations in more than one dimension at the same time. Moreover, it is also clear that depending on the grade of reconfiguration and on how many dimensions we apply this changes we can end up radically changing our service or even leading to new business models.

New Operative and Strategic Marketig elements

One of the activities that directly affect on the differentiation of business is Marketing. Service enterprises can apply innovations either to strategic or operative marketing.

Under the perspective of strategic marketing we can analyze and innovate in segmentation, where we can create new innovative variables to segmentate, or in positioning, innovating in the kind of facilities, furniture, costumes and other key factors for conforming the image we want to convey.

On the other hand, in terms of operative marketing or marketing mix, we can innovate in pricing, communication and intermediation.

New service concepts

In the service providing system, a new service is sometimes difficult to identify since it is intangible. However, we understand a new service concept as a new way to organize a solution to a problem and make it available for the costumer. Ultimately, it is a new way to solve a problem that till this moment has been solved less satisfactory, or finding the solution for new needs or problems.

However, it is difficult to understand a service as a single isolated element. That is why sometimes we can get to the creation of a new innovative service offer just by recombining different services already existing.

New interfaces with client

One of the most important moments in the organization of services is when there is contact with the customer. Is at this time when the client can better assess whether or not the service is satisfactory compared to the initial expectations.

The client interfaces and different ways to innovate in this area will be different depending on the grade of customer participation in the service. One the one hand, we find services in which the client is simply the receiver of the service. In this case, innovation projects can be promoted to ensure faster service, more enjoyable timeout, better treatment or even better service customization.

On the other hand, we have the cases in which the client is co-producer of the service. In this case, besides the areas mentioned above, it is also possible to innovate in areas such as the provision of tools to integrate the shared information or the grade of client participation in the service.

New service production processes

When a firm innovates in new ways of producing a service, most of the times the service concept is unchanged. However, the introduction of innovations in the process necessary to provide the service introduces significant advantages for the company.

This dimension of the service providing system focuses on how we make the services and the company's ability to find new, more efficient or better ways to make the same service. Therefore, the main reason to innovate in this area is to achieve lower production costs, improve productivity, increase quality and reliability of the service, improve reaction and response time to a problem and the flexibility of the service.

In this dimension we can find good opportunities for innovation, whether in the elimination of operations that add no value to the customer, the incorporation of ICT to improve efficiency and effectiveness or the development of knowledge and skills of those involved in process.

New infrastructures

A new infrastructure can help a company to provide a better service to its customers, or clearly differentiated to what other competitors use to offer. Technological infrastructures, defined as everything that brings together in ICT, are sometimes the first thing that comes to mind when speaking about new infrastructures.

However, other infrastructures as material, knowledge o organizational ones can become a tool for innovation in other dimensions as well as a great source of innovation itself.

3 eHealth Market

'A term needed to describe the combined use of electronic communication and information technology in the health sector. The use in the health sector of digital data – transmitted, stored and retrieved electronically – for clinical, educational and administrative purposes, both at the local site and at a distance.'[25]

There are plenty of definitions about this comparatively new term. From its first uses, around 1990, it's been sometimes used as a general "buzzword," used to characterize not only "Internet medicine", but also almost everything related to computers, telecommunications and medicine.

Moreover, many experts acknowledge in the fact that stamping a definition on something like eHealth is like stamping a definition on 'the Internet', because its definition cannot be pinned down, as it is a dynamic environment, constantly moving and changing.

However, what we can all agree is that eHealth consists on the application of Internet and other related technologies in the healthcare industry to improve the access, efficiency, effectiveness, and quality of clinical and business processes utilized by healthcare organizations, practitioners, patients, and consumers to improve the health status of patients.

Moreover, that term can encompass a large range of services or systems that are at the edge of medicine/healthcare and information technology. Some of them are the following:

- Electronic health records: enabling the communication of patient data between different healthcare professionals (GPs, specialists etc.)
- Telemedicine: physical and psychological treatments at a distance, using different ways of communication as internet or phone.
- Consumer health informatics: use of electronic resources on medical topics by healthy individuals or patients.
- Health knowledge management
- Virtual healthcare teams: consisting of healthcare professionals who collaborate and share information on patients through digital equipment (for transmural care).

- mHealth or m-Health: includes the use of mobile devices for collecting patient health data, providing healthcare information to practitioners, researchers, and patients, real-time monitoring of patient vitals, and direct provision of care (via mobile telemedicine).
- Medical research using Grids: powerful computing and data management capabilities to handle large amounts of heterogeneous data.
- Healthcare Information Systems: also often refer to software solutions for appointment scheduling, patient data management, work schedule management and other administrative tasks surrounding health.

3.1 Where are we?

From all the services or systems that eHealth encompasses, our project will be mainly based on mHealth. As we will explain below, a smartphone will be the responsible of capturing all the data coming from the glucometer via bluetooth and sending it to the server.

There are many technologies related to mHealth but in our project we will use patient monitoring devices and data collection software for the smartphone.

However, another eHealth system will be needed in order to manage, store, and access the data. We will be using a network to transmit the information between the sensor, phone and server, in order for it to travel from the patient to the doctor without the interaction of the patient. We will then store all the data in a server. All this treatment of medical data is called, as we early explained, telemedicine.

Telemedicine can be broken in three main categories:

- Store and forward: involves acquiring medical data (like medical images, biosignals, etc) and then transmitting this data to a doctor or medical specialist at a convenient time for assessment offline. It does not require the presence of both parties at the same time. Dermatology, radiology, and pathology are common specialties that are conducive to asynchronous telemedicine. A properly structured Medical Record preferably in electronic form should be a component of this transfer. A key difference between traditional in-person patient meetings and telemedicine encounters is the omission of an actual physical examination and history. The store-and-forward process requires the clinician to rely on a history report and audio or video information instead of a physical examination.

- Remote monitoring: also known as self-monitoring, enables medical professionals to monitor a patient remotely using various technological devices. This method is primarily used for managing chronic diseases or specific conditions, such as heart disease, diabetes mellitus, or asthma. These services can provide comparable health outcomes to traditional in-person patient encounters, supply greater satisfaction to patients, and may be cost-effective.
- Interactive telemedicine services: provide real-time interactions between patient and provider, to include phone conversations, on-line communication and home visits. Many activities such as history review, physical examination, psychiatric evaluations and ophthalmology assessments can be conducted comparably to those done in traditional face-to-face visits. In addition, "clinician-interactive" telemedicine services may be less costly than in-person clinical visits.

In our project we will be operating in the second category, since the patient will be capturing results from the glucometer in order to send them afterwards to the doctor using the store and forward and store and forward to manage them and send them to the doctor.

3.2 Market Segmentation

eHealth Market can be segmented into three relevant markets, each with estimates of its market potential, while acknowledging that some large companies are active in more than one of these segments:

- Healthcare IT systems: Communication within experts to share information about the patients or any kind of medical information.
- Home Healthcare
- Transactional services, supporting the two previous segments.

Patients increasingly expect to stay longer in their homes, rather than in hospitals or hospices. **Home healthcare** proposes hardware and services aimed at being used by the patient and ambulatory care professionals. In our project, we are using this tools to:

- Automate monitoring of the patient's biological condition.
- Allow the patient to automatically report measurements to the physician, such as blood pressure monitoring, diabetes monitoring, etc.
- Reduce home risks for the patient with the introduction of an alarm signal in case of risk situation.

The main expected output of these systems is to reduce or anticipate the risks of severe crisis, while reducing the need for hospital or specialists visits.

Transactional services segment brings together storage and networking-related services, as platforms that support the identification of both patients and healthcare professionals, communications, and information storage and protection.

Healthcare system editors are actually relying on infrastructure specialists. They let the technological giants, telecom operators or database specialists hosts the transactional building blocks of their applications.

3.3 Challenges of the eHealth Market

eHealth market is still at a very early stage. There are still some issues of regulation, language, investment and training that need to be solved before attaining a serious growth.

Despite of it, there are some factors that aim us to say that big possibilities are attached to this growing market, like the rapid rise in mobile phone penetration in developing countries [27].

One of the biggest challenges is that the population in rich countries is ageing, and while a healthy longevity is a big gain at the personal level, the effects at the societal level are more complex. Every time there will be fewer people of working age having to support and take care of more retired people.

In addition, Physicians and policy makers acknowledge that every time more ageing patients desire to stay in their homes as long as possible, rather tan having to go to an hospital for being treated or just for a quick visit.

Another important factor related to the first one is that the prevalence of chronic conditions and multiple chronic conditions rises constantly with the age and multiple chronic conditions appear usually when people reach 50 to 60 years of age. Moreover, we can add the fact that in 2005, research showed that an adult with two or more chronic diseases costs four times as much on average as a healthy adult [13].

Because of this reasons, the overall cost of healthcare is expected to continue rising in the coming years. The current medical workforce and systems won't be sufficient to cover all this increasing healthcare needs with the same quality of service as today, and even less if we speak in terms of reducing health costs.

3.4 Inhibitors of eHealth adoption

Despite the big possibilities that this market seems to have, there are as always some barriers we have to take into account and try to overcome in order to introduce us in this market with success. Some of the most important are:

- The human relationship: some patients or doctors still prefer the traditional way of interacting with the other.
- Reluctance of healthcare players to embrace new technologies: Institutions may be slow to develop the use of new working methods and business processes that are based on the use of new platforms and devices.
- Medical data security hurdles, and ethical and legal requirements
- Potential health hazards from wireless communication technology: several studies about the impact of electromagnetic waves have been issued.
- Current health systems are different in each country: Global regulation will be needed to lower the current barriers to eHealth industry, and it appears to be way out of reach

Management of complex value chains and processes is necessary: Telecom operators might be in competition with other players such as software developers, health providers and IT players. Key partnerships will be needed to ensure collaboration between this players.

3.5 Innovations in eHealth

As we have learned, eHealth consists on the application of Internet and other related technologies to improve the access, efficiency, effectiveness, and quality of clinical processes. Therefore, eHealth can be a great tool to innovate in medical service providing and consequently improve the health status of patients.

Expensive, inefficient, and often ineffective, health care is dying for innovation. Seeing this, some of the most innovative technological companies all over the world have decided to use eHealth tools to give medicine market the makeover it needs.

Next, we will take a look at some of the latest innovative products and services related to eHealth.

CISCO Healthpresence

Using the network as a platform, the Cisco HealthPresence solution combines high-definition video, advanced audio, and network-transmitted medical data to create an environment similar to what people experience when they visit their doctor or health specialist. It can be configured to support many locations and uses presence and skill-based routing to connect patients to the most appropriate expert.

Designed for use by an attendant who is a licensed healthcare professional, it extends the reach of providers and specialists, and it can be used to redirect unnecessary visits to low-cost clinics.

Figure 3.1: CISCO HealthPresence virtual visit

It increases productivity by using scarce resources more efficiently, and it lowers healthcare costs by making delivery more streamlined. The Cisco HealthPresence solution creates a dynamic environment for patient care, delivers a true-to-life experience between the patient and the provider.

Results from Cisco HealthPresence pilot programs in Aberdeen, Scotland and San Jose, California, which found that more than 90 percent of the patients were satisfied with the remote care experience and would recommend it to others [5].

Google Health and Body Browser

Even Google, one of the most innovative companies nowadays, has decided to enter in the Health market with two products.

The first one is **Google Health**, that is a personal health information centralization service, also known as personal health record services. The service allows Google users to volunteer their health records, either manually or by logging into their accounts at partnered health services providers, into the Google Health system, thereby merging potentially separate health records into one centralized Google Health profile.

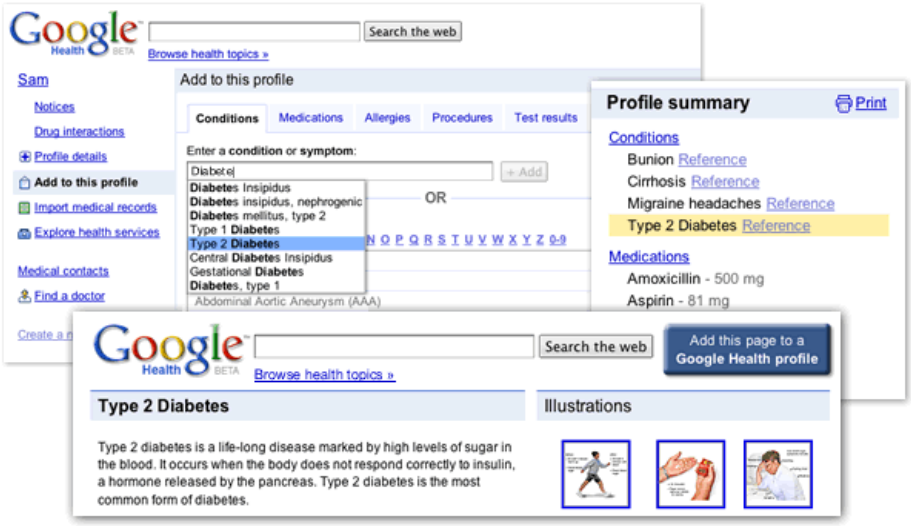

Figure 3.2: Google Health

Volunteered information can include "health conditions, medications, allergies, and lab results". Once entered, Google Health uses the information to provide the user with a merged health record, information on conditions, and possible interactions between drugs, conditions, and allergies.

On the other hand we found **Google Body Browser**. This body browser program allows users to examine the human body in layered, 3D detail.

The Google Body program enables users to peel back layers to see detailed views including muscle structures, organs, arteries, veins, and nerves.

The tool has similar functionality to Google Earth, allowing users to zoom and pan around various areas and search for specific parts of the body.

Users can then click the ‘labels’ option, and the program transforms into a virtual biology book, providing detailed notes on the body you are viewing.

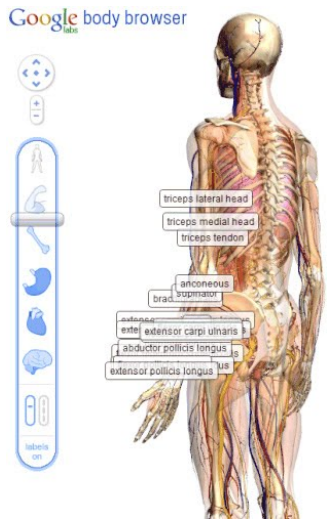

Figure 3.3: Google Body Browser

This can help health providers and specialists as long as they can found information about any of the parts of the human body we are travelling through.

Siemens Soarian IC

Soarian Integrated Care supports communication between various health care facilities. This is enabled through the exchange of diagnostic reports and images between the participants, and through the administration of a common patient file. As a result, the facilities participating in the treatment have easy access to all relevant information entered in the treatment network and provided in the patient file.

To ensure a smooth treatment process without unnecessarily repeating examinations, patient data has to be exchanged, such as information regarding treatments and therapies, diagnoses, and images. Often, the treating physician wants to obtain the opinion of a colleague or another institute, especially in the case of a difficult diagnosis.

With Soarian Integrated Care, all of this occurs easily via a secure common network, enabling all participants to view the relevant information with a click of the mouse.

This can help lower treatment costs and increase treatment quality, particularly when dealing with chronic diseases.

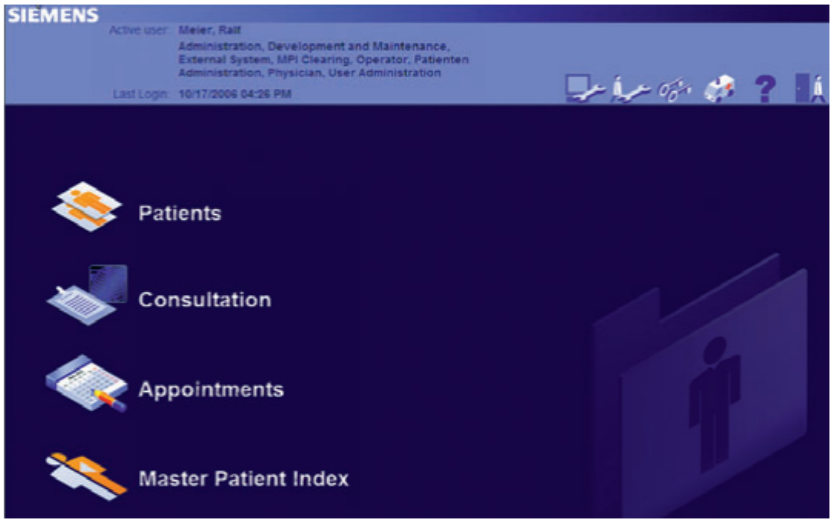

Figure 3.4: Siemens Soarian

4 GlucoServer

Lately we are witnessing that the entire world is facing a significant social change, brought about by an unprecedented demographic change: the ratio of elderly people to the entire population is steadily growing, while the ratio of younger age groups, especially the working population is shrinking.

Moreover, the number of chronic diseases rises with the age and diabetes is nowadays one of the most common chronic diseases with 220 million diabetic people all over the world [29].

Patients can better manage their diabetes and take maximum control of their condition through daily self-monitoring and recording of blood glucose values and other relevant data. It has been scientifically established, that continual monitoring of blood glucose levels has a positive effect on the general health of diabetes patients.

Seeing that, our main goal is to propose a system able to track diabetic patients' state during their normal life: at home, when moving in town, etc. With this system, we will facilitate the relation patient-doctor, avoiding the discomfort of moving to a health center in order to deliver the glucose results and even eliminating the need to record manually the results obtained by the Glucometer.

4.1 Service definition

GlucoServer is a system designed to be offered to hospitals as a service aimed to improve and give added value to the monitoring and tracking service of patients with diabetes.

Therefore, in order to contract the service and apply it to their service providing logistic, hospitals would have to pay a fee proportional to the number of patients who were using this service.

The payment of this fee would mean the provisioning of all necessary measurement devices that we will discuss below, and subsequently all patient information storage and remote access to this information from doctors' computers through an ftp connection. Of course, all doctors will be instructed at a distance by GlucoServer employees in the use of the programs necessary to access to this patients information.

The servers responsible of storing all patient information will be all found in one place, owned by GlucoServer, and therefore each hospital would have access only to reports and results of patients treated in its center. Thus we get that all maintenance tasks are carried by GlucoServer and we get to avoid hospital equipment installation and maintenance work.

The application for Android smartphones, called 'GlucoServer 1.0', necessary for the proper functioning of the system would be available in the Market for free, a fact that doesn't mean any added expense to the hospital.

4.2 Potential Market

Diabetes is a problem that nowadays affects millions of people around the world and the medical specialty that takes care of his study is endocrinology. Thus, when looking for potential customers of our service we have to set only in hospitals that have an endocrinology department. Is still common to see hospitals where there is not any department dedicated to this disease, and its treatment and monitoring is delegated to primary care physicians. However, it is increasingly common to see hospitals adapting to the demand created by the growth of chronic diseases like diabetes.

If we start analyzing the situation in Paris, we see that from 58 existing hospitals in the city, 7 of them have endocrinology dedicated departments and services [22]. However, if we expand the search to the entire country of France, we see how the potential market is expanded to 750 hospitals [30].

'GlucoServer 1.0' application has been completely designed in English mostly because it is intended to be used globally. This point leads us to believe that if we extrapolate the case of France to the entire world, we get a very high potential market, although only developed countries can be considered, due to the technologies required by our system for its proper functioning.

After analyzing our potential market, it is important to analyze also the potential market that hospitals providing this service will have, because our income will also depend on the number of patients using the service.

As we have said several times, for the proper functioning of the system is necessary for patients to install an application on a smartphone with bluetooth connectivity and Internet access. Taking the case of the United States, recent studies by Nielsen show that in 2011 the number of smartphones will overtake current phones[36]. This news show us that despite the potential market is reduced by approximately 50%, it remains very large and the forecast is for growth in the coming years.

Finally, but still in the context of smartphones, 'GlucoServer 1.0' application has been designed for smartphones with Android operating system, because today is the operating system with more users over iPhone and Blackberry. However, once the system is designed and the application developed, it would not be difficult to translate it to other operating systems for a more global use and therefore less loss of potential market.

4.3 Competence Analysis

As we have seen, 'GlucoServer' creates an infrastructure to transmit, in the most practical way for the patient, information from the glucometer to the doctor. However, this system has a distinguishing feature: it is conceived as a service addressed to hospitals.

In this context I have not been able to find any service that could represent a direct competitor of ours. However, we have to analyze if there are existing patents protecting any of the technologies or ideas used in our system, or any project that could represent an indirect competence.

4.3.1 Patents

The term patent usually refers to an exclusive right granted to anyone who invents any new, useful, and non-obvious process, machine, article of manufacture, or composition of matter, or any new and useful improvement thereof, and claims that right in a formal patent application.

Today, there are many search engines to find patents, but one of the best free tools is Google Patents. Using this tool I have been able to find some patents related to our system.

One of the most related patents that I have found is **'Remote Health Monitoring System'**, by Stephen J. Brown, Gowthaman Gunabushanam (US 7399276 B1). As we can read in their abstract: 'The invention relates generally to remote health monitoring systems, as applied to the field of public health surveillance. In particular, it relates to a multi-user remote health monitoring system that is capable of reliably identifying and collecting data from frontline healthcare providers, laboratory and hospital information systems, patients and healthy individuals in a number of ways, with a view to aid in the field of public health. The system can also be used to query and collect additional information regarding specifics pertaining to the health of the individuals, and for patient tracking, monitoring, and the collection of individual data'.

This invention is related to ours since it creates a network to collect data from multiple users. The main difference is that our system is focused on collecting monitoring data only from patients. Moreover, this idea explains the concept of the network necessary to transmit the information, but not how to implement the user-side in order to send this information to the server.

Another related patent is **'Remote Health Monitoring and Maintenance System'**, by Stephen J. Brown (US 7624028 B1). He defines his idea in the abstract: 'A system and method is described that enables a health care provider to monitor and manage a health condition of a patient. The system includes a health care provider apparatus operated by a health care provider and a remotely programmable patient apparatus that is operated by a patient. The health care provider develops a script program using the health care provider apparatus and then sends the script program to a remotely programmable patient apparatus through a communication network such as the World Wide Web. The script program is a computer-executable patient protocol that provides information to the patient about the patients health condition and that interactively monitors the patient health condition by asking the patient questions and by receiving answers to those questions. The answers to these health related questions are then forwarded

as patient data from the remotely programmable patient apparatus to the health care provider apparatus through the communication network. The patient data may also include information supplied by a physiological monitoring device such as a blood glucose monitor that is connected to the remotely programmable patient apparatus. When the patient data arrives at the health care provider apparatus, the patient data is processed for further management of the patients health condition by the health care provider, such as forwarding another script program to the remotely programmable patient apparatus'.

This patent is not as general as the first one and explains more carefully all the method and the system to transmit the information to the healthcare provider. However, our system solves some problems related to this patent such as the need to have multiple extra devices and a computer. In our system we are using a Smartphone, which enables us to unify all the features of this system in just one device.

4.3.2 Indirect competence

I have already explained that our system is unique in the context of a service addressed to hospitals. However, there are some products that could represent an indirect competence to us due to their characteristics.

The second company that could be a competitor is **BodyTel** with its product **GlucoTel.** This product is a glucometer which carries a built-in Bluetooth wireless technology module which enables the patient to automatically transmit measured body values to a secure internet database using the phone or the computer as a transmission hub.

While the transmission from the device to the cell phone is done via Bluetooth wireless technology, the transmission from the cell phone to the internet database is done by mobile internet connection.

The patient is the owner of their values and only they have the authority to grant access to other people such as healthcare professionals, caregivers or family members. Those who are granted access by the patient also have to give their consent before they can access the BodyTel database with their own login account.

Another company offering this kind of product is **MyGlucoHealth**. The concept is the same as the previous one, as the patients are normally the only ones who have granted access to all the information stored in the database.

However, with MyGlucoHealth, nobody else will have access to this information and the patients are responsible for sending the information they want to their caregivers, family or friends via e-mail or SMS, depending on their choice.

4.4 Arquitecture and performance of GlucoServer

GlucoServer's system considers three main parts: the glucometer, the smartphone and the data server. In Figure 4.1 we can see a diagram of the proposed system.

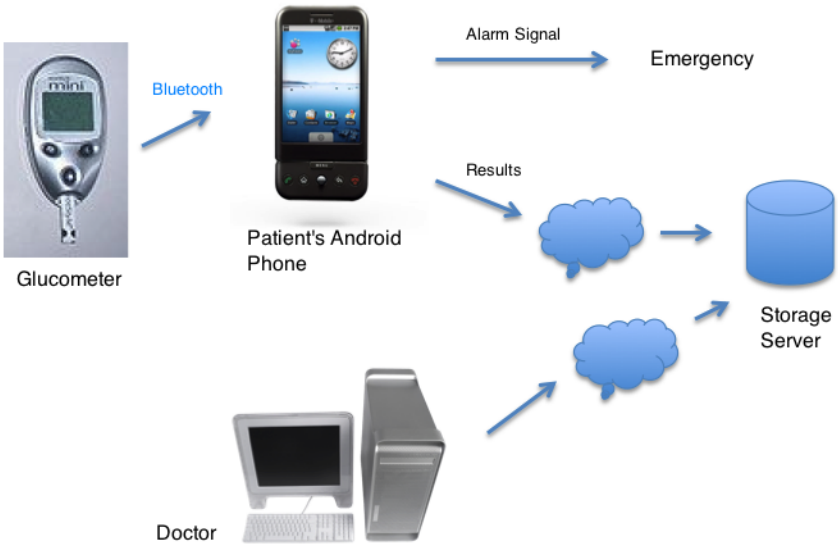

Figure 4.1: Architecture of GlucoServer

In this system all the process starts with the glucometer, responsible of measuring the patients glucose level and immediately sending the result to the smartphone via bluetooth. Once in the smartphone, the application 'Glucosever 1.0' is responsible of processing the data received and deciding if the result is dangerous for the patient or not, taking into account the range of accepted results previously set by the patient . If the result is correct and safe for the patient it will be sent to the data server attached to the name and the numeric identifier of the patient for its correct storage. If not, it will be stored as well in the server but also an alarm SMS will be sent to the emergency medical services number containing the glucose level and the localisation of the patient.

Finally, the doctor should be able to consult all of his patients' results stored in the server. He will only have to search by name or identifier.

4.4.1 Device: Glucometer

A glucose meter or glucometer is a medical device for determining the approximate concentration of glucose in the blood. It is a key element of home blood glucose monitoring by people with diabetes mellitus or hypoglycemia. A small drop of blood, obtained by pricking the skin with a lancet, is placed on a disposable test strip that the meter reads and uses to calculate the blood glucose level. The meter then displays the level in mg/dl or mmol/l, depending on the preferences of the user.

In order for our glucometer to be able to communicate with the system it must be a device enabled with Bluetooth capabilities. Three commercial brands with bluetooth devices are coexisting nowadays in the market: Body Tel, myglucohealth and TysonBio. We have already mentioned the two first brands in the environment analysis and, as we have seen, they are completing their product with the possibility of storing all the results in a database creating an account through its website, and the possibility as well of sending alarm messages to familiars or close people we decide.

The third brand (TysonBio) is offering also a management platform with automatic transferring of data to PC and the possibility to create customizable reports to share with the doctor.

We could choose any of this three enterprises to negotiate about the selling of the devices and their operating protocols. However, TysonBio is the one offering the service that differs the most from ours. Moreover, is a company more focused in the devices than in the collateral services. For this reason, in our system we will use the model AC100 from ThysonByo, with bluetooth capabilities.

4.4.2 Server

The server is where all the patients' information will be stored. As we have already explained, the name and the numeric identifier of the patient will be attached to all the results sent, in order to store them orderly.

The doctors will have access to the patients' information through an FTP connection to the Server. They will just have to search by the name and the identifier of the desired patient and the corresponding directory will appear to consult it. Once results have been consulted, doctor will decide either to erase or keep them for future reference.

Data Protection

As the Servers will contain confidential data and information about the patients of the hospitals and this information will be travelling from the devices, we will have to follow the code of practice for information security management ISO 27002:2005. This is the standard for information security that follows the standards published in the Code of Good Practice for the Management of Information Security of the national publication UNE-ISO/IEC 17799.

This standard specifies control objectives to protect information assets against threats to their confidentiality, integrity and availability. It defines guidelines related to human resources security, physical and environmental security, communications and operations management and access control, among others.

4.4.3 Application: GlucoServer 1.0

The application is maybe the most important part of the system because it is the intermediary between the patient and the doctor. It will be the responsible of processing the data received from the glucometer and send it to the server but also it is meant to be a tool to facilitate communication between the patient and his doctor.

GlucoServer 1.0 will be available for all smartphones running Android operating system. However, only smatrphones running 2.0 or newer versions will have full access to all the features due to compatibility issues with bluetooth in older versions.

Getting started

When we execute 'GlucoServer 1.0' for the first time in our smartphone we will see the main window corresponding to Figure 4.2. As we can see this windows is composed of the application title and three buttons: 'View Results', 'Contact Doctor' and 'Preferences and Info'. Pressing any of this buttons we will have access to different features of the application which, one by one, will be carefully explained later.

Moreover, if we press the menu button in our device, an menu showing three more options will appear (Figure 4.3):

- About: on pressing this button a dialog window with information about the application and the developer will appear.

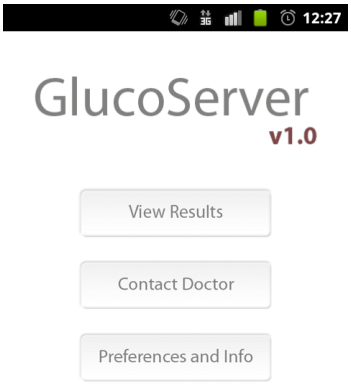

Figure 4.2: Screen capture: Main menu

- Scan: this option will activate the background process responsible of managing the connection with the glucometer. It will be explained more carefully the following sections.
- Exit: this option will turn off the application and any process running in the background.

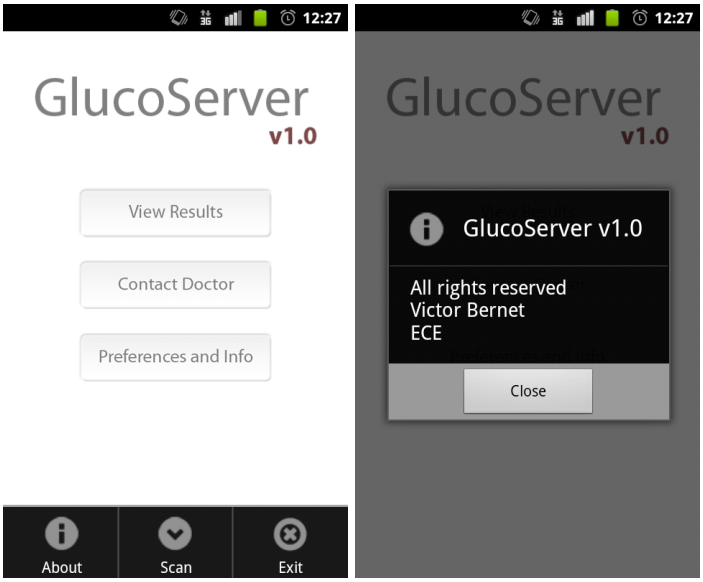

Figure 4.3: Screen captures: Main window options menu, 'About'

Setting up Preferences

The first thing to do after installing and executing 'GlucoServer 1.0' for the first time is setting all the preferences and filling all the information required for the propper functioning of the application. Until this is completely done, a remembrance message will appear each time we try to explore the features or menus shown in the main window and we wont be able to use any of them.

The preferences that we have to set are divided into two main groups: personal information related to the patient and doctor's information. The information that the application needs to know about the patient is:

- Name: the patient will have to introduce his/her name and surname in order to attach them to the results sent to the server.
- Identifier: the numeric identifier of the patient that will travel attached as well to each result.
- Max and Min Glucose Level: the patient will set the range of glucose level in which he or his doctor considers there's no risk. This will be the guide for the application in order to send or not the alarm SMS when a result from the glucometer is received and processed.

The second part of the information that needs to be filled before start using the application is about the doctor. In that part the user will be requested for the e-mail and the phone number of the doctor.The application needs this information for the already mentioned 'Contact Doctor' feature, that we will explain later.

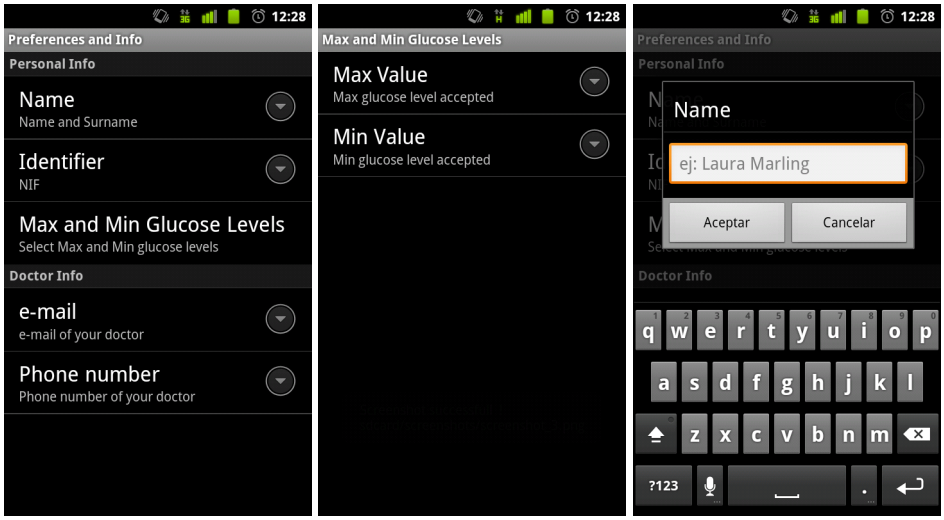

Figure 4.4: Screen captures: 'Preferences and Info'

Once all the camps are filled, when the user returns back to the main window, his name and identifier will appear below the three buttons as shown in the following screen capture. Moreover, he will be already able to start exploring the application and using all the features.

Figure 4.5: Screen capture: Main window with user information

Scanning for results

As we have already mentioned, the main function of this application is processing the information received from the glucometer, sending it to the server for its storage and in some cases sending as well a message to the emergency system. For this purpose, fist of all we have to initiate the communication between the glucometer and our device.

To do this, the first action required is going to the main window and pressing the Scan button, located in the options menu, as we can see in Figure 4.2. Once selected, if bluetooth is not active in our smartphone, a dialog window will appear asking for permission to activate it. Moreover, if we have not previously paired both devices, a second dialog window will appear asking for permission too.

When bluetooth is already active and our devices properly paired, an icon will be added to the system's status bar and an expanded message, under the 'Ongoing' title, to the 'Notifications' window (Figure 4.6), informing us that the connection process has been successful and our application is already waiting for receiving results from the glucometer. This notification will remain active all the time in order to inform us that the process responsible of managing our connection is still running in the background even when the application is not visible because we are using other applications or just not interacting with the phone. In order to return to the application we will just have to press the expanded message in the 'notifications' window.

From this time on, each time that a measurement is made by the glucometer and received

by the phone, a notification icon will be added to the status bar and a message detailing the result of the measurement will be shown in the notifications window as well (Figure 4.6). This messages could be deleted unlike the ones informing about the process running in the background, and even pressing them we will be taken to the 'View Results' feature explained below.

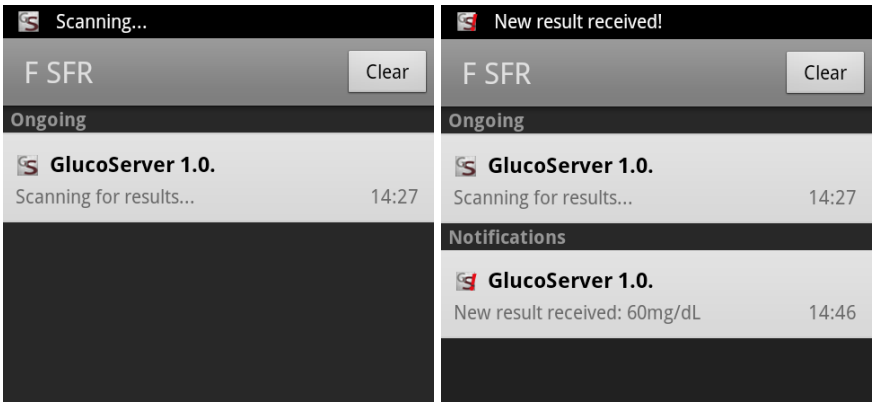

Figure 4.6: Screen capture: 'Scanning...' and 'New result received!' notifications

All the results received will be processed, saved in the phone memory and sent to the server for their storage. Moreover, all the results will be also compared to the maximum and minimum values set by the patient in the preferences menu. If any result is found outside this range of correct values, the application will conclude that this value is unsafe for the patient and will compose a message, informing about the glucose level of the patient and his location, that will be show to the patient before being sent to the emergency medical services. The fact of letting the patient see the message and choose whether to send the alarm or not is just to avoid false alarms or cases in which the patient can quickly react and solve the situation without the participation of medical service.

Finally, if we want to cut the communication with the glucometer off for any reason, we can turn this process off just by pressing the 'Exit' button located in the options menu of the main window.

View Results

When we press the 'View Results' button in the main process we access a list view of all the results stored in the smarthpone. If there are no results stored in the memory the application will tell us 'No results to show...'. However, if there are one or more we will see a list of all the results starting by the latest stored one.

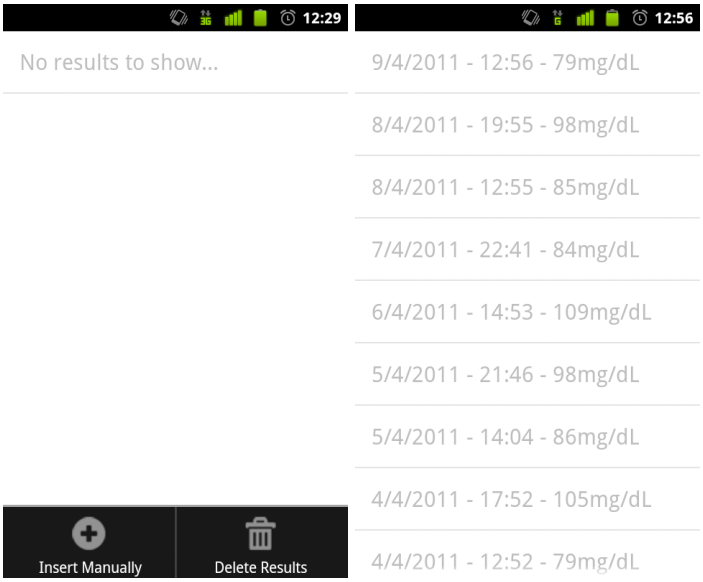

Figure 4.7: Screen capture: 'View results' options menu, result list

If we press the device menu button the options menu will appear with two items:

- Insert Manually: It permits us to introduce results manually instead of receiving them from our glucometer. It is important to have that choice in order to be able to send our glucose measurements to the Server even if we don't have our bluetooth device by hand. In Figure 4.8 we can see the screen capture corresponding to this window. As we can see time and date will be set to the actual values. Although we can set different values just pressing the 'Set' button. Also we can choose the units of measurement: mg/dL or mmol/L. Moreover, if we introduce a critical value, the application will as well give us the option of sending the alarm SMS to the emergency service.

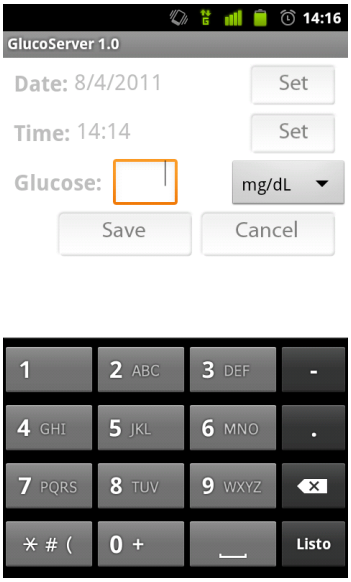

Figure 4.8: Scree capture: 'Insert Manually' feature

- Delete results: This option erases all the results stored in the memory of the phone, but not the ones stored in the Server. That means that there's no problem on doing this when we have memory troubles because none of our results will be lost to the doctor's view.

Contact Doctor

The last feature of the app called 'Contact Doctor' tries to take to the simplest the way patients contact their doctors when they have a doubt or just to inform them about something related to their health issue.

When we press the corresponding button in the main window we access to this feature consting only on a simple view with three buttons as we can see in Figure 4.9. Just pressing the button corresponding to the kind of communication we preffer, we will be redirected to the related app. It is now when the information we filled about the doctor in the preferences menu takes sense.

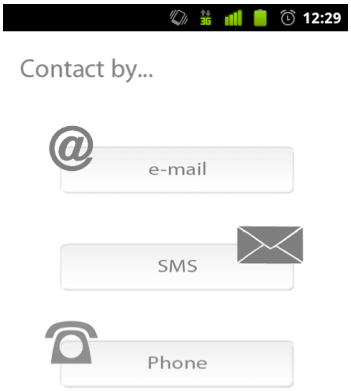

Figure 4.9: Screen capture: 'Contact Doctor'

The three options the user can choose from are:

- e-mail: when we press this button we will be redirected to the e-mail application where the address and subject will be already filled. We will just have to fill the message and send it.
- SMS: the users will be redirected to their SMS application where they will only have to fill the message as well.
- Phone: when the user presses this button the application will immediately initiate the phone call using the phone native app and the doctor's phone number.

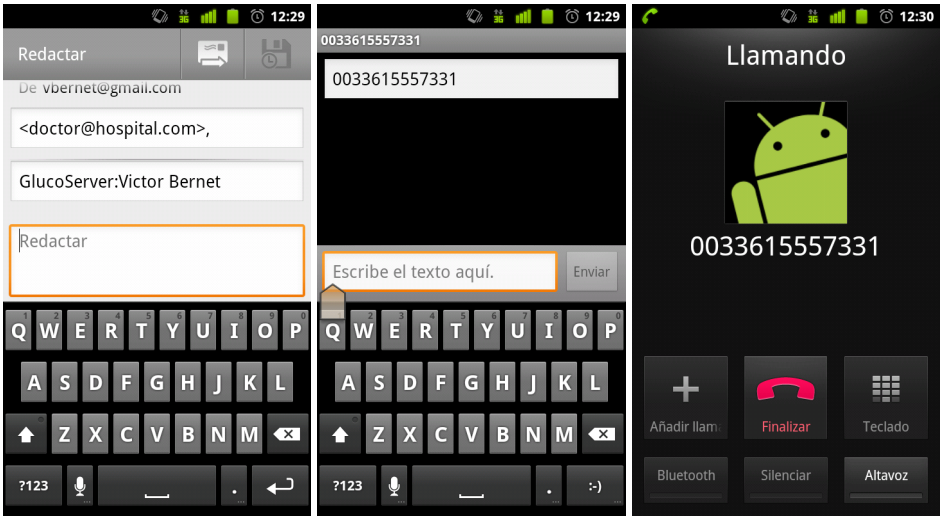

Figure 4.10: Screen captures: e-mail, SMS and Phone

4.5 SWOT

When launching any product or service to market, all the pros and cons that it may have must be analyzed in order to try to predict its future. To analyze these items there is a tool called SWOT, which stands for Strengths, Weaknesses, Opportunities and Threats.

Thus, GlucoServer has been analyzed using the SWOT tool to know its prevailing situation in the market and the real potential to gain a foothold in the market for tools related to health monitoring.

The SWOT analysis consists of two parts. In the first one, the product is analyzed from the standpoint of environment and therefore the opportunities or threats that offers. As we know, the success of any product or service shall be subject to a lesser or greater extent by the state of its environment, so we have to know how it is and how it evolves to know as well how our service will.

Opportunities are positive external situations that are generated in the same environment and that, once identified, can be exploited:

- The use of ICT is getting more and more popular and people are beginning to adapt preferring to do things at a distance.
- Although GlucoServer is intended to be a global product, the economy in the country where it will be initially implemented (France) has withstood the crisis and is picking up faster than other countries.
- eHealth market is a comparatively new concept and so much remains to be exploited.
- The number of diabetic patients is increasing and in consequence hospitals are giving greater attention to this disease creating endocrinology departments.
- The smartphone market is expanding leaving behind the conventional phone.

Threats are negative external situations that may threaten our service and in that case may be necessary to create a strategy to overcome them:

- Big worldwide companies like CISCO, SIEMENS or Microsoft are taking an interest in the eHealth market.
- Promotion is difficult in this kind of markets.

Once the external analysis is done we move on to the internal analysis. This analysis will allow us to set which attributes would enable our product to generate a competitive advantage over the rest of its competitors. Internal analysis is divided into strengths and weaknesses.

Strengths are all internal and positive elements that differentiate GlucoServer from other competitors:

- GlucoServer is the only company offering this service to hospitals.
- GlucoServer cost is low compared to the revenues it can produce.
- The application 'GlucoServer 1.0' has a simple and easy to understand interface.
- 'GlucoServer 1.0' will be truly accessible since it will be in the Android Market for its free download.
- The fact of having all the servers in one place makes maintenance much easier.
- No extra devices are needed, just the glucometer and the phone.

Weaknesses are internal problems that once identified could be eliminated by developing a good strategy:

- We depend on TysonBio as it will be our glucose meter devices supplier.
- Doctors should be trained in the use of software to access the server.
- At the beginning, 'GlucoServer 1.0' will only be accessible to users who own an Android smartphone.
- The fact that 'GlucoServer 1.0' is developed in English can be a hindrance for the implementation of the service in France, despite is relatively easy to translate it.
- The product will not be known in the market at the beginning.

4.6 Innovation in GlucoServer

GlucoServer has been created, as already mentioned, as a service addressed to hospitals for being used by these as a tool to innovate. However, the service created, aside from being able to be used as a tool for innovation, can be understood as an innovation itself as long as we are providing hospitals a new service that nobody else is providing.

As seen above, there are some similar systems on the market speaking in terms of technology. However, all these systems are focused on a particular use only by the user who purchases the product and the transmission and storage service attached. In our case, we could say that the service offered is a new or significantly improved service in terms of the use to which it is intended and also in terms of implemented technologies as we are expanding the concept to its use in smartphones, giving the patient greater freedom of movement and continued assistance as long as it carries over its smartphone.

Therefore, referring to the definition given by the Oslo Manual, GlucoServer would be a product innovation in services.

Moreover, we can also classify our product innovation depending on how it happens. In this case it would be a radical innovation, since this service offer focused on hospitals haven't existed till nowadays and neither its technical specifications for serving it.

4.6.1 GlucoServer as an innovation tool

As previously mentioned, GlucoServer is, in addition to being an innovative service Itself, a tool that will help hospitals which decide to hire its service to innovate as well in the services they provide. Specially, GlucoServer will make them innovate in the services of care and monitoring of patients with diabetes.

Any innovation in the services that these hospitals provide will start with the redefinition of one or more of the necessary dimensions to provide a service discussed in chapter 2. Is in this redefining task where GlucoServer comes into play as thanks to its features it can allow them to innovate in more than one dimension at once, bringing new added value to initial service.

GlucoServer provides hospitals with a **new infrastructure** for relishing their services to patients with diabetes. In particular, it is a new technological infrastructure that uses the Internet and new technologies to deliver the outcomes of patients to the doctor and also facilitate communication between them. Moreover, the use of a new infrastructure for providing a service can become an essential tool for innovation in other dimensions as discussed below.

On the one hand, GlucoServer will allow hospitals to innovate in the **interface with the client**, allowing monitoring and diagnosis of patients at a distance without direct contact with the doctor. Among other things, this will improve the time of diagnosis, because the results are sent to the server immediately after each measurement. Moreover, it will improve the flexibility of the service, allowing the patient to send the results from home and without any extra effort than to make the measurement of glucose with the glucometer provided.

On the other hand, our system can help hospitals to innovate in their **services production processes**, improving efficiency and effectiveness of service provided. GlucoServer allow the introduction of improvements such as a reduction in the cost of services because it does not depend on physical visits, which are often expensive and sometimes a waste of time for both the patient and the doctor. Also, allow greater service flexibility, allowing the customer not to depend on the hospital at the time of sending the results or visiting the doctor. Moreover, the fact of having all the results in a database accessible by the doctors can be a good help for a better diagnosis since more than one doctor can access to the same data at the same time at a distance to discuss the case.

5 Development of 'GlucoServer 1.0' App

5.1 Introduction to Mobile Development

A mobile platform, is the operating system that controls a mobile device or information appliance. We could compare them to an operating system such as Windows, Mac OS or Linux that controls a desktop computer or laptop. However, they are currently simpler, and deal more with the wireless versions of broadband and local connectivity and mobile multimedia formats.

Typical examples of devices running a mobile operating system are smartphones, personal digital assistants (PDAs), tablet computers and information appliances, or what are sometimes referred to as smart devices, which may also include embedded systems, or other mobile devices and wireless devices.

There is a wide selection of platforms with which you can realize your mobile vision. So first of all it would be good to introduce the most common environments and outline their differences. Then, a more detailed description of the Android Platform will follow.

5.1.1 Overview of Mobile Platforms

There are many mobile platforms used in the market; some are open source, some are not. Also we could make a difference between the native application platforms and others like Java ME, Flash compatible Platforms or BREW. The main benefits of programming apps natively include better integration with the platform's features as well as better performance.

One of the most typical drawback for developers are the effort and complexity of supporting several native platforms, so we have to decide if we can handle it or just limiting our app to one platform, what will reduce our market but will as well let us focus and improve our app to the best.

The increasing importance of mobile devices has triggered intense competition amongst software giants such as Google, Microsoft, and Apple, as well as mobile industry leaders Nokia, Research In Motion (RIM), and Palm, in a bid to capture the largest market

share pre-emptively. There have been many attempts in the past 10 years to change the direction of the mobile device market but it was with the release of the iPhone in 2007 that Apple significantly disrupted the mobile industry and effectively ushered in a new era of smartphone operating systems that focus on user experience and rely on touch-based interaction.

It was in November of the same year that Google formed the Open Handset Alliance with 79 other hardware, software, and telecom companies to make inroads into the smartphone market through its new Android operating system. Though its reception was mainly positive from the media and public, the release of Android created a rift between Apple and Google, eventually leading to the resignation of Google's CEO, Eric Schmidt, from Apple's board of directors. Since the launch of both Apple's iOS and Google's Android, the smartphone market has exploded in popularity and in 2010, accounted for more than 17.3% of all mobile phones sold[14]. Although it may seem a not so high rate, by 2015, the specialists predicts the percentage to be 60%.[31] This has led to greater consumer awareness of the various mobile operating systems, with telecoms and manufacturers regularly advertising the advantages of their OS.

But apart from these two giants we find many other enterprises trying to have their piece of the cake on the Smartphone Market and some of them are starting to become more and more popular. In the following table, we can see a summary of the most important existing platforms nowadays:

Platform	Company	Language(s)	Remarks
Android	Open Handset Alliange (Google)	Java, C, C++	Open Source OS (based on Linux)
Bada	Samsung	C, C++	Samsung's mobile platform running on Linux or RealTime OS
BlackBerry	RIM	Java	J2ME compatible, extensions enable tighter integration
iOS	Apple	Objective-C, C	Requires Apple Developer Account
MeeGo	Linux Foundation	Qt, WebApps, C++, others	Intel and Nokia guided open source OS (based on Linux)
Symbian	Symbian Foundation	C, C++, Java, Qt, WebApps, others	Open source OS built from the ground up for mobile devices
webOS	Hp/Palm	HTML, CSS, JavaScript, C	Supports widget style programming. Based on Linux
Windows Mobile	Microsoft	C#, C	.NET CF or Windows Mobile API, most devices ship with J2ME compatible JVM
Windows Phone	Microsoft	C#, VB.NET	Silverlight, XNA frameworks

Table 5.1: Most important Mobile Platforms on 2011

5.1.2 Market Situation

The first thing we have to do when speaking about Mobile Platforms and their Market Situation is to distinguish between the USA Market and the worldwide one. This is just because of one name: Symbian.

Going hand by hand with Nokia from the very beginning they accomplished to be the one of the first Mobile Platform worldwidely, but something very different happens if we take a look at the situation only in the USA, where this Operating System is almost nonexisting.

As we can see in Figure 5.1, during the last quatrimester of 2010 in the worldwide market Symbian was the second Operating System with sales of 31 million devices, in front of the 33,3 million sold by Android. In the third and fourth positions we could find iOS and Blackberry with a very close percentage of Market Share around the 15% each. Following these big giants, but far from them, we can find Windows Mobile with a 3% and

other Operating Systems as could be Windows Phone 7, MeeGo, etc. sharing the last 3%. [3]

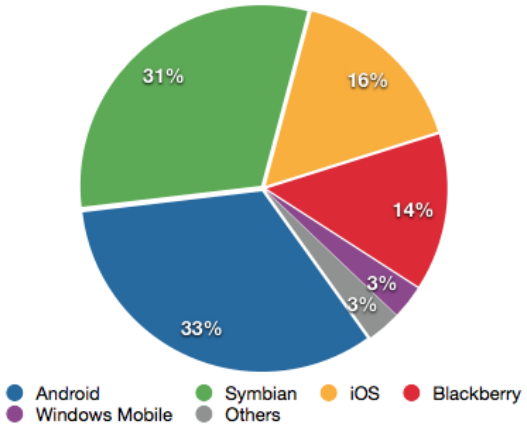

Figure 5.1: Worldwide Market Share 2010 Q4

But something different happens when we analyze the situation in the USA. As we can see, during the Q4 of 2010, the Mobile Operating System that registered the biggest sales amount is Android with a 29% of the market share, ahead of iOS and RIM, both with a 27%. [37]The three of them controlled the Smartphone Scene in the USA with an almost nonexistent Symbian, and WP7 and webOS still starting to grow due in part to their recent creation.

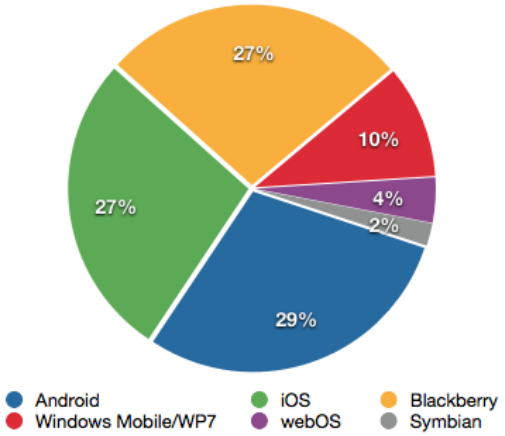

Figure 5.2: USA Market Share 2010 Q4

If we speak in terms of quick growth, Android is the clear winner both in worldwide and

USA markets, as we can appreciate in Figure 5.3. During last year, Android grew in the worldwide market from 8.7% in the Q4 of 2009 to a 33% in the same quatrimester of 2010 to get in the fist place. And it seems to continue like this according to IDC analysts who appreciated that Android will continue controlling the market with 45% in 2015[23]. Not the same good news for Symbian that in Q4 of 2009 was the winner with almost the half of the Mobile Platform Market Share and decayed to a 31% in just one year. The same happened to Blackberry which fell down in just one year from 20% to 14%.

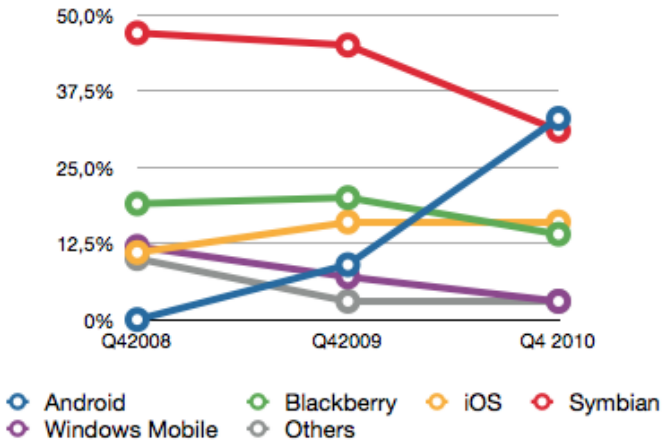

Figure 5.3: Progression of Market Share[2, 16]

5.2 Android Platform

The Android platform is developed by the Open Handset Alliance led by Google and publicly available since November 2007.

Android is an operating system and an application framework, mainly created for its use on smartphones, with a complete tooling support and a variety of preinstalled applications. In late 2010, Google announced that everyday more than 300.000 Android devices, including tablets, are shipped to end users[24]. As said before, since the platform is supported by many hardware manufacturers, it is the fastest growing smartphone operating system. Additionally, Android is used for media players, setup boxes, desktop phones and car entertainment systems.

Figure 5.4: Android Logo

5.2.1 Evolution

Android is also growing very fast when it comes to the platform features. Each update typically fix base operating system bugs and add new functions. One curiosity is that generally, each operating system upgrade is developed under a name related to desserts.

The first versions **1.0** and **1.1** where launched in September 2008 and February 2009 respectively. They were just the start and they couldn't still be called a complete operating system as we know it nowadays.

The first significant update came with **Android 1.5 'Cupcake'.** It was released in April 2009 and it came with a completely updated interface and many new features like the possibility of having folders and widgets in the main screens, the capability to record and play video or also a new keyboard with text prediction.

In September 2009 version **1.6 'Donut'** was released. It included a brand new camera and gallery integrated interface, an updated voice search and an improved search experience to look for contacts, bookmarks and even websites from the home screen. Moreover, gesture framework and developing tools for GestureBuilder or WVGA screens resolution support were some of the features brought by this version.

Only one month later Android **2.0 'Eclair'** was released with improved Google Maps, hardware speed and virtual keyboard. It also appeared with Microsoft Exchange support, new contacts list, integrated camera flash support, digital zoom, animated wallpapers and an updated user interface. One of the most important updates in this version, at least for this project, is the Bluetooth API appearance in the SDK.

In January 2010, and under the same dessert name (Eclair) the **2.1** SDK appeared and and with it the first Google Smartphone appeared. Its name was **Nexus One** and it was a collaboration between Google and the Taiwanese company HTC, which was the responsible of manufacturing the devices. Despite it was conceived as a device for developers,

it was sold in the Google On-line Store and agreements with some mobile operators were reached to sell it attached to a phone contract in USA and Europe.

Figure 5.5: Google Nexus One

Afterwards, in May 2010, **2.2 'Froyo'** version was released. Some people say that with this version Android started to be a real Mobile Platform and could be compared to other stable mobile platforms such as iOS. The improvements in this version were many, but most of them were related to the general optimization of the system, memory and performance. Some of them are the improvement of the application speed, thanks to the JIT implementation, the integration of JavaScript V8 in the Browser or an updated application launcher.

Moreover, this new version allowed the user to create a Wi-Fi Hotspot or doing USB tethering, to turn down the data traffic or to quickly switch between different keyboard languages and dictionaries.

Figure 5.6: Cupcake, Donut, Eclair, Froyo, Gingerbread and Honeycomb Logos

The last version of Android for smartphones for the time being is **2.3 'Gingerbread'** and it was released in December 2010. This version, as well as the previous one, brought us a lot of improvements and new features. One of the most innovative is the support for NFC (Near Field Communication). This technology will allow the user to pay in any commerce or shop only with its mobile phone and the first tests are already being made in the United States.

Other important updates of this version are a redesigned multi-touch keyboard, improved copy and paste tools, new sound effects such as equalization, native VoIP support, extra

wide and WXGA resolution screen support and a download manager. Also for developers some improvements such as better support for native code development or an improvement in the data, audio and graphics entry for game developers happened.

Moreover, with this version, the expected appearance of the second Google smartphone finally happened. It was named **Nexus S** and this time the collaboration was with Samsung, the manufacturer responsible for the Galaxy S, one of the most successful Android smartphones by now.

Figure 5.7: Google Nexus S

The last version released is Android **3.0 'Honeycomb'** and has been specially designed for its deployment on tablets and other devices with larger screens. The first device out in the market with this new Android version is **Motorola Xoom**, which has gone hand by hand with Google during its development. This version has a completely new user interface with a 3D desktop, redesigned widgets and native apps such as Google Talk, Gmail or Browser.

Figure 5.8: Motorola Xoom

5.2.2 Anatomy of an Android App

Android includes some new development concepts to tackle the problem of writing applications for devices with limited resources. It also includes some interesting techniques to make it possible to have a higher layer of integration between applications than other mobile platforms.

The most important development concept on Android are **Activities**, which are the graphical presentation layer for an Android application, and in the typical case there is one Activity for every screen. Usually the window that the Activity draws fills the screen, but it might be smaller and float on top of other windows. To show its content and to interact with the user an Activity uses a hierarchy of **Views** and **Layouts**.

One aspect that differentiates an Android Activity from the normal concept of a Window is its lifecycle, see Figure 5.9. The Activity lifecycle is designed around requirements and limitations of the mobile devices it is supposed to run on. The lack of possibility to swap Activities out of memory creates the need for an Activity to be able to recreate itself after being removed from memory. This is solved with Bundles to which Activities can store their necessary data when they get destroyed and read it back when they are recreated.

The second basic concept in Android programming is **Intents**. An Intent is a specification of what action a developer wants to perform. A simple example of this is a request such as "I want to start Activity A" but it can also be a more general request, for instance "I want to open this PDF file" or "I want to send a SMS". To listen to this more general intentions, application developers can set up Intent Filters that are designed to show the rest of the system what capabilities a certain Activity have.

The nice thing about this intents philosophy is that functionality can be executed by many applications and the Android system will use the preferred application for a specific intent. For example the intent of sharing a web page triggered by a news reader app can open an email client or a text messaging app depending on the user's preference and on the applications installed.

Intents is not limited to making request but can also broadcast information to the rest of the system. One example of this is the music player that broadcasts every time a new song is being played. Some of these broadcasts are standardized to help integration between applications. This enables a developer to for example build a home screen widget that downloads and shows the lyrics of the current song without actually knowing anything about the application playing the song. To listen for these kinds of Intents a developer can set up **Broadcast receivers** whose only purpose is to listen to and react to these kinds of broadcasts.

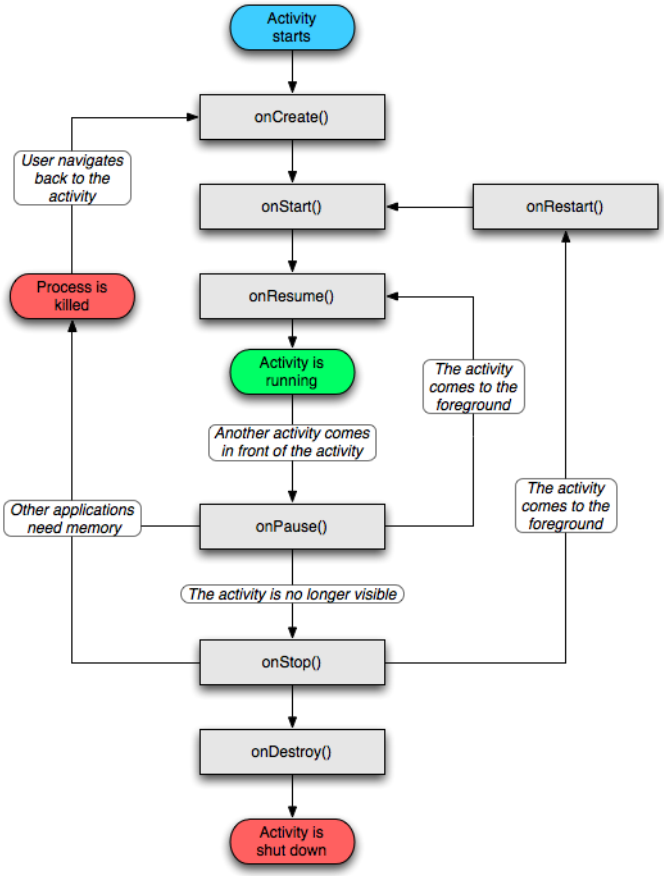

Figure 5.9: The Android Activity Lifecycle [7]

The Android platform is designed from the ground up to be multitasking. One big part of this is to be able to run several applications at once, but equally important is to be able to run backgrounds **Services**. Services do not have a visual user interface and are designed to run in the background for an indefinite period of time. Services can perform tasks such as play music or download data in the background. A Service can choose to communicate with the user either by a notification or by launching an Activity.

All Android applications run sandboxed in separate virtual machines, and one consequence of this security model is that applications can not share memory or files directly. Instead they use **Content providers** to let other applications access their data. The Content provider can indirectly give access to data from files, databases or memory. There is also a set of Content providers provided by system applications.

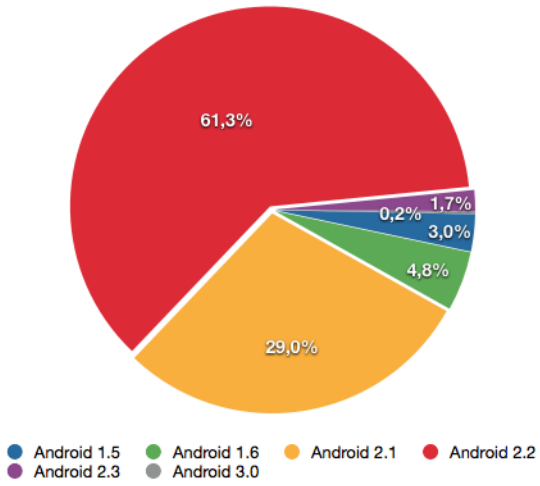

Figure 5.10: Android platform versions distribution [9]

5.2.3 The main deal: Fragmentation

The Android Platform has had an extra ordinary development pace since it launch. As we can see in Figure 5.9 still three of the first versions of this mobile operating system make up more than a quarter of the devices running Android, and this has created problems for developers who wants to create software using the latest features, but have been held back because they do not want to exclude many of the Android users.

One of the main reasons for this problem is that new releases have to go through the phone manufacturers modifications and quality assurance testing before sent out to the users phones. This has in the past often taken longer than users and developers have hoped for. The underlaying reason for this is that manufacturers has not been able to keep up with Google's release schedule, which is much faster than whats normal in the industry.

Google has applied some changes to the last releases of the Android platform, trying to solve this problem by transitioning many of the components and pre-installed applications to the Android Market. This started with the release of version 2.2 and the version after that, codenamed Gingerbread. This has already been done for example with the Google Maps application, and the result is that the users using earlier versions of Android can update to the same version as those running the latest version. Google has said that this process of moving components to the Android Market will not be limited to Applications but will also cover system components such as input methods.

Other important cause of fragmentation in the Android platform is related to the Hardware.

As we know, Android runs in lots of devices from many different manufacturers. This implies that the Apps must be prepared as well for working with lots of different screen sizes and resolutions. Each time that a new version of the platform is released, the support for different kind of screens is expanded but it's work for the Developer to take them into account.

The Fragments API is a tool made by Google to help developers to adapt more easily their applications to the different size and resolution screens. This tool could be the solution to the hardware fragmentation in Android.

Lately, Google also improved this API including the ability to easily "scale" applications within different versions from 1.6 forward. That means that from now on it will be easier for developers to make their applications work in old devices with low resolutions as well as in a 4.3" screen smartphone with high resolution.

To sum up, fragmentation is a big deal for developers but as well for users, as long as they are the final receiver of the applications developed. Moreover, it's sometimes hard to decide which device would be the best to meet our needs from a huge range of possible candidates, and this could bring the client to the decision of putting aside Android platform.

5.2.4 Distribution and Monetization

After creating the next killer application and testing it, a developer has to think about how to distribute and take profit of it. The main distribution channel on the Android platform is the official Android Market, which is distributed with almost all Android-running devices. On late December, 2010 the Android market reached the 200,000 app milestone [4].

A user needs to have a Google account to be able to use the Android Market, and in order to be allowed to publish applications in the Android market a developer needs an Android Market Developer account. This account is available to individuals and companies for a one-time fee of 25$ [18]. This gives them the ability to upload unlimited number of applications to the Android Market. Moreover, they can choose to distribute their applications for free or a price, and in the case of paid applications 70% of the price goes to the developer while Google keeps the other 30% [20]. Although, to receive the revenues made from the paid apps developers must create a Google Checkout account and link it to their developer account.

Android Market is nowadays available in 44 countries of which only 32 support paid applications, and only developers from 29 countries are allowed to publish paid applications

[19]. Google insists that they are working hard to add more countries to the list, but they are not really able to provide any guidance on timelines.

On February 2011 Google presented a new web client providing access to the market via PC[33]. With this, requested applications will directly be downloaded and installed on the registered Android device. This provides the user another way to buy apps from the official Android Market in addition to the traditional native application and makes it even easier to do it wherever they are.

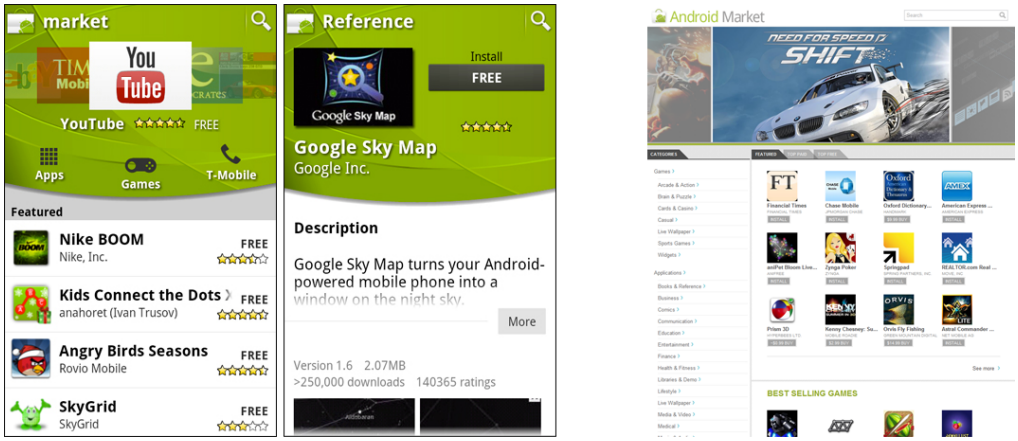

Figure 5.11: Captures of Market application and web client

On Android there is also the possibility to distribute applications outside the official Android Market. Developers can publish applications on their homepage and even send their applications directly to users. The possibility to install applications outside of the Android Market has spawned a range of different 3rd party markets.

One of the most interesting options is SlideME, which lets you browse its catalogue of apps on-line or via an installable app called SAM that runs the whole experience on your Android device. This market focuses on promoting the work of small, independent developers. One advantage of this is that it's a good place for publishers to scout out developer talent early, and app developers can be contacted and hired directly via SlideME [34].

Finally, on late March 2011 Amazon, the most important on-line shop all over the world, joined the android applications market launching its own Amazon AppStore [32]. Starting with 3800 apps, and despite being only available in the USA, they made an important incursion in this market counting with applications made by some of the most important development enterprises such as Handmark, Glu Mobile or even Gameloft, which until that moment only had sold its HD games on its own website, and launching some applications on exclusivity as Angry Birds RIO.

5.3 'GlucoServer 1.0' Project

Now that we have learned a bit about Mobile Platforms and more specifically about the platform we are going to develop for (Android), It is time to explain the process that I have followed to develop 'GlucoServer 1.0' from the very first steps.

In this section I try to summarize a bit just by discussing the most important or interesting facts to understand the development process. In order to see all the code of the Application, you can go to Annex number 1.

5.3.1 Development Tools

The main programming language for Android is Java. But beware, only a subset of the Java libraries is supported and there are lots of platform specific APIs. Android development is, as other development in the Java programming language, often done in a **development environment such as Eclipse**. It is possible to do development in other IDE's including Netbeans, but the official development environment is Eclipse. It is also the best choice as there is good support for development, deployment and especially, so-called library projects that allow to share code and resources between several projects. Eclipse is an open source development platform originally built by IBM in 2001 but is now maintained by the non-profit foundation The Eclipse Foundation and we can download it for free from it's website .

Once we have the correct version of Eclipse installed in our computer, to get started, it's necessary to have the **Android SDK**, which is available for Windows, Mac OS X, and Linux. It contains tools to build, test, debug and analyse applications. The Android SDK also has a different set of APIs from the normal Java environment.

Then, it is also highly recommended to download and install the **Google's Android Development Tools, ADT**. It is a set of Eclipse plugins made to ease Android development. ADT enables developers to create Android projects with all the necessary files and build configurations, as well as simplifying the process of building, testing and signing an application before release. In the day-to-day work the ADT plugin enables control of an attached device or emulator. These controls includes setting breakpoints, changing the capabilities of the device, logging, emulating positions or even incoming calls.

When developing for Android the developer can either test applications on an attached device or an emulator. In comparison with iPhone development, Android makes it very easy to test an application on different devices. Binaries can be distributed to any device for testing. Developers can also set a device in debug mode which enables access to the

debug console output of the device and to set breakpoints. Development can be done on any consumer or development device as long as it is set to debug mode, under Setting -> Applications -> Development -> USB debugging, and the Android USB driver is installed.

The Android project also provides an emulator, which is very different from the iPhone simulator in that it is a real hardware emulator. One of the big advantages of having a hardware emulator is that it gives developers the ability to run different builds of the operating system. It is for instance possible to run the real system images from device manufacturers such as HTC. This is very handy since a developer can see how an application performs on different customizations of Android.

Figure 5.12: Android 2.2 running on the emulator

The **emulator runs an Android Virtual Device, AVD**, which is a profile that defines the hardware and software capabilities of an emulated device. Software wise a developer can specify any downloaded SDK version or any other device image that he or she might want to test on. A developer can specify if the virtual device should have the Google libraries such as maps installed or not, as well as specifying the device RAM size, hardware keyboard support, camera support, GPS support, screen resolution and screen density. The AVDs are all managed through the ADT plugin in Eclipse that we have already mentioned.

5.3.2 Creating the Project

To create an Android application, we will need to create a corresponding Android project. The project will hold all of our source code, resources, third-party JARs, and related materials. The Android build tools will turn the contents of our project into an Android package file (.apk), which is the Android application. Those tools will also help us get our .apk file onto an Android emulator or an actual Android device for testing purposes, as we have already mentioned.

When creating our project, we will need to indicate to Android the API level with which we are working. The API level is a simple integer that maps to an Android version; for example, API level 3 means Android 1.5. When creating a project, we will be able to tell Android the minimum and maximum API levels our application supports. In our case, we will develop our application in API level 8 (Android 2.2 'Froyo'). However, our app will work also in some older versions with bluetooth API (from 2.0 on) and newer versions as well (Figure ?).

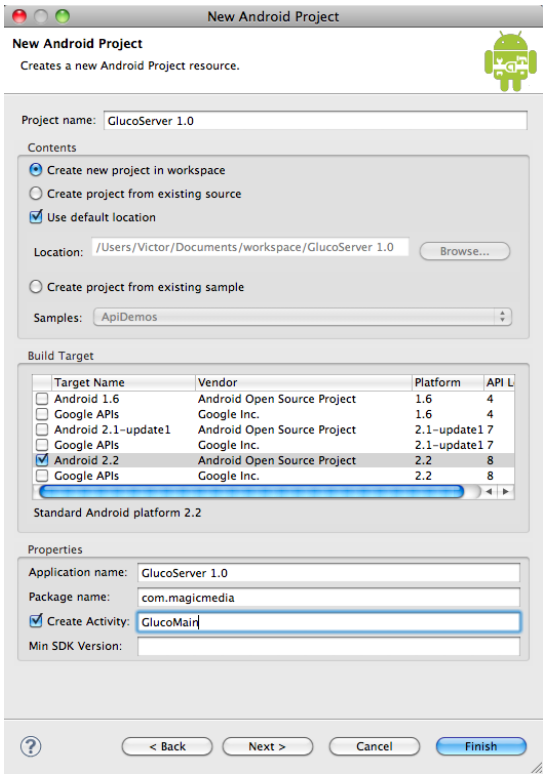

Figure 5.13: New-project wizard

Root Contents

Once we have created the new Android project using the 'new-project wizard', we proceed to analyze its structure. The Android build system is organized around a specific directory tree structure for the Android project, much like any other Java project. The specifics, though, are fairly unique to Android.

When you create a new Android project , we get several items in the project's root directory, including the following:

- AndroidManifest.xml: An XML file describing the application being built and which components (activities, services, and so on) are being supplied by that application.
- build.xml: An Ant script for compiling the application and installing it on the device.
- default.properties and local.properties: Property files used by the Ant build script.
- assets/: A folder that holds other static files you wish packaged with the application for deployment onto the device.
- bin/: A folder that holds the application once it is compiled (.apk).
- gen/: Where Android's build tools will place source code that they generate.
- libs/: A folder that holds any third-party JARs our application requires.
- src/: A folder that holds the Java source code for the application. Here is where we will save all the code for our activities and our service.
- res/: A folder that holds resources—such as icons, graphic user interface (GUI) layouts, and the like—that are packaged with the compiled Java in the application.
- tests/: A folder that holds an entirely separate Android project used for testing the one we create.

To develop our App, we will work over the AndroidManifest.xml and the src/ and res/ folders as we can see below.

5.3.3 Android Manifest

Every application must have an AndroidManifest.xml file in its root directory. The manifest presents essential information about the application to the Android system, information the system must have before it can run any of the application's code. Some of this information could be the activities or services that the application is composed of. Moreover, we can define also permissions the application must have in order to access protected parts of the API, interact with other applications and some other things.

In order to understand its structure we are going to analyze the AndroidManifest.xml corresponding to 'GlucoServer 1.0' app, explaining each of its elements.

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

        <uses-sdk minSdkVersion="5" />

        <uses-permission android:name="android.permission.CALL_PHONE"/>
        <uses-permission android:name="android.permission.BLUETOOTH" />
        <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
        <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
        <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
        <uses-permission android:name="android.permission.INTERNET"/>

        <application android:icon="@drawable/icon" android:label="@string/app_name">
                <activity android:name=".GlucoMain"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".Viewresults"></activity>
                <activity android:name=".Preferences"></activity>
                <activity android:name=".ContactDoc"></activity>
                <activity android:name=".ManualInsertResult"></activity>
                <service android:name=".BlueService"></service>
        </application>

</manifest>
```

<manifest>: This is the main element, like the root of the structure since all the other elements will be declared within it. It must be present in the manifest and can occur only once. In our case we declare attributes corresponding to the package, the version code, and the version name.

- **<uses-sdk>:** Lets us express an application's compatibility with one or more versions of the Android platform, by means of an API Level integer. In our case we will declare only the attribute referring to the minimum API Level and we will set it to 5, corresponding to Android 2.0, due to bluetooth issues in lower versions.
- **<uses-permission>:** This element will request to the user a permission that the application must be granted in order for it to operate correctly. As we can see in the code, we declare multiple permissions for our app to work properly as location,

bluetooth and ability to place phone calls or access the internet.

- **<application>:** This is the second most important element after <manifest>. It also must be present and can only occur once. This element contains subelements that declare each of the application's components like activities, services, content providers and so on. Moreover, it has attributes that can affect all the components. In our case we declare only the attributes corresponding to the label and the icon we will see on the launcher.
  - **<activity>:** Declares an activity that implements part of the application's visual user interface. All activities must be represented by elements in the manifest file. Any that are not declared there, will not be seen by the system and will never be run. In all the activities we have to declare the name attribute.
    - **<intent-filter>:** This element that in our case is only declared within the main activity specifies the types of intents that an activity, service, or broadcast receiver can respond to. In our case when we first create a project <action> and <category> will be declared.
  - **<service>:** Declares a service as one of the application's components. Unlike activities, services lack a visual user interface. They're used to implement long-running background operations or a rich communications API that can be called by other applications. As well as activities all services must be represented by <service> elements in the manifest file. Any that are not declared there will not be seen by the system and will never be run.

There are many more elements and attributes that we can declare on the manifest. However, these we have studied are the necessary for the proper functioning of our application.

5.3.4 Designing the interface

In an Android application, the user interface is built using View and ViewGroup objects. There are many types of views and view groups, each of which is a descendant of the View class.

View objects are the basic units of user interface expression on the Android platform. The View class serves as the base for subclasses called "**widgets**", which offer fully implemented UI objects, like text fields and buttons. The ViewGroup class serves as the base for subclasses called "**layouts**", which offer different kinds of layout architecture, like linear, tabular and relative.

A View object is a data structure whose properties store the layout parameters and content for a specific rectangular area of the screen. A View object handles its own measurement, layout, drawing, focus change, scrolling, and key/gesture interactions for the rectangular area of the screen in which it resides. As an object in the user interface, a View is also a point of interaction for the user and the receiver of the interaction events.

The first time we compile the project, out in the main activity's namespace directory, the Android build chain will create R.java in the gen/ directory. This file contains a number of constants tied to the various resources we will place in the directory tree.

Declaring a Layout

As we have previously mentioned, layout is the architecture for the user interface in an Activity. It defines the layout structure and holds all the elements that appear to the user. We can declare our layout in two ways:

- Declare UI elements in XML. Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
- Instantiate layout elements at runtime. We can create View and ViewGroup objects (and manipulate their properties) programmatically.

The Android framework gives us the flexibility to use either or both of these methods for declaring and managing our application's UI. For example, we can declare our application's default layouts in XML, including the screen elements that will appear in them and their properties and then add code in our application that modifies the state of the screen objects, including those declared in XML, at run time.

The advantage to declaring our UI in XML is that it enables us to better separate the presentation of our application from the code that controls its behavior. UI descriptions are external to application code, which means that we can modify or adapt it without having to modify our source code and recompile.

R.java file will contain as well constants tied to all the layouts so we can reference them in our code.

As we have already mentioned, all regular XML layouts will be stored in the res/layout directory. We are going to analyze their structure basing ourselves on the layout corresponding to the main activity, called main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#FFFFFF"
    >

<ImageView
        android:id="@+id/title"
        android:paddingTop="25px"
        android:paddingBottom="22px"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:src="@drawable/titleglucoserv"
    android:background="#FFFFFF"
    />

<ImageButton
        android:id="@+id/uno"
        android:paddingBottom="13px"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:src="@drawable/viewresults"
        android:background="#FFFFFF"
        />

<ImageButton
        android:id="@+id/dos"
        android:paddingBottom="13px"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:src="@drawable/contactdoctor"
        android:background="#FFFFFF"
        />

<ImageButton
        android:id="@+id/tres"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:src="@drawable/preferences"
        android:background="#FFFFFF"
        />

<View
        android:layout_width="fill_parent"
        android:layout_height="25px"
        android:background="#FFFFFF"/>

<TextView
        android:id= "@+id/name"
        android:layout_width="fill_parent"
```

```
        android:layout_height="wrap_content"
        android:background="#FFFFFF"
        android:text="Name"
        android:gravity="center_horizontal"
        />

<TextView
        android:id= "@+id/identf"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#FFFFFF"
        android:text="Identifier"
        android:gravity="center_horizontal"
        />


</LinearLayout>
```

In this layout, we can see how we have defined a linear layout (<LinearLayout>) with some attributes such as the vertical orientation and the white background among others.

Once we have defined the kind of layout we proceed to define within it all the elements (widgets) that will compose the graphical interface of the activity such as the heading, with an <ImageView> element, the buttons with <ImageButton> and the two <TextView> elements that will show the patients' info in the main window, once we have filled the preferences.

Anything we want to use in our Java source, though, needs an android:id. The convention is to use @+id/... as the id value, where the ... represents our locally unique name for the widget in question. In the XML layout example, @+id/uno is the identifier for the first Button widget we will see on screen. All this identifiers will be as well added to the R.java file so we can work with these elements in the Java code.

Creating a Menu Resource

Instead of instantiating a Menu in our application code, we define a menu and all its items in an XML menu resource, then inflate the menu resource (load it as a programmable object) in our application code, as we will see in following sections. Using a menu resource to define our menu is a good practice because it separates the content for the menu from our application code. It's also easier to visualize the structure and content of a menu in XML.

To create a menu resource, we first create an XML file inside our project's res/menu/ directory. To understand its structure we will analyze the principalmenu.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<menu
  xmlns:android="http://schemas.android.com/apk/res/android">

  <item android:id="@+id/about"
        android:title="About"
        android:icon="@drawable/about"/>

  <item android:id="@+id/scan"
        android:title="Scan"
        android:icon="@drawable/scan"/>

  <item android:id="@+id/Exit"
        android:title="Exit"
        android:icon="@drawable/exit"/>

</menu>
```

As I have mentioned, the structure is really simple since we only have to declare that it is a menu layout (<menu> element) and then declare within it as many items as we want (<item>). For each item, he have to define the image and the label the user will see by declaring the 'icon' and 'title' attributes.

Designing Visual Resources

If we take a look to the previous XML files, we will notice that images are referenced as @drawable/... where the ellipsis is the base name of the image file we want to use. Image identifiers, as well as the rest of components stored in the res/ directory will be added to R.java.

Android supports images in the PNG, JPEG, and GIF formats. However, PNG is the overall preferred format.

Designing Tools For designing all the icons that will populate the layouts, menus and even the notification bar in our case, Android UI team provides developers with **guidelines** for the interaction and visual design of Android applications. Creating a unified look and feel throughout a user interface will add value to our app and will make the UI seem more professional to users.

Android also provides us with the **Android Icon Templates Pack**, a collection of template designs, textures, and layer styles that make it easier to create icons that conform to the guidelines given by them. The icon templates are provided in the **Adobe Photoshop** file

format (.psd), which preserves the layers and design treatments they used when creating the standard icons for the Android platform.

However, many times we will have to get hands on other icon templates provided by developers or just creating our own new icons without any template.

Density-Specific Icon Sets We have explained some facts about the Android fragmentation, so we already know that Android Platform is designed to run on a variety of devices that offer a range of screen sizes and resolutions. When designing the icons for our application, we have to keep in mind that our application may be installed on any of those devices.

Android platform makes it straightforward for developers to provide icons in such a way that they will be displayed properly on any device, regardless of the device's screen size or resolution.

Icon Type	Standard Asset Sizes (in Pixels), for Generalized Screen Densities		
	Low density screen *(ldpi)*	Medium density screen *(mdpi)*	High density screen *(hdpi)*
Launcher	36 x 36 px	48 x 48 px	72 x 72 px
Menu	36 x 36 px	48 x 48 px	72 x 72 px
Status Bar (Android 2.3 and later)	12w x 19h px (preferred, width may vary)	16w x 25h px (preferred, width may vary)	24w x 38h px (preferred, width may vary)
Status Bar (Android 2.2 and below)	19 x 19 px	25 x 25 px	38 x 38 px

Table 5.2: Icon dimensions for each of the three generalized screen densities, by icon type [8].

For creating my set of Icons I followed the recommended approach of creating a separate set of icons for each of the three generalized screen densities (low, medium and high) and store them in density-specific resource directories in my application (Table 5.2). Then, when the application runs, the Android platform will be responsible of checking the characteristics of the device screen and loading icons from the appropriate density-specific resource.

Figure 5.14: Launcher Icon

Figure 5.15: Menu Icons

Figure 5.16: Notification Icons

5.3.5 Writing the code: Activities and Services

Now that we understand the structure of our project and we know that we have to take care of the Manifest and the Layouts for the proper functioning of the app, we can proceed to write the code for all the Activities and the Service responsible of managing the connection with the glucose meter.

For the main activity we will analyze all the code very carefully in order to understand the structure of an Android activity. From then on, I will try to summarize explaining just the most significant or interesting parts of the code.

Main Activity

The main activity of the application called GlucoMain.java is a simple activity, but at the same time it is very complete as it has many characteristics that will help us to understand the structure of an activity and its components. Some of this characteristics are, among others, the use of the BluetoothAdapter, alert dialogs, the contextual menu with the corresponding menu layout and the creation of both explicit and implicit intents.

We are going to analyze it step by step to understand the structure and the code of an Android activity. Afterwards we will only analyze the most important or different parts of each activity.

```
package com.magicmedia;

import android.app.Activity;
import android.app.AlertDialog;
```

```
import android.bluetooth.BluetoothAdapter;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.Window;
import android.widget.ImageButton;
import android.widget.TextView;
import android.widget.Toast;
```

The package declaration needs to be the same as the one we used when creating the project and, as with any other Java project, we need to import any classes we reference. Most of the Android-specific classes are in the android package.

```
public class GlucoMain extends Activity implements View.OnClickListener {

        SharedPreferences preferences;
        BluetoothAdapter mBluetoothAdapter;

        ImageButton viw;
        ImageButton contct;
        ImageButton prefs;
        Menu theMenu;
        TextView name;
        TextView identf;
```

Activities are public classes, inheriting from the android.app.Activity base class. In this case, the activity holds some global variables that we have to define here, such as preferences, buttons, text or the Bluetooth Adapter, that is the entry-point for all Bluetooth interaction. Since, for simplicity, we want to trap all button clicks just within the activity itself, we also have the activity class implement OnClickListener.

```
        @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.main);

        preferences = PreferenceManager.getDefaultSharedPreferences(this);

        name = (TextView)findViewById(R.id.name);
        identf = (TextView)findViewById(R.id.identf);
        viw=(ImageButton)findViewById(R.id.uno);
        contct=(ImageButton)findViewById(R.id.dos);
```

```
        prefs=(ImageButton)findViewById(R.id.tres);
        viw.setOnClickListener(this);
                contct.setOnClickListener(this);
                prefs.setOnClickListener(this);
    }
```

The onCreate() method is invoked when the activity is started, as we have already seen. The first thing we do is chain upward to the superclass, so the stock Android activity initialization can be done. Then we request no title for the window (requestWindowFeature()) and we set it to reference the corresponding XML layout (setContentView(R.layout.main)). Moreover, we get the application-level shared preferences that work in concert with Android's overall preference framework (getDefaultSharedPreferences()).

We need to get our hands on our Button and Text instances, for which we use the findViewById() call. Since we identified our buttons and text as in the layout, now we can reference the button's identifier created automatically in the R.java file. Finally, we tell it to send all button clicks to the activity instance itself (via setOnClickListener()).

```
        public void onResume(){
        super.onResume();
        name.setText(preferences.getString("editName",""));
        identf.setText(preferences.getString("editIdentifier",""));
    }
```

The onResume() method is invoked when the activity comes to foreground. Each time that it happens, we will set the text views with the information set by the user in the Preferences and Info section.

```
    public void onClick(View view) {
        if(view==viw){
                if(arepreferencesset()){
                        Intent viewlos = new Intent(GlucoMain.this, Viewresults.class);
                        startActivity(viewlos);
                }
        }
        else if (view == contct) {
                if(arepreferencesset()){
                        Intent contact = new Intent(GlucoMain.this, ContactDoc.class);
                        startActivity(contact);
                }
        }
        else {
                Intent prefs = new Intent(GlucoMain.this, Preferences.class);
                startActivity(prefs);
        }
    }
```

In Android, a button click causes onClick() to be invoked in the OnClickListener instance configured for the button. The listener is provided to the view that triggered the click (in this case, which button). Then we just have to compare the one received to know which one have been clicked and create the corresponding intent. As the activities we launch in this method are of our own, we will create an explicit intent, naming the component we wish to launch and starting the corresponding activity.

Before making any intent, we will check if all the preferences are set by calling the method arepreferencesset() that we will explain below.

```
public boolean onCreateOptionsMenu (Menu menu) {
    theMenu=menu;
    new MenuInflater(getApplication()).inflate(R.menu.principalmenu, menu);
    return(super.onCreateOptionsMenu(menu));
}
```

We need to implement onCreateOptionsMenu() in order to create a menu for our activity. This callback receives an instance of Menu.

Actually using the menu, once it's defined in XML, is easy. We just have to create a MenuInflater and tell it to inflate our menu (principalmenu.xml).

```
public boolean onOptionsItemSelected(MenuItem item) {
    return (applyMenuChoice(item));
}
```

If the user makes a menu choice, your activity will be notified that a menu choice was selected via the onOptionsItemSelected() callback. You are given the MenuItem object corresponding to the selected menu choice. Our implementations of onOptionsItemSelected() delegate to a private applyMenuChoice() method.

```
    private boolean applyMenuChoice(MenuItem item) {
    switch (item.getItemId()){
```

In applyMenuChoice(), we see which of our menu choices were chosen by making a switch() on the menu ID (item.getItemId()) and take appropriate behavior.

```
case R.id.about:
new AlertDialog.Builder(this)
        .setTitle("GlucoServer v1.0")
        .setIcon(R.drawable.about)
        .setMessage("All rights reserved \nVictor Bernet \nECE ")
        .setNeutralButton("Close", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dlg, int sumthin) {
                }
        })
```

```
        .show();
return(true);
```

For showing the About information I preferred a classic dialog box style, so I decided to use an AlertDialog. An AlertDialog pops up, grabs the focus, and stays there until closed by the user.

The simplest way to construct an AlertDialog is to use the Builder class. Following in true builder style, Builder offers a series of methods to configure an AlertDialog, each method returning the Builder for easy chaining. At the end, we call show() on the builder to display the dialog.

```
case R.id.scan:
if(arepreferencesset()){
        mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
if (mBluetoothAdapter == null) {
        Toast toast = Toast.makeText(getApplicationContext(),
                        "Device does not support Bluetooth", 6000);
        toast.show();
}
else{
        if (!mBluetoothAdapter.isEnabled()) {
            Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivity(enableBtIntent);
            while (!mBluetoothAdapter.isEnabled());
        }
Intent blueserv = new Intent(this, BlueService.class);
startService(blueserv);
}
}
return(true);
```

This option of the menu will launch the Service responsible of managing the bluetooth connection with the glucose meter. As we do for the other intents of this activity, we will first check if the preferences are set by calling the arepreferencesset() method.

The BluetoothAdapter is required for any and all Bluetooth activity. To get the BluetoothAdapter, we call the static getDefaultAdapter() method. If getDefaultAdapter() returns null, then the device does not support Bluetooth and we will just inform the user with a toast message.

Next, we need to ensure that Bluetooth is enabled. We call isEnabled() to check whether Bluetooth is currently enable and if this method returns false, then Bluetooth is disabled. To request that Bluetooth be enabled, we call startActivity() with the ACTION_REQUEST_ENABLE action Intent. This will issue a request to enable Bluetooth through the system settings (without stopping the application). A dialog will appear requesting user permission to en-

able Bluetooth. If the user responds "Yes," the system will begin to enable Bluetooth and focus will return to your application once the process completes.

Finally if all the process succeeds we will create the explicit intent and start the Service (startService()).

```
                case R.id.Exit:
                Intent blueserv = new Intent(this, BlueService.class);
                stopService(blueserv);
                this.finish();
                return(true);
        }
        return (false);
    }
```

If the user chooses the Exit menu option we will just stop the Service running in the background and finish the current main activity.

```
    private boolean arepreferencesset(){
        if((""==preferences.getString("editName", ""))||
                (""==preferences.getString("editIdentifier", ""))||
                (""==preferences.getString("editMinValue", ""))||
                (""==preferences.getString("editMaxValue", ""))||
                (""==preferences.getString("editEmail", ""))||
                (""==preferences.getString("editPhoneNum", ""))||
                (""==preferences.getString("editServ", ""))){
                Toast toast = Toast.makeText(getApplicationContext(),
                        "Fill 'Preferences and Info' first...", 6000);
                toast.show();
                return(false);
        }
        return(true);
    }
}
```

Finally, we can analyze the arepreferencesset() method, created just to verify that all the preferences are filled in order for the app to work properly. If they are not, a toast message will appear aiming the user to fill them before using any of the features of the app.

View Results

There are some interesting things about this activity called viewresults.java. The first one is that as long as this activity is based on a ListView we have to create our activity as a subclass of ListActivity, instead of as a regular Activity:

```
public class Viewresults extends ListActivity {
```

In the onCreate() method we will set the background and the selection hint color to white:

```
ListView lv = getListView();
lv.setBackgroundColor(Color.WHITE);
lv.setCacheColorHint(Color.WHITE);
```

The other interesting fact is that this ListView will be filled with the glucose results that will be stored in a file called 'results.txt'. Every time this activity comes to foreground the list will be refreshed with the existing results:

```
...
private final static String RESULTS= "results.txt";
...

public void onResume(){
        super.onResume();
        ArrayList<String> items = new ArrayList<String>();

        //Read from file
        try{
                InputStream in=openFileInput(RESULTS);

                if(in != null) {
                        InputStreamReader tmp= new InputStreamReader(in);
                        BufferedReader reader = new BufferedReader(tmp);
                        String str;

                        while ((str = reader.readLine()) != null){
                                items.add(str);
                        }
                        tmp.close();
                        in.close();
                }
        }
        catch (java.io.FileNotFoundException e) {
                //Still no File
        }
        catch (Throwable t){
                Toast.makeText(this, "error", 2000).show();
        }
        if(items.isEmpty()){
```

```
                items.add("No results to show...");
        }

        //Populate List View with Results from file
        setListAdapter(new ArrayAdapter<String>(this,
        R.layout.viewresults, items));
}
```

First of all, we create an ArrayList called 'items' and we fill it taking each line of the results.txt file as an element of the Array. If after this the ArrayList is still empty because no results have arrived or the user have erased them, the string "No results to show..." will be added to the ArrayList.

With ListActivity we can set the list adapter via setListAdapter(), in our case, providing an ArrayList wrapping an array of strings, one for each line of the results file, as the third parameter.

The second parameter to our ArrayAdapter, R.layout.viewresults, controls the appearance of the rows and is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/results"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:padding="15dp"
        android:textSize="20sp"
        android:background="#FFFFFF">
</TextView>
```

Also this activity has an options menu in which we have to choose between inserting a result manually by creating the corresponding intent to launch the activity, or deleting all the data stored in the results file:

```
private boolean applyMenuChoice(MenuItem item) {
switch (item.getItemId()){

case R.id.manualinsert:
        Intent insertman = new Intent(Viewresults.this, ManualInsertResult.class);
        startActivity(insertman);
        return (true);

case R.id.borrar: //Delete results from file
        try{
                FileOutputStream fOut = openFileOutput(RESULTS, 0);

                OutputStreamWriter out = new OutputStreamWriter(fOut);
                out.write("");
```

```
                out.close();
                fOut.close();
            }
            catch (Throwable t){
                Toast
                        .makeText(this, "Error", 2000).show();
            }
            this.finish();
            return(true);
        }
        return (false);
    }
```

Insert Result Manually

ManualInsertResult.java is an activity that takes the information filled manually by the user and stores it in the results.txt file. After this, the information is sent to the Server for its storage using the sendresulttoserver() method. We will explain this method in 'Testing the App' section.

```
//We keep the previous existing results in buf

try{
        FileOutputStream fOut = openFileOutput(RESULTS, 0);
        OutputStreamWriter out = new OutputStreamWriter(fOut);

        out.write(mDateDisplay.getText().toString() + " - " +
        mTimeDisplay.getText().toString() + " - " +
        glucoDisplay.getText().toString() +
        s.getSelectedItem().toString() + "\n" + buf);

        out.close();
        fOut.close();
}
catch (Throwable t){
        Toast.makeText(this, "Error", 2000).show();
}
sendresulttoserver();
```

However, every result inserted will be compared to the values set by the users in the preferences menu about the highest and lowest values accepted as safe for their life:

```
if(resinserted >= Integer.parseInt(preferences.getString
("editMaxValue","n/a"))){
        Toast.makeText(this, "Critical high value", 5000).show();
        emergencias();
}
```

```
else if(resinserted <= Integer.parseInt(preferences.getString
("editMinValue","n/a"))){
        Toast.makeText(this, "Critical low value", 5000).show();
        emergencias();
}
```

As we can see, as long as we detect a critical value we inform the user with a toast message and we call emergencias() method, responsible of sending an SMS to the emergency number containing glucose level and location of the patient:

```
public void emergencias(){

LocationManager lm = (LocationManager) this.getSystemService(Context.LOCATION_SERVICE);
Criteria cri = new Criteria();
cri.setAccuracy(Criteria.ACCURACY_FINE);
String provider = lm.getBestProvider(cri, true);
Location loc = lm.getLastKnownLocation(provider);
Double d1 = loc.getLatitude();
Double d2 = loc.getLongitude();

String ltitude = Location.convert(d1,Location.FORMAT_DEGREES);
String lgitude = Location.convert(d2,Location.FORMAT_DEGREES);
ltitude = ltitude.replace(",", ".");
lgitude = lgitude.replace(",", ".");
String valoract = glucoDisplay.getText().toString();
String tipus = s.getSelectedItem().toString();
Intent sendsms = new Intent(Intent.ACTION_VIEW);
        sendsms.putExtra("address","111");
        sendsms.putExtra("sms_body", "Help me!\n" + "My glucose level is: "
        + valoract + " " + tipus +". " + "I'm in latitude/longitude:\n"
        + ltitude + "/" + lgitude);
        sendsms.setType("vnd.android-dir/mms-sms");
        startActivity(sendsms);

}
```

Android gives us access to the location services supported by the device through the classes in the android.location package. The central component of the location framework is the LocationManager system service, which provides APIs to determine location.

As with other system services, we can't instantiate a LocationManager directly. Rather, we have to request an instance from the system by calling getSystemService (Context.LOCATION_SERVICE). The method returns a handle to a new LocationManager instance.

Once we have the LocationManager instance we get the last know location choosing the best provider in each case (getBestProvider(,), getLastKnownLocation()). Finally we get the latitude and longitude of the location, process the information and send the message creating an intent and providing the emergency number and the text as extras.

Contact Doctor

ContactDoc.java is probably the simplest activity of the application. However, its development involves having some knowledge about the creation of some specific Intents demanding action from external applications.

In our case, the intents we will have to deal with are sending a SMS, sending an e-mail or placing a phone call, depending on the user's choice. We can see the code necessary to do it below:

```
        ...
        preferences = PreferenceManager.getDefaultSharedPreferences(this);
        ...

        public void onClick(View view) {
        if (view == send){              //Send e-mail
                String username = preferences.getString("editEmail", "n/a");
                Intent sendmessage = new Intent(Intent.ACTION_SENDTO, Uri.parse("mailto:"+ username));
                        sendmessage.putExtra(android.content.Intent.EXTRA_SUBJECT,
                        "GlucoServer:" + preferences.getString("editName", "n/a"));
                startActivity(sendmessage);
        }

        else if (view==sms){            //Send SMS
                String sms = preferences.getString("editPhoneNum", "n/a");
                Intent sendsms = new Intent(Intent.ACTION_VIEW);
                sendsms.putExtra("address", sms);
                sendsms.setType("vnd.android-dir/mms-sms");
                startActivity(sendsms);
        }

        else if (view==phone){          //Place Phonecall
                String num = preferences.getString("editPhoneNum", "n/a");
                try {
                Intent callIntent = new Intent(Intent.ACTION_CALL);
                callIntent.setData(Uri.parse("tel:" + num));
                startActivity(callIntent);
            } catch (ActivityNotFoundException activityException) {
                 Log.e("helloandroid dialing example", "Call failed");
            }
        }
    }
```

As we can see, we get all the information, such as the phone number and the e-mail address, from the shared preferences, previously set by the user.

Preferences

In almost all the activities we have analyzed we have been speaking about the preferences and the need to fill them for the well functioning of the application features. Well, the time has come to explain how the Preferences.java activity has been developed.

A Preference activity is mostly based on its XML layout. However, a preferences layout has its own characteristics and, as well as menu layouts, is saved in a different folder from the rest (res/xml/):

```
<?xml version="1.0" encoding="utf-8"?>

<PreferenceScreen
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:title="Preferences and Info">
        <PreferenceCategory android:title="Personal Info">
                <EditTextPreference
                android:summary="Name and Surname"
                android:hint="ej: Laura Marling"
                android:title="Name"
                android:key="editName" />
                <EditTextPreference
                android:summary="NIF"
                android:hint="ej: 11111111A"
                android:title="Identifier"
                android:key="editIdentifier" />
                <PreferenceScreen
                android:key="GlucoseLevels"
                android:title="Max and Min Glucose Levels"
                android:summary="Select Max and Min glucose levels">
                <EditTextPreference
                        android:inputType="number"
                        android:summary="Max glucose level accepted"
                        android:hint="ej: 250"
                        android:maxLength="3"
                        android:title="Max Value"
                        android:key="editMaxValue" />
                <EditTextPreference
                        android:inputType="number"
                        android:summary="Min glucose level accepted"
                        android:hint="ej: 40"
                        android:maxLength="3"
                        android:title="Min Value"
                        android:key="editMinValue" />
        </PreferenceScreen>
        </PreferenceCategory>
        <PreferenceCategory android:title="Doctor Info">
                <EditTextPreference
                android:summary="e-mail of your doctor"
                android:hint="ej: doctor@hospital.com"
                android:title="e-mail"
```

```
            android:key="editEmail" />
            <EditTextPreference
            android:summary="Phone number of your doctor"
            android:hint="ej: +34655555555"
            android:title="Phone number"
            android:inputType="number"
            android:key="editPhoneNum" />
        </PreferenceCategory>
        <PreferenceCategory android:title="Local Server Info">
            <EditTextPreference
            android:summary="Local Server IP Adress"
            android:hint="ej: 192.168.1.111"
            android:title="Server IP"
            android:key="editServ" />
        </PreferenceCategory>
</PreferenceScreen>
```

Categories are added via a PreferenceCategory element in the preference XML and are used to group together related preferences. Rather than have our preferences all as children of the root PreferenceScreen, we place three different PreferenceCategory elements in the PreferenceScreen, each one with its 'title' attribute, and then put the preferences in their appropriate categories. Visually, this adds a divider with the category title between groups of preferences.

We also introduce one more PreferenceScreen element to locate two of the preferences in a different screen, as we can see (Min Value and Max Value).

As we can see, there is a third category that we haven't mentioned when explaining how to set the preferences in the 'Arquitecture and performance of GlucoServer' section. This is because we have created this preference only in order to simulate the system to test the app. We will speak more carefully about it in 'Testing the App' section.

Once we have set up the preference XML, we use a nearly built-in activity for allowing users to set their preferences. The activity is "nearly built-in" because we merely need to subclass it extending to PreferenceActivity, and point it to our preference XML, plus hook the activity into the rest of your application inserting it into the manifest.

```
...
public class Preferences extends PreferenceActivity {
...
        public void onCreate(Bundle savedInstanceState) {
            ...
            addPreferencesFromResource(R.xml.programprefs);
        }
}
```

Service: BlueService.java

As we know, a Service is an application component representing either an application's desire to perform a longer-running operation while not interacting with the user or to supply functionality for other applications to use. In our case it is the first option, as we want to create a Service for managing the Bluetooth connection on the Background without interacting with the user. For this reason, we wont have to declare any layout or view objects.

The lifecycle of a service differs slightly from a regular activity. When we call startService() from GucoMain.java the system will retrieve the service creating it and calling its onCreate() method and then calling its onStartCommand(Intent, int, int) method. The service will at this point continue running until we call stopService() from GlucoMain.java. Multiple calls to startService() will only result in multiple corresponding calls to onStartCommand(), so no matter how many times it is started a service will be stopped once stopService() is called. Then the service's onDestroy() method is called and the service is effectively terminated.

We will now analyze each of this Service's methods basing ourselves on our BlueService.java service code.

onCreate() As in most of the activities, the first thing we do is getting the shared preferences for using them. Then, we place a status bar notification in order to notify the user that the Service is already managing the connection with the glucometer in the background.

To do this, we first have to get a reference to the NotificationManager and instantiate the Notification with the icon and the message to be shown. After this, we define the Notification's expanded message and the Intent to be fired when the user selects this expanded notification message.

Moreover, we add two features to our notification using flags, specifically FLAG_ONGOING_EVENT, that will group the notification under the "Ongoing" title in the Notifications window, and FLAG_NO_CLEAR to indicate that the notification should not be cleared by the "Clear notifications" button. Finally, we pass the Notification to the NotificationManager.

Moreover, we get a BluetoothAdapter in order to manage the Bluetooth connection, as it is our starting point for all Bluetooth actions.

```
public void onCreate() {
//code to execute when the service is first called.
```

```
        preferences = PreferenceManager.getDefaultSharedPreferences(this);

        String ns = Context.NOTIFICATION_SERVICE;
        NotificationManager mNotificationManager=(NotificationManager)getSystemService(ns);
        Notification notification = new Notification(R.drawable.notification,
        "Scanning...", System.currentTimeMillis());

        Context context = getApplicationContext();
        CharSequence contentTitle = "GlucoServer 1.0.";
        CharSequence contentText = "Scanning for results...";
        Intent notificationIntent = new Intent(this, GlucoMain.class);
        PendingIntent contentIntent = PendingIntent.getActivity(this,
        0, notificationIntent, 0);
        notification.setLatestEventInfo(context, contentTitle,
        contentText, contentIntent);

        notification.flags |= Notification.FLAG_ONGOING_EVENT;
        notification.flags |= Notification.FLAG_NO_CLEAR;
        //The notification will be active while the Service
        is running in the Background.

        mNotificationManager.notify(1, notification);

        mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
    }
```

onStartCommand(Intent intent, int flags, int startId)() Services, like other application objects, run in the main thread of their hosting process. This means that, as our service is going to be blocking, since doing networking operations, we would spawn its own thread in which to do that work.

We return START_STICKY as we want our Service to be explicitly started and stopped as needed from the main activity.

```
    public int onStartCommand(Intent intent, int flags, int startId) {
            myServer = new ServerThread();
            myServer.start();
    return START_STICKY;
}
```

Our Service must act as a server by holding an open BluetoothServerSocket. The purpose of the server socket is to listen for incoming connection requests and when one is accepted, provide a connected BluetoothSocket. We are now going to analyze the most important parts of the Thread responsible of doing this and manage the received information:

```
private class ServerThread extends Thread {
```

```
    private final BluetoothServerSocket myServSocket;
    public ServerThread() {
        BluetoothServerSocket tmp = null;
        ...
            tmp = mBluetoothAdapter.listenUsingRfcommWithServiceRecord
                    ("myServer", MY_UUID);
        ...
        myServSocket = tmp;
    }
    public void run() {
        BluetoothSocket connectSocket = null;
        InputStream inStream = null;
        byte[] buffer = new byte[10];
        while (true) {
                buf = new StringBuffer();
                ...
            connectSocket = myServSocket.accept();
            ...
                        inStream = connectSocket.getInputStream();
                    ...
                        inStream.read(buffer);
                        enviar = new String(buffer);
                        llega = enviar.replace(" ","");
                        notifynow(llega);
                    ...
               inStream.close();
                ...
               connectSocket.close();
                        ...
                        //Write the result in the text file (results.txt)
            ...
                sendresulttoserver(llega);

                enviar = enviar.split(" ")[0];
                int resinserted = Integer.parseInt(enviar);
                if(resinserted >= Integer.parseInt(preferences.getString
                        ("editMaxValue","n/a"))){
                        emergencias();
                }
                else if(resinserted <= Integer.parseInt(preferences.getString
                        ("editMinValue","n/a"))){
                        emergencias();
                }
        }
        void cancel() {
                ...
                myServSocket.close();
                ...
        }
}
```

First of all, we get a BluetoothServerSocket by calling the listenUsingRfcommWithServiceRecord(String, UUID). Then, we can start listening for connection requests by calling

accept(). This is a blocking call. It will return when either a connection has been accepted or an exception has occurred. A connection is accepted only when a remote device has sent a connection request with a UUID matching the one registered with this listening server socket. When successful, accept() will return a connected BluetoothSocket.

Once we have the BluetoothSocket, we get an InputStream that handle the incoming transmissions through the socket via getInputStream(). We read the necessary data from the stream with read(byte[]) and call notifynow() method to notify the user about the new result received.

The following actions are the same as in Manualinsertresult.java. We write the result into the text file and we send the results to the server calling sendresulttoserver(). Moreover, we compare the received result with the values set in the preferences to see if it is necessary to send a SMS to the emergency number and, in that case, we call emergencias() method.

There is only one difference in the emergencias() method. As we are working on a background Service, we will have to add a concrete flag when creating the intent to send the SMS to the emergency number. This flag will allow us to call startActivity() from outside of an Activity.

```
sendsms.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
```

Finally we need to create the method cancel() used to close the socket as soon as the onDestroy() method is called, as we will see below.

onDestroy() Before our Service is terminated we have to stop the Thread and take the Notification out from the status bar.

```
public void onDestroy() {
//code to execute when the service is shutting down
        myServer.cancel();
        String ns = Context.NOTIFICATION_SERVICE;
        NotificationManager mNotificationManager = (NotificationManager) getSystemService(ns);
        mNotificationManager.cancel(1);
}
```

5.3.6 Testing the App

Once the app is developed, we have to test if it works properly. As we know, this app is just part of a bigger and more complex communication system. So, in order to test it we will need to simulate this system. Since we don't have the bluetooth devices neither the servers at our disposal, we will simulate the network locally, using an android device as the glucometer and a laptop as the server.

The 'glucometer' will send the information to the Android Phone via Bluetooth, as the real device would do, and the results will be sent to the server through the Wifi local network.

Glucometer: 'Device 1.0' App

As I have already mentioned, there are some commercial brands distributing bluetooth glucometers, but that does not mean that they are always available to give the protocols and other information about their devices. That is why I decided to simulate the glucometer using another Android device and creating and App responsible of sending the results

Device 1.0 will be the app responsible of doing this. This app is a new project with its own manifest and resources but the code is mostly based in the Manualinsertresult.java file and as well on its corresponding layout.

However, we will include a <TextEdit> widget to the original layout in order to introduce the Bluetooth Address of the Phone we are going to send the information.

Moreover, in the JAVA file, when the user press the button 'Save', the app will send the information to the Phone instead of processing it an saving it into the results.txt file. In 'GlucoServer 1.0' Service, we connected to the socket as the server. Now we are going to see how we connect to the socket as the client to send the information every time a result is sent.

```
...
mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
...

public void onClick(View view) {
        ...
        else if(view==ok){
                        Toast.makeText(this, "Enviado", 5000).show();
                        BluetoothDevice device =
                        mBluetoothAdapter.getRemoteDevice(address.getText().toString());
                        myConnection = new ConnectThread(device);
```

```
                myConnection.start();
        }
        ...
}

private class ConnectThread extends Thread {
        private final BluetoothSocket mmSocket;
        private final BluetoothDevice mmDevice;

        public ConnectThread(BluetoothDevice device) {
        BluetoothSocket tmp = null;
        mmDevice= device;
        ...
            tmp = mmDevice.createRfcommSocketToServiceRecord(MY_UUID);
        ...
        mmSocket = tmp;
        }

        public void run() {
        OutputStream outStream = null;
        String stringToConvert = glucoDisplay.getText().toString() +
        "   " + s.getSelectedItem().toString();
        byte[] theByteArray = stringToConvert.getBytes();
        ...
                mmSocket.connect();
                ...
                outStream = mmSocket.getOutputStream();
                        ...
                        outStream.write(theByteArray);
                        ...
                        mmSocket.close();
                        ...
        }
}
```

In order to initiate a connection with the remote server, we must first obtain a BluetoothDevice object that represents the remote device. Using the BluetoothDevice, we get a BluetoothSocket by calling createRfcommSocketToServiceRecord(UUID). This initializes a BluetoothSocket that will connect to the BluetoothDevice. The UUID passed here must match the UUID used by the server device when it opened its BluetoothServerSocket.

Then, we initiate the connection by calling connect() and get the OutputStream that will send the result through the socket, via getOutputStream(). To write data to the streams we use write(byte[]). Finally, we just make sure to close the socket by calling close().

Server: ServidorLocal

In order to create a connection between our phone application and the laptop 'Server', we must implement both the server-side and client-side mechanisms, because the laptop 'server' must open a server socket and the phone app must initiate the connection.

We have seen that when we wanted to send a result to the server we used sendresulttoserver() method. We are now going to analyze it:

```
public void sendresulttoserver(String llega){
        Socket socket = null;
        DataOutputStream dataOutputStream = null;
        StringBuilder date = getdate();
                ...
                socket = new Socket(preferences.getString("editServ","n/a"), 8888);
                dataOutputStream = new DataOutputStream(socket.getOutputStream());
                dataOutputStream.writeUTF((preferences.getString("editIdentifier","n/a")) +
                " " + (preferences.getString("editName","n/a")));
                dataOutputStream.writeUTF(date + " - " + llega + "\n");
                ...
                socket.close();
                ...
                dataOutputStream.close();
                ...
    }
```

For creating a regular TCP client-side socket we use Socket(). The we get the DataOutputStream using getOutputStream().

We will now write two different strings in the socket using writeUTF(), one for the identifier and the name of the user who is sending the information, and one for the result we want to store in the server.

For the server-side of the Socket we will create a regular Java Project called ServidorLocal. On this project we will just have to create one class (MyClass.java) responsible of receiving the information and showing it on screen on our Laptop. This class will be a regular Java server-side socket as we can see below:

```
public class MyClass {

        public static void main(String[] args){
                ServerSocket serverSocket = null;
                Socket socket = null;
                DataInputStream dataInputStream = null;
                ...
                serverSocket = new ServerSocket(8888);
                System.out.println("Listening :8888");
```

```
                while(true){
                        ...
                        socket = serverSocket.accept();
                        dataInputStream = new DataInputStream(socket.getInputStream());
                        System.out.println("ip: " + socket.getInetAddress());
                        System.out.println("User: " + dataInputStream.readUTF());
                        System.out.println("Result: " + dataInputStream.readUTF());
                        ...
                        socket.close();
                        ...
                        dataInputStream.close();
                }
        }
}
```

6 Conclusions

Innovation has always existed, but it has been in recent years when companies have started to invest more in I+D. This change has occurred due to the fact that companies have noticed that they need to innovate in order to compete in an environment increasingly uncertain and changing. All competitors innovate, and no company wants to be left behind.

However, despite being aware of the need for innovation in order to bring competitive advantages, the lack of resources and time for innovation is still a challenge for many of these companies.

This change in the way companies conceive innovation has coincided as well with a period of maximum improvement concerning the technology, increasingly integrated into every market sector. That is exactly what health is about, the implementation of Internet and other related technologies in the healthcare industry to innovate and thus improve the access, efficiency, effectiveness, and quality of the services provided.

Health sector is dying for innovation and eHealth can be a great tool to give this sector a new perspective. That is why the main objective of the project was to introduce in the eHealth market a new service called 'GlucoServer' and aimed to help healthcare providers to innovate in their services.

After designing, developing and making the exploitation plan of this new service and the attached technology, the conclusions are quite positive.

First of all, we have found that GlucoServer is a viable project in terms of technology. All the required technologies such as bluetooth glucometers can be found nowadays in the market and the application responsible of managing all the data has been developed and tested to check its proper functioning.

Secondly, as we have analyzed, we have a big potential market to address our service and even more if we decide to expand it globally, which is one of the main reasons why the application has been fully developed in English.

Another positive indicator that leads us to be optimistic about this project is the fact that we are entering in an underexploited market, as it is still at a early stage. Despite there

are still some barriers we have to take into account and try to overcome there are many factors that aim us to say that big possibilities are attached to this growing market, as we have analyzed in previous sections.

However, taking a look at our environment we can notice that also some of the most innovative and big companies all over the world have noticed this opportunities and start to show their interest in this market creating really interesting products and services. This fact, coupled with that are no direct competitors with our service, leads us to this that now is the right moment to launch this service.

Finally, after analyzing our service in an innovation context, we have noticed that as well as being an innovative service it can be used by the hospitals as a tool for innovation. However, the grade of success in the market is the best indicator to decide wether or not a new product or service is a real innovation.

At a personal level, the conclusions are also quite positive and satisfying and the end of the project brings a new set of skills I may use for future projects. On the one hand, the time spent on the realization of the exploitation plan and the innovation analysis has given me the opportunity to implement the knowledge and skills acquired during my participation in the Programa Innova-TICs.

On the other hand, the development of the application 'GlucoServer 1.0' has helped me to improve my programming skills as well as learning how to develop applications for Android Platform. Building an application was quite a challenging project but as at the end of any difficult task, looking back at all the effort you put on, you feel satisfied and full of optimism to face new projects.

Bibliography

[1] Android-er. Simple communication using java.net.socket, January 2011.

[2] Canalys. Android overtakes symbian as world's best-selling smartphone platform in q4 2010, January 2010.

[3] Canalys. Google's android becomes the world's leading smart phone platform, January 2011.

[4] A. Chowdhry. Android market hits over 200,000 apps, December 2010.

[5] CISCO. Cisco telemedicine pilot program demonstrates a new way to deliver health services across the globe, (05/05/2011).

[6] Institut de l'audiovisuel et des télécommunications en Europe (France)., editor. *eHealth: is the stage set for the health IT and telemedecine market?* IDATE, 2009.

[7] Android Developers. Activity, (11/05/2011).

[8] Android Developers. Icon design guidelines, (18/05/2011).

[9] Android Developers. Platform versions, (21/03/2011).

[10] CIDEM Centre d'Innovació i Desenvolupament Empresarial. La gestió de la innovació en empreses de serveis, 2007.

[11] Blog Emprendedores. Que es innovacion, (2/04/2011).

[12] G. Eysenbach. What is e-health?, (28/10/2010).

[13] Agency for Healthcare Research and Quality (AHRQ). About two in three publicly insured adults under age 65 have one or more chronic conditions, May 2009.

[14] C. Foresman. Mobile market up, smartphones up, android and iphone way up, March 2011.

[15] J. Gadrey. *Productivity, innovation and knowledge in services*. Edward Elgar Publishing, 2002.

[16] Gartner. Gartner says worldwide smartphone sales reached its lowest growth rate with 3.7 per cent increase in fourth quarter of 2008, March 2009.

[17] P. Grundström. Mobile development for iphone and android. a comparison of 3rd party development for the iphone and android platforms, March 2010.

[18] Android Market Help. Developer registration, (23/03/2011).

[19] Android Market Help. Supported locations for distributing applications, (23/03/2011).

[20] Android Market Help. Transaction fees, (23/03/2011).

[21] HiMSS. Himss e-health sig white paper, 2003.

[22] Hopital.fr. Trouver un établissement, (4/04/2011).

[23] IDC. Idc forecasts worldwide smartphone market to grow by nearly 502011, March 2011.

[24] Andrew Kameka. Android has 150k apps, 350k daily activations, and more notes from eric schmidtÕs mwc keynote, February 2011.

[25] J. Mitchell. From telehealth to e-health: the unstoppable rise of e-health. Technical report, Federal Australian Department of Communications, Information Technology and the Arts (DOCITA), 1999.

[26] L. Morris. *Permanent Innovation*. Innovation Academy, 2011.

[27] Cellular News. Global mobile phone subscribers to reach 4.5 billion by 2012, March 2008.

[28] OECD and Eurostat. *Oslo Manual: Guidelines for Collecting and Interpreting Innovation Data*, 3rd edition, 2005.

[29] World Health Organization. Diabetes fact sheet nŒ312, (31/03/2011).

[30] Le Point. Classement des 725 hopitaux traitant la pathologie endocrinienne. les 40 meilleurs., (4/04/2011).

[31] J. A. Racoma. Symbian still leading smartphone market in apac, but android expected to overtake in 2011, March 2011.

[32] D. Rubia. Llega amazon appstore con 3800 apps para android, March 2011.

[33] V. Savov. Android market gets a web store with ota installations, in-app purchases coming soon, February 2011.

[34] SlideMe. Application promoter, (24/03/2011).

[35] R. Virkus, R. Gülle, T. Rouffineau, C. Brady, W. Kriesing, et al. *Don't Panic, Mobile Developer's Guide to the Galaxy*. Enough Software GmbH + Co. KG, 2010.

[36] Nielsen Wire. Smartphones to overtake feature phones in u.s. by 2011, March 2010.

[37] Nielsen Wire. Who is winning the u.s. smartphone battle?, March 2011.