Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Industriales

Departamento de Automática, Ingeniería Electrónica e Informática Industrial

Master on Industrial Electronics

Integration of Wireless Sensor Networks in the Internet of Things by Using Low Power WIFI

Author: Manuel Pineda Serrano

Director: Jorge Portilla Berrueco

March 2013

Master Thesis

Acknowledgements

Secretaría Nacional de Educación Superior, Ciencia, Tecnología e Innovación de la República de Ecuador" SENESCYT".

"Motorola Mobility Spain" TAMs Department.

Dr. Jorge Portilla Berrueco, great support to develop this Master Thesis.

Teachers and partners from CEI – UPM to share knowledge.

Dedicated to

Stephanie, Amelia, Donatila, Gricelda; great women and inspiration

Index

Chapter 1

INTRODUCTION AND CHALLENGES IN THIS WORK

"The quest for knowledge and the application of that knowledge for man's benefit will not be denied."

Roger M. Blough

Introduction and Challenges in this work

I. Introduction

This research and implementation belongs to a widest world, belongs to the vision of the future, belongs to the tendency of services implies to use in the better way hardware and infrastructure existing around Wireless Sensor Networks. Mobile communications, modern smartphones, miniaturization of devices have opened a new field to develop other spectrum of services that close machines to humans using modern and popular peripherals to communication. Chapter I is a short summary about main aspects of concepts involved in this work. In last part will present Objectives and limitations of this final project and organization of the report.

I.1 Wireless sensor networks (a general overview - tendencies)

In February 2003, MIT identified 10 emerging technologies that will change the world: Wireless Sensor Networks (WSN) was the first in the list.

A WSN consists of spatially distributed autonomous sensors installied around a phenomenon called nodes to monitor physical or environmental conditions such as: temperature, sound, pressure, etc. Nodes have restricted processing capacity and power, storage and communication in a wireless network [Escolar´09].

Every node can connect to one or more sensors. Network topology has three basic components: *gateway/coordinator* which is the central data collector and transmits data to other devices, *routers* should look for the best path to communicate devices in the network and *final devices or final nodes* which work in data acquisition and transmit. Wireless technology traditionally is based on IEEE 802.15.4 recommendation and its protocols such as: ZigBee, 6LoWPAN, Wireless HART, new trends driving IEEE 802.11 recommendation using Low Power WIFI chipsets [Advantech].

There is a big scope of applications to WSN for example: Security, Efficiency, Tracking, etc.

Actually, there is a lot of scientist research about this hot topic around the world, every group has take a battle flag trying to obtain the best metrics such as: coverage, response time, life time, installation, precision and frequency in the measure of variables, security, power consumption, size, possible application areas, etc. Many people compare these times with the birth of internet. Most recently research papers write about power consumption, mobile nodes, transmit methods, and services to WSN [Forster´12]. About the last comment have consult a white paper from Harbor Research: Where´s the money in Wireless Sensor Networks?

This article indicates among other things: It is important to recapitulate the tendency in WSN that have seen for decades in digital technology generally, and more recently in other kinds of product businesses "less and less physical value (products), more and more metaphysical value (services)". This idea inspired in "ephemeralization" theory from Buckminster Fuller, so evolving technology to become less and less material. In electronic technology, the exponential miniaturization of integrated circuits and data storage are obvious examples of "ephemeralization" [Allmendinger´12].

Sustainable profits, on the other hand, can be achieved in data management and the analytic activities that flow from the data themselves. The high margins in roughly 2007-2008 and beyond will not be in products at all, but in **data management and related services**. FIGURE 1 indicates a clear prediction about Smart Services growing.

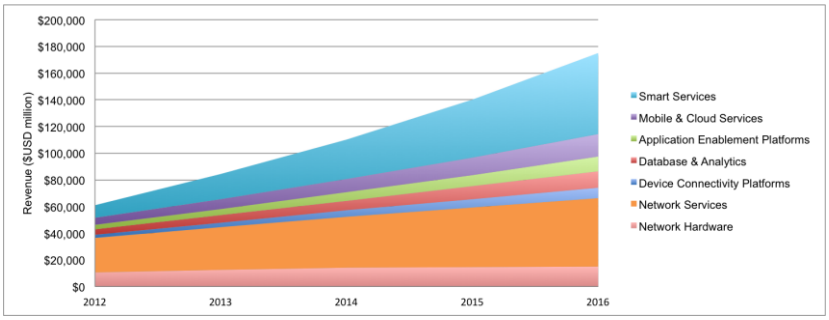


FIGURE 1: Tendencies of WSN market

In summary there is a clear orientation to grow in services and data processing to close information to human being, in this sense, concentration and data management to increase importance to the present work. An Android App gives mobility and flexibility to the data provided by WSN nodes.

I.2 Internet of Things (IoT)

Internet of things (IoT) is a modern concept; it is a kind of vision firstly devised by Nicola Tesla in an interview with Colliers magazine in 1926 [Postscapes]: "*When wireless is perfectly applied the whole earth will be converted into a huge brain, which in fact it is, all things being particles of a real and rhythmic whole.....and the instruments through which we shall be able to do this will be amazingly simple compared with our present telephone. A man will be able to carry one in his vest pocket.*"

There is a lot of modern definitions about IoT but it was chosen one that have been consider brief and complete By CASAGRAS:, it has been taken from "RFID and the Inclusive Model for the Internet of Things"

"*A global network infrastructure, linking physical and virtual objects through the exploitation of data capture and communication capabilities. This infrastructure includes existing and evolving Internet and network developments. It will offer specific object-identification, sensor and connection capability as the basis for the development of independent cooperative services and applications. These will be characterized by a high degree of autonomous data capture, event transfer, network connectivity and interoperability.*" [Casagras´07].

IoT is located inside of the Internet of Services, a widest world inside of the Future Internet Vision like show FIGURE 2 where can identify easily which are other fields of research [Haller´09].

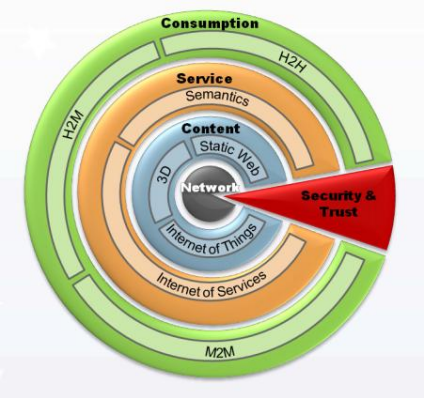


FIGURE 2: Internet of Things: Integral Part of the Future Internet Vision

The world today shows us that IoT is not the future, it is present how can explain next values presented by Harbor Research [Allmendinger´12] in a comprehensive study about the IoT:

"Today the number of connected devices on the planet is approaching the number of people - almost 7 billion. There are already more sensors on earth than people".

"In fact, IoT is a very complex vision about the complete integration of things, people, systems and real-time real-world inputs"

"Smart devices will enable new services such as status monitoring, usage tracking, consumable replenishment, automated repair, and new modes of entertainment whose value together could reach beyond $500 billion in value-added revenues by 2015".

WSN are one of the key components in most recently line of tendencies about IoT, because this technology emphasizes the pervasiveness of networks and data acquisition connections between machines and physical objects on the network, such as sensors, actuators, devices M2M (machine-to-machine) using wireless sensor networks [Tozlu´11].

IoT is composed of four layers: *Application Layer, Service Layer, Network Layer and Device Layer*. *Application Layer* is the truly application system, *Service Layer* is defined how cloud computing. *Network Layer* is every wire and wireless infrastructure, finally *Device Layer* is the key infrastructure that connects everything with internet [Exupery´09].

WSNs have two layers that perfectly fit in Application Layer and Network Layer. From the point of view of the device layer a node is composed by wireless communication and sensor interface to collect and transmit analog and digital signals to main controller.

IoT has specific aspects that should be considered:

- **Real-World Integration**, "*Better processes*" can achieve using benefits of Monitoring and Awareness. "*Better decision making*" get through execution and control.
- **Identification of things in the network,** IPV4 and IPV6 are strongly solutions.
- **Heterogeneity of Devices and Networks,** there is a lot of devices (Sensors & Actuators, Mobile phones, PLCs & embedded systems, etc) manufactured with different wireless interfaces (RFID, Bluetooth, ZigBee, WIFI, WIFI Direct, NFC, etc) and many possibilities of networks (WPAN, PAN, Mobile Networks, etc) that interact.
- **Mobility of entities,** nodes and networks. **What? Where?**
- **Managing Scale**
- **Distributed Intelligence,** a main aspect of IoT; there is a lot of information, a lot of peripherals, processors working in parallel and communicated simultaneously.

Several issues have been detected in analyses and first implementation of this technology; however: power consumption, standardization, security, identification of things, is hot topics in research agenda.

The main identified IoT application domains according to Antoine de Saint-Exupery, in his article: "Internet of Things Strategic, Research, Roadmap", are the follow:

Aerospace and aviation, Automotive, Telecommunications, Intelligent Buildings, Medical Technology, Healthcare, Independent Living, Pharmaceutical, Retail, Logistics, Supply Chain Management, Manufacturing, Product Lifecycle Management, Oil and Gas, Safety, Security and Privacy, Environment Monitoring, People and Goods Transportation, Food traceability, Agriculture and Breeding, Media, entertainment and Ticketing, Insurance, Recycling [Exupery´09].

I.3 Cookies Platform

Actually, WSNs is a technology that dispose of a very popular and mature protocol like ZigBee and hardware devices available in the market, many researches groups focus their efforts in improve wireless network management or generate algorithms to handle applications using existing technology, which implies advantages and disadvantages. By other hand, "Centre of Industrial Electronics" (CEI) of "Universidad Politécnica de Madrid", is part of the group of researches that choose the way of generating their own technology. CEI builder a hardware platform called Cookies [Mujica'12], everything with the purpose of have a better control in low level and research about power consumption, autonomy, flexibility and network scalability. Cookies are a modular platform based in four layers according to the philosophy of functionalities of WSNs nodes, these four layers are: Wireless Communications Layer, Power Layer, Processing Layer, Sensing and internal communications Layer. FIGURE 3 shows a typical node with its layers:

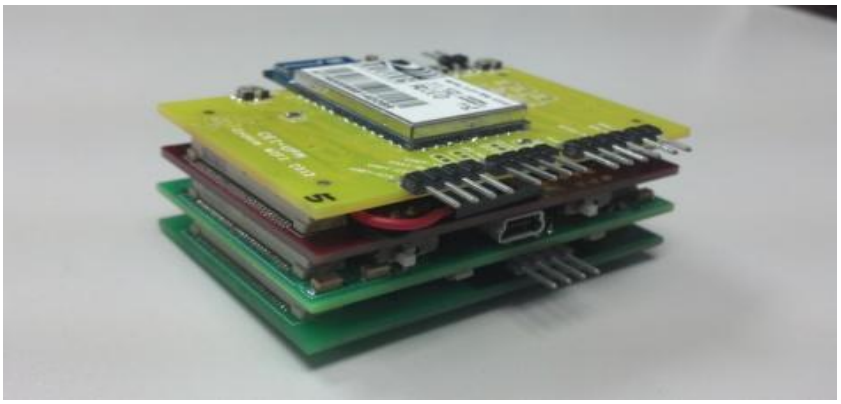


FIGURE 3: Cookie Node

Layers are connected through a common communication bus that permits interchange signals, this bus is divided in two parts: A connector in one side of layers called *"digital bus"* because contains signals from digital sensors, communication interfaces, inputs and outputs of general purpose, among others. In the other side of layers a connector with a bus called *"analog bus"* that carries analog signals from analog sensors to analog-digital converters, control signals to actuators, inputs and outputs of general purpose, UART, etc.

The main goal of interconnection bus is versatility and flexibility, because is possible to add more layers such us: Storage Layer, Debugging and Processing Layer among others if it is necessary. Interchange Layers permit for example: Interact with different environments, which require change Sensing Layer. It is possible transmit using different protocols changing Wireless Communication Layer; actually it is possible to use ZigBee and WIFI protocols. In the same way Power Layer can obtain energy from batteries, USB port, etc.

As it was referred in the last paragraph priorities and requirements of WSNs research has permitted to teachers and students associated to CEI generate every day new conclusions and knowledge about this topic. In this research line, a specific group builds a communication layer provided by a Low Power WIFI chipset called WiFly from Roving Networks Manufacturer; this present work is oriented to connect Android App with this layer.

I.4 Mobile Applications

Mobile applications or mobile apps are compact software programs that perform specific tasks for the mobile user; developed for small handheld devices, such as mobile phones, smartphones, PDAs and so on. Mobile apps can come preloaded on the handheld device as well as can be downloaded by users from app stores or the Internet. The most popular smartphone platforms that support mobile apps are Windows Mobile, Android, Symbian, IOS, Java ME and Palm [moviThinking].

There are three types of app: Native, WEB and Hybrid.

I.4.1 Native Application

In simple terms, a "Native" application is a downloadable app which is installed and run directly from the phone/device; they either arrive pre-installed on the phone.

Native applications are written specifically for the target mobile device's operating system (iOS, Android, Windows OS...etc) and are always in it is own specific coding language (Objective-C, Java...etc). Each operating system requires the application to

be written entirely from scratch in that language. This usually requires there to be a separate body of core application code for each device on which run the application.

As the languages are entirely different, each operating system also requires a developer skilled in that particular language to generate the application. Obviously these two factors can significantly increase development costs. Actually there is a huge tendency to manufacturer devices where it is possible run an open system that is the case of Android Alliance.

Other attribute of Native Applications is many of that are oriented to handle available hardware like peripherals to give a better user experience.

Advantages: fast, specifically tailored to operating system, feels "Native" to the device, as is the case of IOS application, which is leader with any native applications.

Disadvantages: more costly to build, more costly to maintain, narrow/limited audience, maintaining multiple "versions" of the application across the various mobile device operating systems is very intensive.

Examples of this kind of App are File Managers like ASTRO, Contacts Handlers, Accounts managers, calculators, games, etc.

I.4.2 Mobile Web Application

Unlike the Native Application, Mobile Web Applications can be accessed and run by anyone with a suitably modern smartphone no matter which device or operating system they happen to be using. This is how mobiThinking came to ask W3C expert Dominique Hazael-Massieux activity lead for the W3C's Mobile Web Initiative, to explain when a useful mobile Web site became a useful mobile Web app.

Mobile Web applications are usually written using "open" web coding languages (HTML, CSS, Javascript) which are the foundation stone of almost every website on the internet. The primary benefit of this is that it means that your developers/agency can build a mobile app utilizing some of the technologies in which they are already proficient.

Advantages: Runs on any device (Blackberry, iPhone, any Android Phone, any Windows Mobile...etc), single codebase means it's easy to update and maintain, utilizes "open web" technologies meaning it's less costly to develop.

Disadvantages: Not quite as slick/polished as a fully native application, requires initial web connection, although can be made to run largely offline, offline storage of data is limited/capped - consider if your app is data intensive, generally not available in App Stores.

As examples of this technology are many apps to different carriers to buy tickets, access to specific information, etc.

I.4.3 Hybrid Application

A Hybrid Application is essentially an application which is developed using "open web" technologies and then packaged up into a fully native application. Once packaged this application is essentially fully native and can be downloaded onto the device and also distributed via the various App stores like Apple Store, Play Store, etc.

As per the Mobile Web approach, Hybrid applications utilize skills in which developers should already been fluent. Whilst there is a learning curve to understanding the technologies involved in the "packaging process", the ability access all the features of a modern "smart phone" (camera, motion sensors...etc) via a single coding language potentially allows for faster and less costly development.

A major advantage of this approach is that there is one single codebase from which it is possible produce several "native" versions of the application for: iOS, Android, Blackberry, Windows Phone, WebOS, Symbian

Advantages: Single codebases produces multiple applications - extremely cost effective, takes advantage of open web technologies in which your developer team will be already skilled, cost effective. Native Application benefits, distributes via app stores, runs on device, has access to all features of device.

Disadvantages: Perhaps not 100% as slick as a fully native application, potential for issues with debugging across various platforms as not writing pure "native" code.

I.5 Objectives and Limitation of this work

The objective of this work is to generate experience and a practical example of internet of things, connected two different platforms through a standard protocol. In this sense, it is to research about possible ways to obtain a communication in two directions between a smart-phone available with Android Operative System and a wireless node provided by Low Power WIFI chipset in Communication Layer from Cookie Platform through WIFI interface (IEEE 802.11). Develop a Native Android App that will get information about variables sensing by wireless node. It implies to design a presentation layer dynamically updated with variable changes and a histogram that register last point's tendency. Finally generate a log file (.doc) in the same smartphone that collect peripheral events with the purpose of debugging and achieve a good performance of the Communication Layer.

I.6 Organization of the Report

Report contains five chapters, first chapter concludes with this brief explanation about the organization of the report; this explains main concepts around the state of the art, objectives and limitations of this research.

Chapter II highlights aspects about software tools involved in the development of Android App.

Chapter III explains modules implementation that composed Android App to achieve objectives written in chapter I using tools highlighted in Chapter II.

Experimental results, Chapter IV reflects the experience of the group against the mobile application deployment, it is a kind of scheduling quoting the milestones of the study, hardware interaction (communication with Cookie node) and every resolved issue in Presentation Layer.

Finally, Chapter V contains conclusions achieved in this work, further have been included futures lines of research that consider final users demanding.

Chapter 2

BUILDING AN ANDROID APPLICATION (DESIGN CONSIDERATIONS)

"The man who views the world at 50 the same as he did at 20 has wasted 30 years of his life."

Muhammad Ali

BUILDING AN ANDROID APPLICATION (DESIGN CONSIDERATIONS)

The previous chapter joined concepts and state of the art in which the present work was inspired. Now, in deductive way will explain different aspects that involve the tools which permitted develop this work.

This chapter summarizes main ideas and approaches getting in officials web sites, Android Developers and experience of different known gurus in Java programming and Android SDK, all of this supported in Eclipse environment.

The first part of this chapter highlights general knowledge about Android Applications and its components. After, we will write a brief glossary of Object-Oriented Programming languages (OOP). OOP´s terms applied in Java programming used in the rest of the document.

Finally, in concrete way main development considerations to design an Android App will be described.

I. Android Overview

Different aspects as: "Mobility" and "Portability" concepts, mobile communication advances (3G, UMTS, LTE, etc), increment of bandwidth looking for faster internet connections to: browsing, emailing, video broadcasting in real time, insatiable need to be ON LINE in chats, social networks; popularization of cameras, audio/video players, reduction battery size, have motivated the development of a major commercial and technological achievements: smartphones and tablets.

Android is the official operative system accepted by Open Handset Alliance, developed and supported by Google. Linux based, open source code and permissive licensing are any aspects that allows the software to be freely modified and distributed by device manufacturers, wireless carriers and developers becoming clear competition from iOS of Apple, company with which maintain an evident war for patent litigation.

Android provides a Java and C++ programming interface, but in this work will reference to Java tools. The Android SDK provides all necessary tools to develop Android Applications (Android APP). This includes a compiler, debugger (supported in Eclipse for example) and a device emulator, so many times is not essential to have an Android device to work.

Android allows background processing (threads and asynchronous tasks), provides a rich user interface library, supports 2-D and 3-D graphics using the OpenGL libraries, access to the file system and provides an embedded SQLite database.

Android applications philosophy involves three steps to follow: Design, Develop and Distribute:

Design the user interface; focus on how a user will interact with it. Android offers a lot of components oriented to handle with "*direct manipulation*", using "*multi-touch gestures*".

Develop using Android's framework that provides the APIs to build apps that take full advantage of device hardware, connected accessory devices, the Internet, software features, between others.

Distribute through *Google Play* service in which programmers can offer their Android application to Android users. This platform allows install and update APP if release versions are available.

I.1 Android Fundamentals

Android applications can be written in the Java programming language which is an OOP. Android SDK tools compile the code joined additional data; libraries, etc. called resources files allocated in Android package to generate a single archive file ".apk".

By default, every application runs in its own Linux process. Android starts the process when any of the application's components need to be executed, then shuts down the process when it's no longer needed or when the system must recover memory for other applications.

Each process has its own virtual machine (VM), so an application's code runs in isolation from other applications and has access only to the components that it requires to do its work and no more. In this way, the Android system implements the *principle of least privilege*. That is, each application, by default, has access only to the components that it requires to do its work and no more. This creates a very secure environment in which an application cannot access parts of the system for which it is not given permission.

Android application can request permission to access device data and resources such as the user's contacts, SMS messages, the mountable storage (SD card), camera, Bluetooth, WIFI and more, everything should be declared in Android Manifest file that will be explained after.

I.2 Android Applications Components

The objective of this chapter is not to write an Android tutorial but to remark any important concept. For that reason the main components such as activities, services, contents providers, and broadcast receiver will be described. The others as App Widgets, Media and Camera, Location and Sensors, etc. are defined in the Android developers web site. Process and Threads will be analyzed in chapter III because threads were a great tool to develop the present work.

I.2.1 Activity

An *Activity* is the visual representation of an Android application, corresponds to a single screen with a user interface.

Activities use views and fragments to create the user interface and to interact with the user.

An Android APP can have several activities, for example *whatsapp* can access to camera, take a picture and share it.

I.2.2 Services

Services perform background tasks without providing a user interface. They can notify the user via the notification framework in Android.

I.2.3 Content Provider

A *content provider* gives a structured interface to application data. Via a *content provider* the application can share data with other applications. Android contains a SQLite database that stores data, which would be accessed via the *content provider*.

I.2.4 Broadcast Receiver

Broadcast receiver can be registered to receive *system messages* and *intents*. A *broadcast receiver* will get notified by the Android system, if the specified situation happens. Although broadcast receivers don't display a user interface, they may *create a status bar notification* to alert the user when a broadcast event occurs. For example a Broadcast Receiver could get called once the Android system completed the boot process or if a phone call is received.

Activities, services and *broadcast receivers* of an application are activated through messages, called "*intents*". *Intents* are asynchronous messages which allow the application to request functionality from other components of the Android system in the same or different applications. An application can call a component directly (explicit Intent) or ask the Android system to evaluate registered components based on the Intent data (implicit intents). Applications register themselves to *intent* via an *intent filter*. Each filter describes a capability of the component, a set of intents that the component is willing to receive. An explicit intent is always delivered to its target, no matter what it contains; the filter is not consulted. But an implicit intent is delivered to a component only if it can pass through one of the component's filters.

II. ECLIPSE ENVIRONMENT

Eclipse is a community for individuals and organizations who wish to collaborate on commercially-friendly open source software. The Eclipse Project was originally created by IBM in November 2001 and supported by a consortium of software industrial leaders vendors as: Borland, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSof and Webgain.

The Foundation is focused on creating a *unique model of environment* for *open source development* and to promote the adoption of Eclipse technology in commercial and open source solutions [Eclipse].

Eclipse is too the name to call an *Integrated Development Environment* (IDE) open and extensible.

An IDE is a program composed by a useful tools staff to develop software.

Basic elements of an IDE are: code editor, a compiler and a debugger. Additional extensions are possible to install in Eclipse called "*plugins*" to extend its functionality, an example of this is accurately SDK of Android.

Eclipse supports programming languages as: Java C/C++, COBOL, FORTRAN, PHP o Python and others.

Since 2006, the Foundation has coordinated an annual *Simultaneous Release*. Each release includes the Eclipse Platform as well as a number of other Eclipse projects. Table 1 lists release versions of Eclipse; the present work was develop in *Eclipse Juno Service Release 1* version.

Table 1: Eclipse releases versions

Codename	Date	Platform version	Projects
N/A	21 June 2004	3.0 [1]	
N/A	28 June 2005	3.1	
Callisto	30 June 2006	3.2	Callisto projects
Europa	29 June 2007	3.3	Europa projects
Ganymede	25 June 2008	3.4	Ganymede projects
Galileo	24 June 2009	3.5	Galileo projects
Helios	23 June 2010	3.6	Helios projects
Indigo	22 June 2011	3.7	Indigo projects
Juno	**27 June 2012**	**4.2**	**Juno projects**
Kepler	26 June 2013 (planned)	4.3	Kepler projects

II.1 Object-Oriented Programming languages (OOP)

It is a way to program that use a programming paradigm call "objects" [Lewis´08].

This concept born between 1960s, when hardware and software became increasingly complex, establishing need of see Software and Hardware like something modular, that

can execute easy tasks to help something more complex, using the "Divide and conquer" paradigm.

Modern technologies oriented to increase use of GUI (graphical user interface) gives more relevant to software objects because offer a great adaptability. Nowadays, many of languages provide designers this kind of programming like an alternative to make solutions.

An Object- Oriented program may be viewed as a collection of objects. Each object is an independent machine with distinct role capable of receiving messages, processing data and send messages to others, in opposed to traditional form in which a program is like a list of tasks to perform.

A lot of concepts involve OOP languages; these concepts will overview briefly in next part [JavaTuto].

II.1.1 Software Objects

Software Objects are often used model the real world objects that can find in everyday life.

Identifying the state and behavior for real-world objects is a great way to begin thinking in terms of object-oriented programming. Two questions resume these characteristics:

What possible states can this object be in?

What possible behavior can this object perform?

Using an example: desktop radio might have additional states (on, off, current volume, current station) and behavior (turn on, turn off, increase volume, decrease volume, seek, scan, and tune).

An object stores its state in fields (variables, data) and exposes its behavior through methods (functions) that react to certain events.

Object Components are: attributes, identity, relations and methods. By other hand an object is represented by a table or entity that shows their attributes and functions.

II.1.2 Class

A class is a blueprint or prototype from which objects are created, content similar properties and behavior associated to a particular object type.

II.1.3 Inheritance

It is the feature through a class B can inherits properties and behaviors from class A like these had been defined in class B, so it is possible to have super class and class.

II.1.4 Interface

An interface is a contract between a class and the outside world trough methods.

II.1.5 Methods

Algorithm associated to an object or group of objects, their execution is an effect of "message" reception. Methods can change properties of objects or can generate a new event like a new message to other system object.

II.1.6 Event

It is a success system, in other words is the reaction that triggers an object, for example: user interaction or message sending by an object.

II.1.7 Message

A Message is a communication directed to an object that executes methods associated to the generated event.

II.1.8 Package

A Package is a Java concept to handle classes and interfaces in a logical manner, similar to organize them in folders. Placing your code into packages makes large software projects easier to manage.

II.1.9 Property or Attribute

Content a kind of data associated to an object or class of objects which are default particular properties, these values can change it by execution of any method.

II.2 Java Programming Language

Java is a programming language originally developed by James Gosling at Sun Microsystems (which has since merged into Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform.

Java was designed with the next purposes; this software should be: "*simple, object-oriented and familiar*", "*robust and secure*", "*architecture-neutral and portable*", "*high performance*" and "*interpreted, threaded, and dynamic*".

Today, Java is the most popular programming languages in use, particularly for client-server web applications (very popular in mobile software), its main objective is let application developers "write once, run anywhere" (WORA), meaning that code that runs on any Java virtual machine (JVM) regardless of computer architecture.

II.3 IMPORTANT GUIDELINES TO BUILD AN APP IN AIDED ANDROID ECLIPSE

Before starting to develop using Eclipse it is important to have basic tools like a computer with a minimum of resources described in official web page. Then the first step is install Eclipse environment [Eclipse], second step is download SDK tools which will install "*android plugins*" called ADT (Android Development Tools) and an Android device emulator, so that Android applications can be tested without a real Android phone [AndroidDev]; finally several tutorials indicates how to link this couple tools to start to work.

A general description of how to build an Android application implies many aspects to be considered. The next description gathers is the result of our experience designing this work.

II.3.1 Create an Android Project

ADT provides in the main screen of Eclipse environment an icon like this  to access to project wizard where it is possible edit application, project and package names. Other feature to fill is the minimum Android SDK version, dependent of this item our app can run in different devices. These versions and releases have been developed to adapt objects and classes to the advances in hardware and software technology. For example Android Ice Cream Sandwich have improvements in battery life administration, classes to handle NFC technologies between others, these and more details are written in Android Developers web site. Follow the next logical steps of new project wizard

(FIGURE 4) is possible to edit icon that will appears in menu of the phone and activity's title.

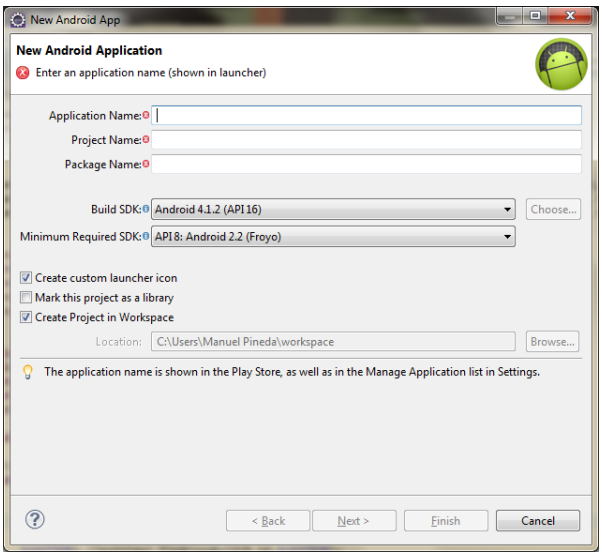


FIGURE 4: Eclipse Environment: First screen of New Android Project Wizard

II.3.2 Android Manifest file

After wizard a file group is generated, between these is possible to identify Android Manifest file.

An Android application declares its required permissions in its AndroidManifest.xml configuration file. These permissions should be written in xml language, for example an application may declare that it requires access to the Internet using next sentence:

```
<uses-permission android:name="android.permission.INTERNET" />
```

The permissions are given to applications depending of hardware and software requirements; in develop time this behavior is important to consider when "force closes" (unexpected program closures) occur.

II.3.3 Activities and Layouts

The user interface for *Activities* is defined via layouts. An example is showed in FIGURE 5. Layouts files are available in "*res*" *folder* and should be edited in XML language or using a graphical assistant to show an approximation of user layer presentation. Widgets (buttons, switches, pickers, etc), grids, views and others can used to get a professional

application. Each element in the layout has own identification (@+id/yourvalue), for example to call it from java code editor. Many characteristics and properties are possible to edit using this file, but generally its behavior will be described in java program.

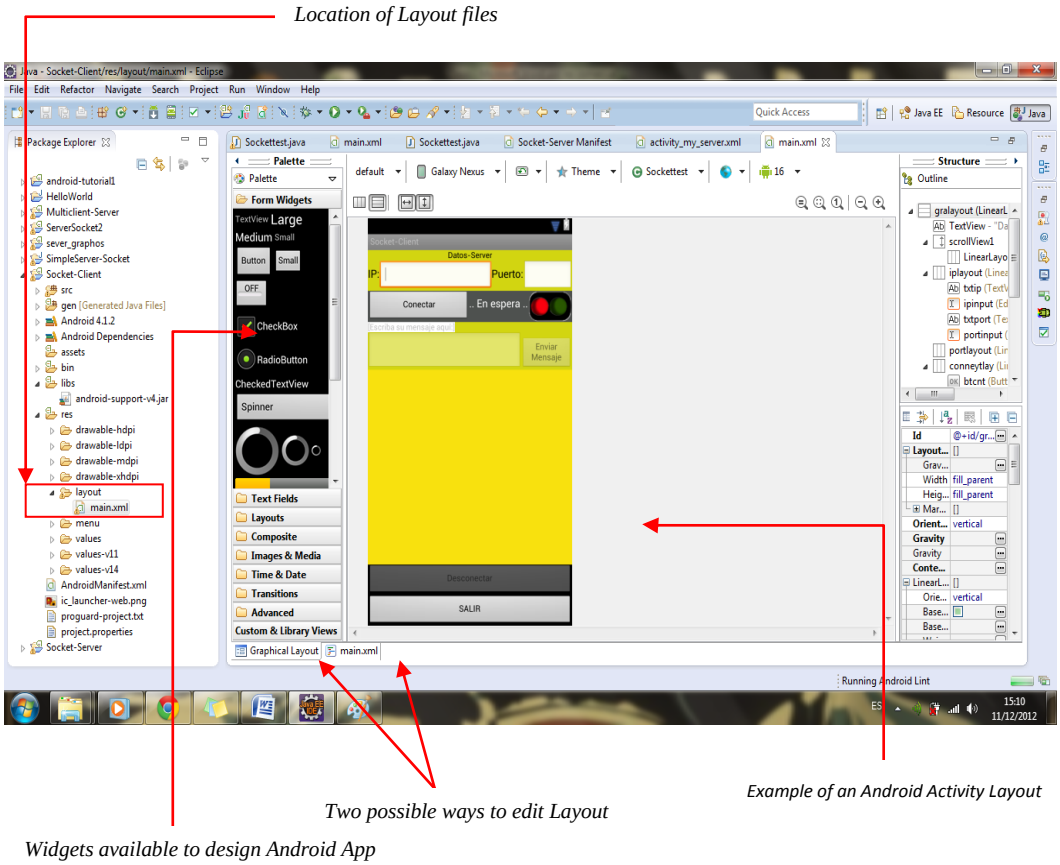


FIGURE 5: Eclipse Environment: Example of an Activity Layout

II.3.4 Java program editor

Android applications are primarily written in the Java programming language. The Java source files are converted to Java class files by the Java compiler. Java files are allocated in *"src" folder* inside of the package which has been created in project wizard.

The next chapter will be reference in more detail to the editor and will show how java file was written to reach the objective of the present work.

II.3.5 Compilation, Debugging and Running an Android Application

The Android Development Tools (ADT) performs compilation steps transparently to the user. Android SDK contains a tool called "*dx*" which converts Java class files into a .dex (Dalvik Executable) file. All class files of one application are placed in one compressed .dex file. During this conversion process redundant information in the class files are optimized in the .dex file. These dex files are therefore much smaller in size than the corresponding class files. Finally, the program *aapt* (Android Asset Packaging Tool) performs this packaging and generates *.apk* file contains all necessary data to run the Android application.

The Android applications can be debugged using virtual or real devices.

SDK tools include an emulator to run an Android system. The emulator behaves like a real Android device (in most cases) and allows test application without having a real device using host resources. Android system versions, size of SD card, screen resolution are some relevant features available to setting in Android virtual emulator; while an Android App is running LogCat, Console (Android) are collecting valid information to developer which permit to correct if any hazard exists in the code.

In this particular case, it is preferred to work with a real device by hardware interaction. Real devices are automatically recognized if drivers were installed previously in the same computer where eclipse environment is running and USB connection is plugging. The Drivers software of real devices is available in the official web page of manufactures. These drivers permit us to do information transactions with pc, use USB modem to access internet using mobile network and install and debug android software. The USB debugging should be activating in menu of our Android device, for do that access to Settings/ Development options/ USB debugging.

After click Run icon  compiling advanced bar increase its percentage, when 100 % is reached if no problems exists a screen called Android device chooser like FIGURE 6 is displayed. This screen permits us to select between real and virtual devices available to execute and debug applications.

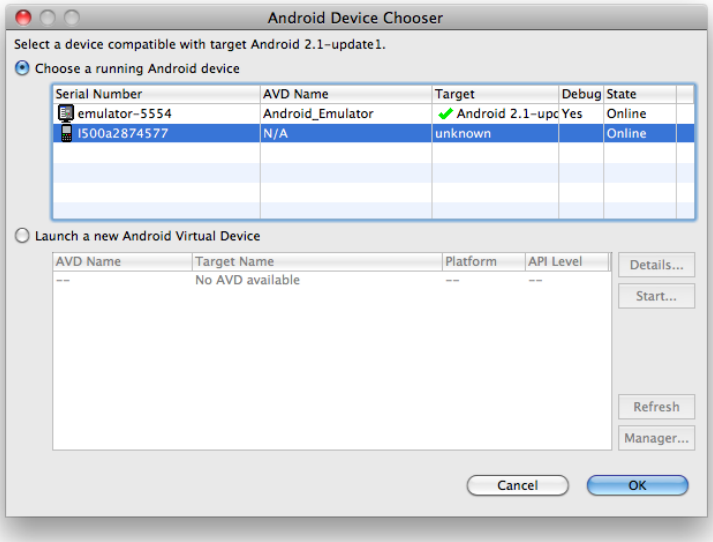
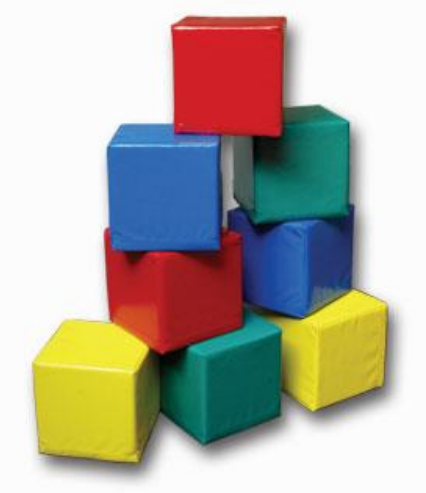


FIGURE 6: Eclipse Environment: Android device chooser

Chapter 3

SOFTWARE IMPLEMENTATION



"The idea is not to live forever; it is to create something that will".

Andy Warhol

Software Implementation

This chapter explain how were used any hardware and software resources from Android devices in this work. This chapter analyzes concepts of networking and devices administration; use pieces of software code proposed as a way to solve problems a lot of these inspired in Java Tutorials (principal from Oracle), forums, Android Developers routines and personal experience.

I. User Application Interface

I.1 Android Manifest file for this current application

As it was explained in last chapter, Android manifest is an auto generated file by eclipse and android tools assistance using own previous settings.

Screen orientation has been set in *"portrait"*, because is easier to appreciate screen content; but there is another possibility as: landscape, sensor (automatic screen adjust when device is turned) and others. Next XML line describes this explanation:

```
android:screenOrientation="portrait"
```

Hardware accelerated has been set in *"false"* because that is a condition to have a good approach of Android Plot (libraries to plot a histogram). This explanation refers to next XML line:

```
android:hardwareAccelerated="false"
```

Finally, *uses permission* gave permission to three resources:

`"android.permission.INTERNET";` permits to access networking resources.

`"android.permission.ACCESS_WIFI_STATE"` permits to access WIFI information as: MAC card address, IP address, state of WIFI connection, etc.

`"android.permission.WRITE_EXTERNAL_STORAGE"` permits to access SD card to write and read data, used to save log file.

These resources allow us debug and save a log file to follow different stages of connection.

Next piece of code refers to the complete android manifest file that was generated, and highlights aspects discussed.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="test.Sockettest"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon"
android:label="@string/app_name"
android:hardwareAccelerated="false">
        <activity android:name=".Sockettest"
android:screenOrientation="portrait"
                  android:label="Socket Server">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>

    <uses-sdk android:minSdkVersion="8" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"
/>
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
</manifest>
```

I.2 Presentation Layout

Application screen design presents basically three sub screens which are called: Console, Dynamic Numerical Report and Dynamic Histogram. The word dynamic refers to the process of update screens when a valid data arrives.

Console, is a yellow text box where can look different states of networking process as: Server up, server down, client connect, client disconnect, inbox of messages from client and outbox of message from server.

Every client is a Cookie for this specific work. Every message is label by the date, time and remote IP address.

An additionally, there is a Text Box provided with a SEND bottom and input method to transmit messages to remote devices (Cookies) which is part of console; this was thought to communicate administration commands. It is available if any remote client connection exists.

Dynamic Numerical Report, show us information data of sensors layer available in Cookies platform, we build a prototype like example to show goals of this work. Temperature, Light, Humidity and Angles of Rotation are available variables in prototype like can be seen in the screenshot (FIGURE 7). In addition, every variable has in front a check box to choose its histogram.

Dynamic Histogram is an interesting tool which presents an auto scale graph of last points registered in internal application buffers.

The next figure show how is the distribution of application screen, there is a specific bottom to close communications with a client and close the activity.

Each element explained has their properties that can be changed from the Java file or the XML layout file by their id, as can be seen in section 3.2. The XML file of Layout shows ids and the properties of the widgets used.

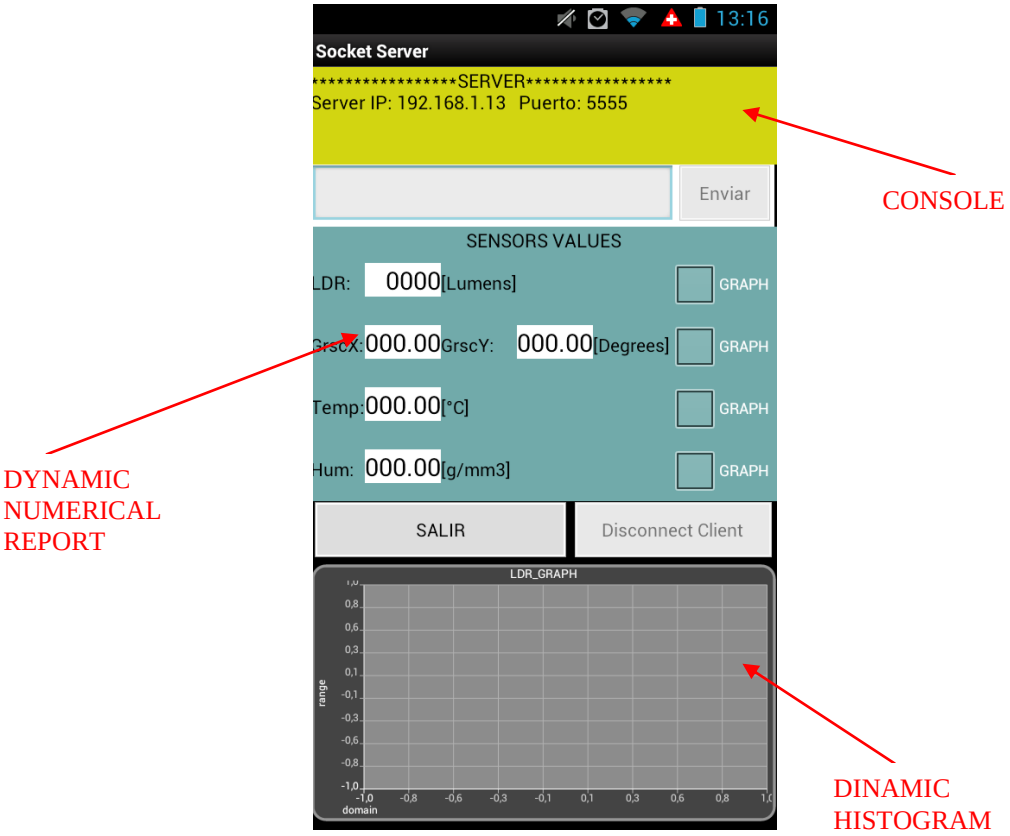


FIGURE 7: Eclipse Environment: Android device chooser

II. Communication Structure

This part of the document explains the criterion of communication layer design, is the core of this work.

II.1 Client – Server Structure (Multiclient Structure)

This structure was chosen because in practical case phone can acts like server (although can be client too) because has more resources than a Cookie node, like more memory to save logs, graphical interface or mobile internet, among others.

The first condition to establish a Client-Server communication is that client and server share the same WIFI network. There are three possibilities to do that:

a) Smart-phone and Cookie node communicates through an access point.

b) Smart-phone creates a Mobile Hotspot.

c) Cookie node creates a Mobile Hotspot, too. This chance is restricted by battery life. FIGURE 8 remarks better these cases.

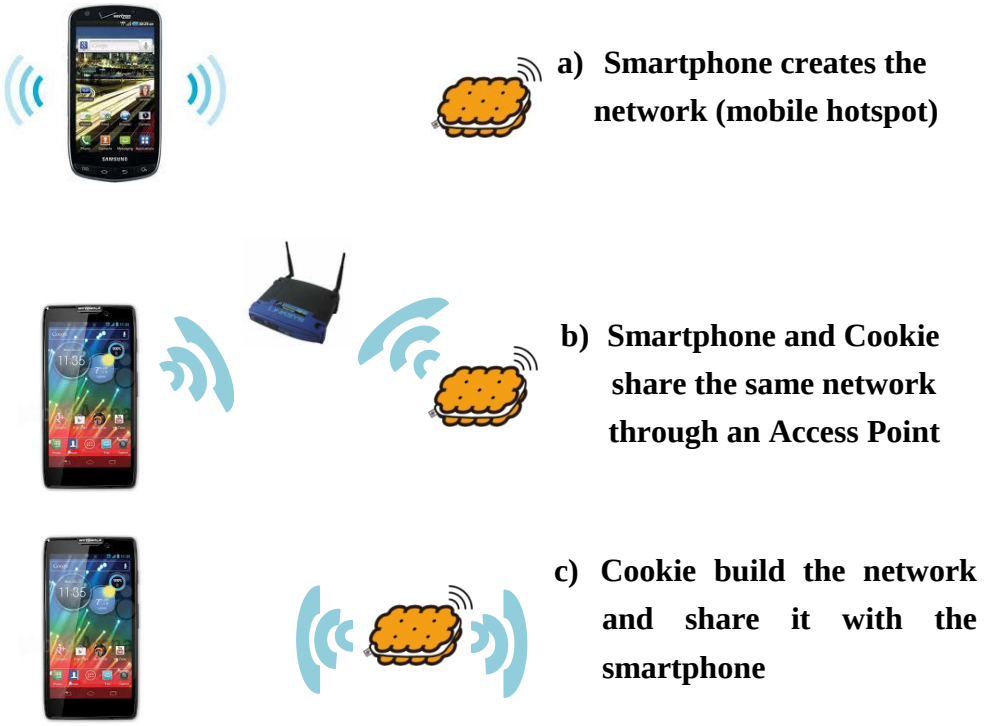


FIGURE 8: Client and Server share WIFI network (Possibilities)

The steps to follow in server configuration are described in scheme on FIGURE 9.

Server and client can have different Operative Systems, in our case the app server is running in an android device and the client will be a Cookie without Operative System that connects and send information through TCP messages and infrastructure.

Android handle sockets in the same way like Java and permit to build a Client-Server structure.

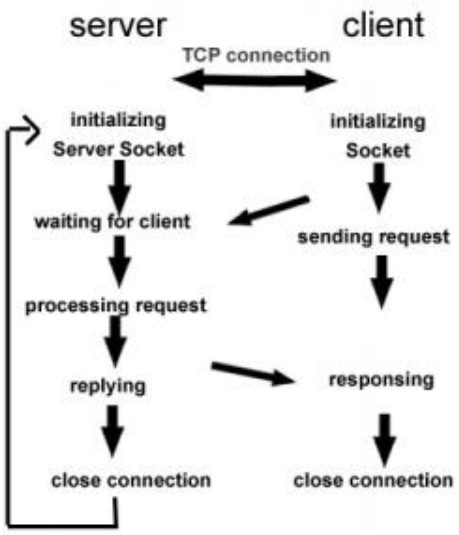


FIGURE 9: Server-Client structure

II.2 Sockets

A socket is one end-point of a two-way communication link between two programs running on the network. Socket classes are used to represent the connection between a client program and a server program.

The java.net is an API (Application Program Interface) or a package of Java that provides two classes "Socket" and "ServerSocket" which implements the client side of the connection and the server side of the connection, respectively [JavaTuto].

The IN/OUTof data in sockets is doing thorough InputStream and OutputStream associated to Sockets.

II.2.1 Create a Socket Server

The tasks implemented in the server side were as follows:

- Create Server Socket
- Accept a client
- Get InputStream y/o OuputStream of the client.

- Create InputStream y/o OutputStream appropiate to our needs (separate information from control and valid data strings).
- Read and write data from and to client.
- Close the socket.

In the server side must be created a ServerSocket with an additional parameter called: "port number"; this value can be a number from 1 to 65535, but from 1 to 1023 are reserved to system services as: SSH, SMTP, ftp, mail, www, telnet, etc. In this case it was used 5555 like port number.

The Java sentence that permits build a new socket is:

```
ServerSocket skServidor = new ServerSocket(5555);
```

It is important to highlight if there is more than one server in a network, each server may use a different port to give their services.

After create the server socket, this is waiting for client connections, to do that should call to `accept()` method. This method blocks the activity until a client connects, in Android blocks application layout, for that reason we implemented this process in background, this is called: "threads".

```
Socket skCliente = skServidor.accept();
```

A server can handle a lot of client connections for that reason will be necessary implements a lot of threads, each one for a client connection in a multi client environment.

After a client is connected, it is possible to send and receive data from the socket client using `getOutputStream()` or `getInputStream()` methods, respectively. It is important remark that these methods send and receive bits. There is a lot of another possibilities like send and receive objects but in our case we don´t use it because `getOutputStream()` or `getInputStream()` solve our problems.

From the client side, Cookie node should use IP address of android device (in the case that smartphone is server) and number of port to establish the communication and interchange messages.

II.2.2 Android Threads

As it was explained in last part of this document, `accept()` method can blocks UI (User Interface) for that reason is important to explain the tool that offers Linux core of android to run process in background, this tool are called “threads”.

When an application is launched, the system creates a thread of execution for the application, called "main." This thread is very important because it is in charge of dispatching events to the appropriate user interface widgets (bottoms, text boxes, check boxes, etc), including drawing events, sometimes “main thread” can be called the UI thread.

There is a lot of process that can block User Interface like: loading a file, accessing data from the Internet or files transfer between Bluetooth devices, etc. To provide a good user experience all potentially slow running operations in an Android application should run asynchronously, e.g. via some way of concurrency constructs of the Java language or the Android framework. This includes all potential slow operations, like network, file and database access and complex calculations. When UI thread is blocked, no events can be dispatched, including drawing events. From the user's perspective, the application appears to hang. Even worse, if the UI thread is blocked for more than a few seconds (about 5 seconds currently) the user is presented with the infamous "Application Not Responding" (ANR) dialog. From this dialog the user can choose to stop the application.

From viewpoint of Android supports the usage of the Threads class to perform asynchronous processing. In this case it was designed an UI-thread and communication-threads to handle clients. The synchronization between communication-threads with UI-thread was developed using `android.os.Handler` class.

The Handler class can update the user interface. A Handler provides methods for receiving instances of the Message or `Runnable` class. To use a handler you have to subclass it and override the `handleMessage()` to process messages. To process a `Runnable` it is possible use the `post()` method. Threads can post messages via the `sendMessage(Message msg)` method or via the `sendEmptyMessage()` method.

An extract of the implemented code is showed here, includes main thread tips, socket server configuration in Communication thread and handler to update UI. Example is applied only for one client, how was written to extends this application to multi client, each client may have its thread. Green letters refers to comments.

```
//Call the main thread; UI thread call main
@Override
        public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // all graphical content and widgets are called to show it

                //Eg; this sentence call yellow console that was configured in App
Layout
                console = (TextView) findViewById(R.id.console);
                //Eg; button1 is SEND bottom to post message to clients was
                configured too in App Layout
                button1= (Button) findViewById(R.id.button1);
                //...
                //Space to call the rest of widgets
                //Next instruction give properties of button1 when exist click event
                button1.setOnClickListener(new OnClickListener() {
                        @Override
                        public void onClick(View v) {
                        // make sure you close the socket upon exiting
                                Snd_txt_Msg(ing_data.getText().toString());


                        }

                });
                //...
                //Space to give attributes to the rest of widgets
                //Create COMUNICATION thread
                this.myCommsThread = new Thread(new CommsThread());
                //Run thread
                this.myCommsThread.start();
                }
```

```
//Implements a thread type Runnable; this is COMMUNICATION THREAD
        class CommsThread implements Runnable {
                public void run() {
                //Create a new Server Socket call ServerSocket with parameter inside
                of PUERTO variable
                try {
                skServidor = new ServerSocket(PUERTO );
                } catch (IOException e) {
                    e.printStackTrace();
                }
                while (!Thread.currentThread().isInterrupted()) {
                    try {
                        //Check if there is a client connection
                        if (skCliente == null) {
                        skCliente = skServidor.accept();//wait for a client
                        //send message; CLIENT CONNECT to Handler subroutine
                        m.what = MSG_ID0;
                        myUpdateHandler.sendMessage(m);

                        }
                        //If skCliente != null means that there is a client is
                        connected and is sending information
                        else {
                        try{
                        //Data is obtained by saved in getInputStream() and saved in
                BufferedReader  BufferedReader in =null;
                        in = new BufferedReader(new
InputStreamReader(skCliente.getInputStream()));
                                //Check if there is payload in data
                                //If is a valid data update console sending a message
to handle                                       if ((inhearing =
in.readLine()) != null) {
                                        mClientMsg =inhearing;
                                        m.what = MSG_ID1;
                                        myUpdateHandler.sendMessage(m);

                                        }

                                //If is a null data means that Client closes its
                                socket
                                //Then update console with DISCONNECT MESSAGE
                                else{
                                        //….Wait for a new client connection
                                        skCliente = null;
                                        m.what = MSG_ID2; //Send a DISSCONNECTION
MESSAGE to Handler
                                        myUpdateHandler.sendMessage(m);
                                }
                        } catch (IOException e) {
                                e.printStackTrace();
                                m.what = MSG_ID3;
                                String state = null;
                                state="Error: CAN´T RECEIVE A CORRECT MESSAGE";//In
case of ERROR
                                mClientMsg =state;
                                myUpdateHandler.sendMessage(m);
```

```
                        }

                        }

                    } catch (IOException e) {
                        e.printStackTrace();
                        m.what = MSG_ID3;
                        String state = null;
                        state="Error: SERVER CAN´T ACCEPT CLIENT";//In case of any
ERROR
                        mClientMsg =state;
                        myUpdateHandler.sendMessage(m);

                    }
                }
            }
        }

//Build a Handler call myUpdateHandler to update UI thread
Handler myUpdateHandler = new Handler() {
//Call subroutine handleMessage that deal Message from Communication thread
        public void handleMessage(Message msg) {
//This sentence call yellow console to update
        console = (TextView) findViewById(R.id.console);
        //Switch sentence permit choose type of message sending by thread
        switch (msg.what) {
        case MSG_ID0://Case msg= MSG_ID0; Client Connect

            break;
        case MSG_ID1:// Case msg= MSG_ID1;There is payload in TCP message

            break;
        case MSG_ID2:// Case msg= MSG_ID2;Client Disconnect

            break;
        case MSG_ID3:// Case msg= MSG_ID3;ERROR

            break;

        default:
            break;
        }
        super.handleMessage(msg);
    }
   };
```

III. Additional Subroutines Implemented

III.1 Obtaining WIFI information

As it was explained in past items to connect a client to the server is important to know IP address of Android device in the network and port that is hearing for clients. Android SO should use WifiManager and WifiInfo Class to obtain this information, class provides the primary API for managing all aspects of Wi-Fi connectivity. It deals with several categories of items:

- The list of configured networks. The list can be viewed and updated, and attributes of individual entries can be modified.
- The currently active Wi-Fi network, if any. Connectivity can be established or torn down, and dynamic information about the state of the network can be queried.
- Results of access point scans, containing enough information to make decisions about what access point to connect to.
- It defines the names of various Intent actions that are broadcast upon any sort of change in Wi-Fi state.

Three instructions were used to obtain WIFI IP address of device. Method `getConnectionInfo()`give us all information about *`WIFI_SERVICE`*, like: network connected, IP address, level of signal, etc. So to separate only IP address of the rest of the information we implement another method called `getIpAddress()`. IP address is provided without dots like an integer, for that reason `String.format` method is called to have an understandable IP address which will present in Console.

```
//Class handle wifi information
import android.net.wifi.WifiInfo;
import android.net.wifi.WifiManager;

//Build an object related to WIFI_SERVICE
WifiManager wifiManager =
(WifiManager)getSystemService(WIFI_SERVICE);
//Method to get Info about WIFI_SERVICE
WifiInfo wifiInfo = wifiManager.getConnectionInfo();
// Method to get IP address and save in an int variable
```

```
int ipAddress = wifiInfo.getIpAddress();
// Finally apply String.format to public in console
IP_server      =      String.format("%d.%d.%d.%d",(ipAddress      &
0xff),(ipAddress >> 8 & 0xff),(ipAddress >> 16 & 0xff),(ipAddress >>
24 & 0xff));
```

III.2 Getting Time and Date of events

To handle dates and time, Android include class called: import `java.util.Date` and `java.text.SimpleDateFormat` to get appropriate format of information, with available methods it was generated two subroutines capable of give us date and time in blue format indicated in next lines:

```
//Class date and time
      import java.util.Date and java.text.SimpleDateFormat

      //get time
      public void gettiempo(){
            Date ahora = new Date();
            SimpleDateFormat formateador = new
SimpleDateFormat("hh:mm:ss");
            TimeStamp=formateador.format(ahora);
      }
      //get date
      public void getfecha(){
            Date ahora = new Date();
            SimpleDateFormat formateador = new
SimpleDateFormat("dd/MM/yyyy");
            DateStamp=formateador.format(ahora);
      }
```

These date and time public subroutines are used to label every event registered by the server in the console of user interface layer.

IV. Generate a Log file

The term log file refers to the file to which a computer system writes a record of its activities. In this particular case each event is registered in the console and saved log file, events like: handshake and information messages, etc. The purpose of Log File is to have a backup to send it by email, upload information to the cloud or feed a data base.

It was chosen save Log File in the internal memory of the smartphone (or can use an external memory like sd card).

Java class: `java.io.File` creates a file if doesn´t exists and `java.io.FileWriter` permits us to write content in a file. We build a public subroutine called "`save_file_log`", this subroutine can save a string in `'mysdfile.txt'`, to do that we used `BufferedWriter` class and build an object `bufferWritter`. In addition this process demand a try-catch sentence, that means if is not possible save file for any hardware or software reason is possible to warn through a toast message: `Cannot save 'mysdfile.txt'`. Toast messages are an object of `android.widget.Toast`, is a kind of floating widget that remains by a while (this time is configurable).

```
//class file and filewriter
import java.io.File;
import java.io.FileWriter;
//suroutine
public void save_file_log(String to_save){
      try{
            File myFile= new File("/sdcard/mysdfile.txt");
            //if file doesn't exists, and then create it
            if(!myFile.exists()){
                  myFile.createNewFile();
            }

            //true = append file
            FileWriter fileWritter = new FileWriter(myFile,true);
          BufferedWriter bufferWritter = new
BufferedWriter(fileWritter);
          bufferWritter.write(to_save);
          bufferWritter.close();

      }catch (IOException e) {//In case of any SW or HW error!!!
            Log.e("Snd_Msg() ERROR -> ", "" + e);
            //Toast widget message
            Toast.makeText(getBaseContext(),"Cannot save
            'mysdfile.txt'", Toast.LENGTH_LONG).show();
      }
}
```

V. *Plotting a Graph in Real Time*

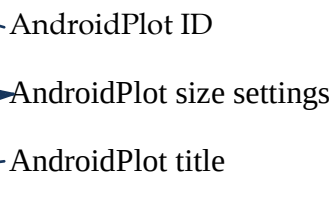
An additional graphical impression widget of data registers continuous variable from remote sensors. To do that additional library were installed in Android platform. AndroidPlot is a free tool to graph. AndroidPlot is an API for creating dynamic and static charts in Android applications. It was designed from the ground up for the Android platform, is compatible with all versions of Android from 1.6 to 4.0 and is used by many apps today.

AndroidPlot supports the following types of charts: Line charts, Scatter charts, Bar charts, Step charts.

AndroidPlot is like any widget that reserves a space for graphic. All information about this tool is available in the official web page [AndroidPlot]. In this document we remark only useful information.

To use Androidplot widget, firstly should be declared in XML file of Android Manifest, we can choose size of plot and tags of the graph, commands are showed in next piece of XML code.

```
<com.androidplot.xy.XYPlot
    android:id="@+id/mySimpleXYPlot"
    android:layout_width="wrap_content"
    android:layout_height="200dp"
    android:layout_marginLeft="0px"
    android:layout_marginRight="0px"
    android:layout_marginTop="0px"
    androidplot.title="GRAPH"
    />
```



This resource should be updated with each valid data from WIFI channel. Variable can choose with check boxes options in Dynamic Numerical Report.

Android plot has its own class that may declare in the head code of the App. `XYSeries` class permit configure the series will be plotting, `XYPlot` plots points in graph, and `LineAndPointFormatter` deal style of points and lines, attributes like colors, etc. This class should call like this:

```
import com.androidplot.xy.LineAndPointFormatter;
import com.androidplot.xy.SimpleXYSeries;
```

```
import com.androidplot.xy.XYPlot;
import com.androidplot.series.XYSeries;
```

Every variable has its own vector to register 30 values, when check an option in Dynamic Numerical Report, others are disable. Sensor´s values vectors feed the graph vector and it is redrawing to show registered values, this is done in a subroutine call `Do_graphics()`.

```
public void Do_graphics(){
      mySimpleXYPlot.clear();

      XYSeries series1 = new SimpleXYSeries(
         Arrays.asList(series1Numbers),  // Array of data
         SimpleXYSeries.ArrayFormat.Y_VALS_ONLY, // only
vertical values
         "PRUEBA"); // Name of the first serie
                  LineAndPointFormatter series1Format = new
LineAndPointFormatter(Color.RED,Color.BLACK,null);
// Point color = RED
                  //Line color = BLACK
                  //Color filling = null; so without filling
                  //Configure series and style plots graph
and redraw
                  mySimpleXYPlot.addSeries(series1,
series1Format);
                  mySimpleXYPlot.redraw();
}
```

Another subroutine called UpdatHystorial() remove first value, moves and update the vector like a FIFO register (First Input First Output) giving dynamic property.

```
public void UpdatHystorial(){
      series1Numbers[num_samples]=series1Numbers[0];
      for(int i = 0; i < num_samples; i++) {
            series1Numbers[i+1]=series1Numbers[i];
      }
}
```

Chapter 4

Experimental Results

"Experience is the universal mother of sciences"

Miguel de Cervantes

Experimental Results

Experimental Results is a chapter that compiles many steps and decisions that was taken in research and development process, everything to reach a stable communication environment. ¨Divide and Conquer" and "Trial and error" were the more used methods to reach goals. Thinking in an Objects Oriented Program is a task that requires experience and expends a lot of time understanding, reading and debugging many examples available in books and tutorials. Actually, official tutorials have a lot of information and examples although many times that is not enough because in some cases is better read forums and web post by a lot of developers around the world. Chapter IV include a description of software and hardware tools to develop environment and a flowchart that collects analyzes, decisions, tasks and process follow to finish this work. " Force closes" are the enemy number one to an Android developer, here will explain in last paragraph the experience about collected issues.

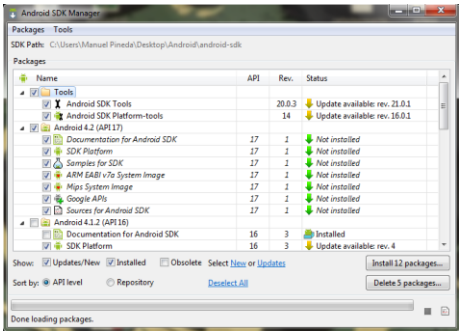
I. Tools

Android Application development tools are extremely cheap. Need a computer with an operative system where Eclipse and Android SDK tools are installed as was explained in chapters II and III. AVD (Android Virtual Device) is an interesting additional tool that permits builds a virtual android device, in this case was preferred to work with real devices because use real peripherals. The computer used was a Sony VAIO with Windows Seven Operative System. Two smart-phones from Motorola, the first one, a Motorola Razr XT910 which has a screen Super AMOLED of 4.3 feet, easy to appreciate and handle applications. It is available with 4.0.4 Android Version. The second one, a Motorola Atrix MB860, it is available with 2.3.4 Android version, this second phone emulates a Cookie node until that was replaced by a real node. It is extremely important to download drivers to handle the smart-phones, which are available at Motorola official web site [Motorola]. Finally, a cable

USB/microUSB was used as the interface between the computer and the phones to download and debug applications. FIGURE 10 collect images of tools used:

Eclipse Java Juno Service Release 1

Computer + Windows 7

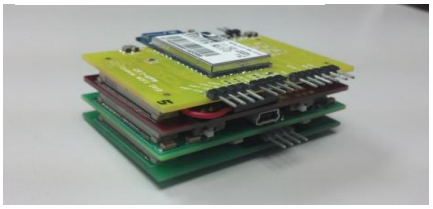


Cable USB (computer) microUSB (smart-phone)

Android SDK Tools Rev. 20.3

Motorola Razr XT910 + Android 4.0.4 Ice cream sandwich

Motorola Atrix MB860 + Android 2.3.4 Gingerbread



Wireless Node Cookie with Wi-Fi layer

FIGURE 10: Development Tools

II. Agenda

Main steps of the research, process implemented, analyzes, etc. are joined in flowchart (FIGURES 11 and 12). Each task will described with more details in next paragraphs.

II.1 Feasibility Analysis

IoT sets communication between independent platforms using a standard peripheral (now IoT vision is not clearly about which is interconnection protocol). In this sense a first question needed to be answered:

"How to create a chat between devices using WIFI and Android Operative System?"

The first step taken was looking for, download and analyze any commercial chats applications available at Google Play (application market of Android), everything to evaluate "*feasibility*". These applications require an additional infrastructure. These options were discussed in 3.2.1 paragraph in chapter III too. Next possibilities were reviewed:

- Communication in Ad-hoc network using P2P (Pear to pear) protocol.
- Communication LAN WIFI structure way, this is using an Access Point with DNS (Domain Name Server). Server – Client structure.

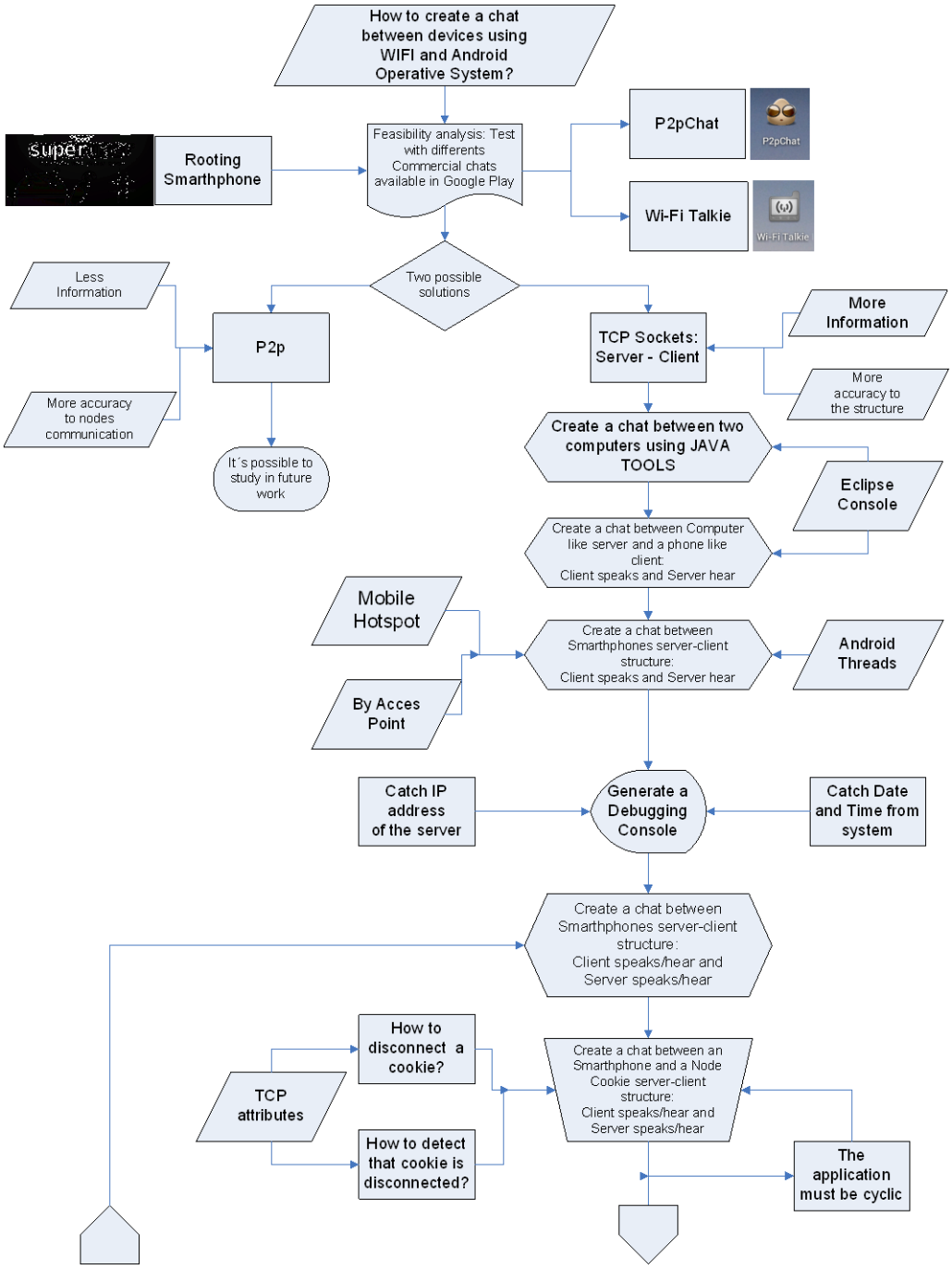


FIGURE 11: Flowchart part 1

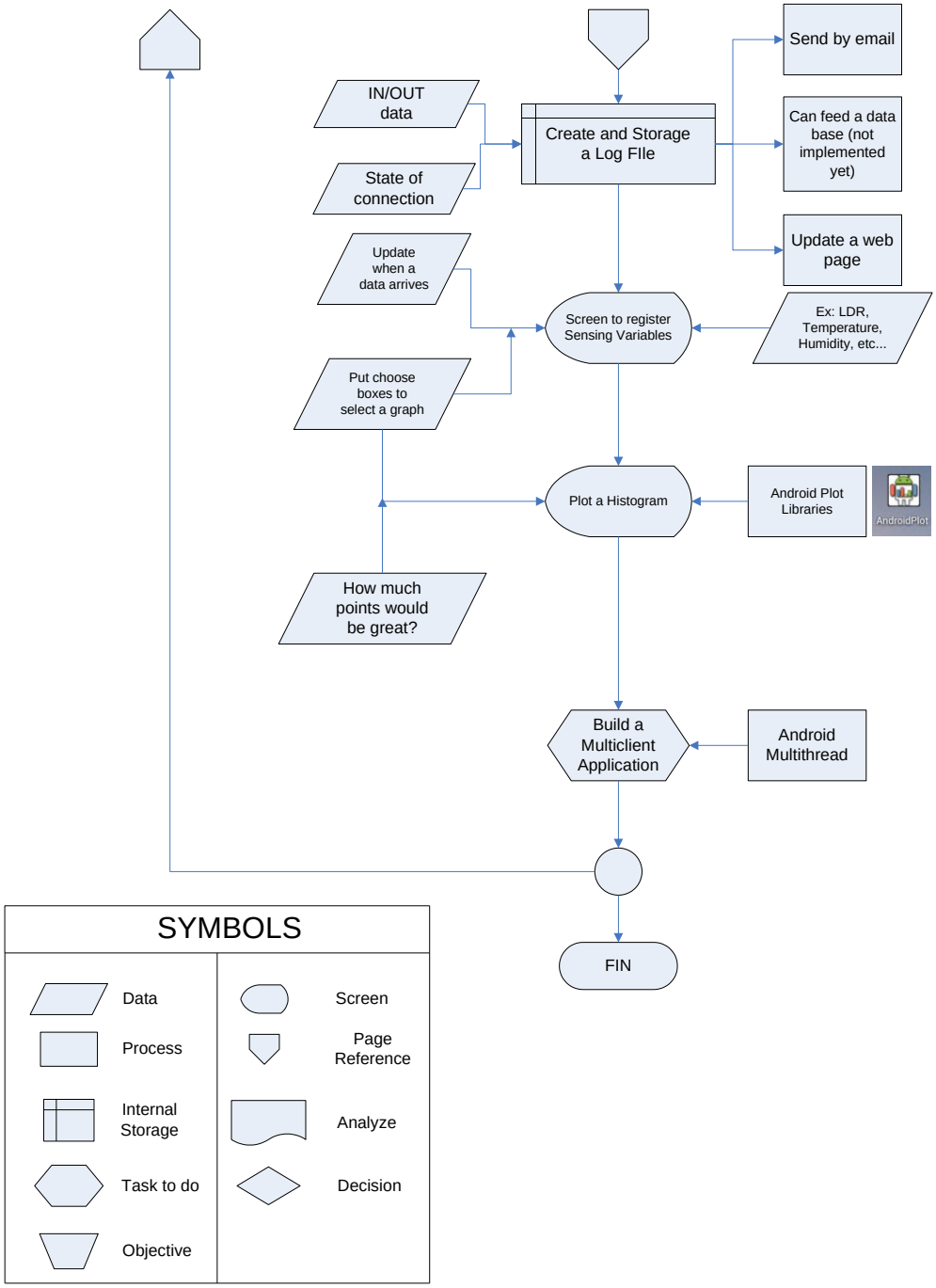


FIGURE 12: Flowchart part 2

- Communication LAN WIFI structure way using Hotspot option available in Android phone, which phone acts like: router and DNS.

Any of these applications demanded total control above the smartphone, that is to say *routed smartphones* with super user permissions because in its execution can handle hotspot, networks manager, permissions, etc; process attributed only to Android System Manager. Routing a phone really can be a risked process because manufacturer programming windows could be opened and generate errors in correct performance of the device, each phone can be routed in different ways and super user permissions is a process which change Android Manifest file sentences, it is understood more later.

Superuser Android Application [Googleplay´01] was the program which gives full control over smartphone. Application (FIGURE 13) tab is described in next paragraph.

FIGURE 13: Logo of Superuser Android Application

"Grant and manage Superuser rights for your phone"

This app requires that you already have root, or a custom recovery image to work. This app cannot be uninstalled if it was installed with your ROM or root. If there is no uninstall option, it's because you placed it in /system/app, not me. It is impossible for an App to be installed in /system/app by the Play store. In fact, the entire /system partition is read only at runtime. The only way for an App to be placed in /system/app is via a rooting process (which usually installs Superuser), or by remounting the system partition as read/write and manually putting it there.

Permissions requested are for:

Internet - updating the binary

External storage - backup/restore for elite users

NFC - creating an allow tag for elite users
Boot completed - fix database on boot

Support request emails that simply say "Doesn't work" or something to that effect are automatically deleted by our stupidity filter and will not be answered.

A couple screenshots to give an idea about use of this App are presented in next pictures:

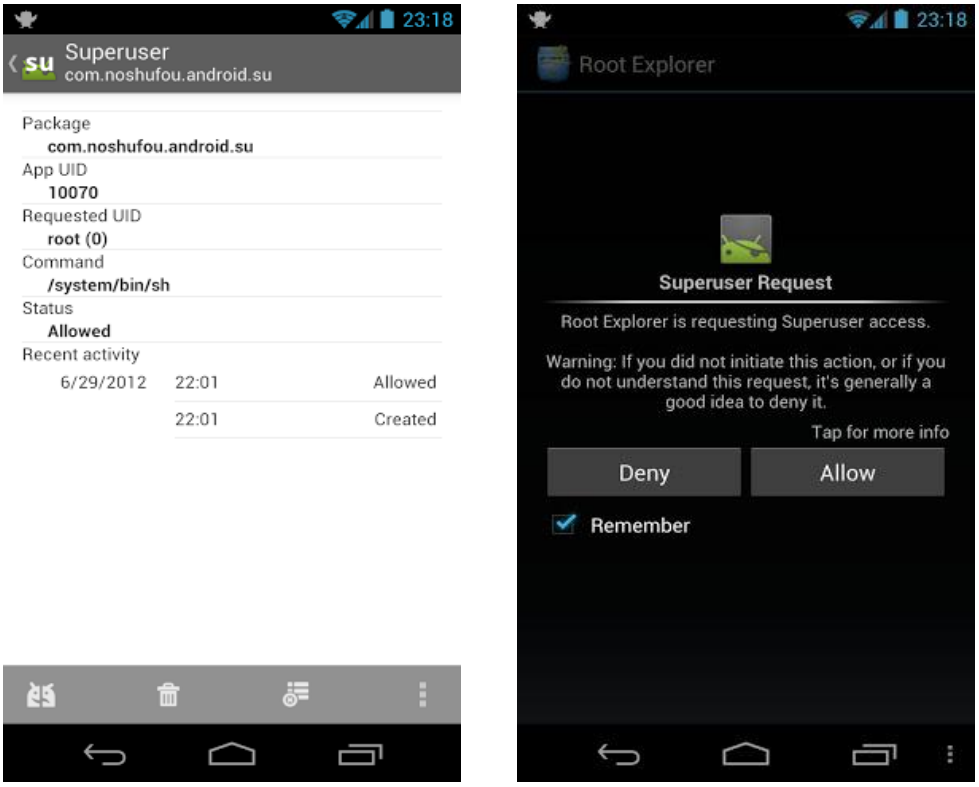
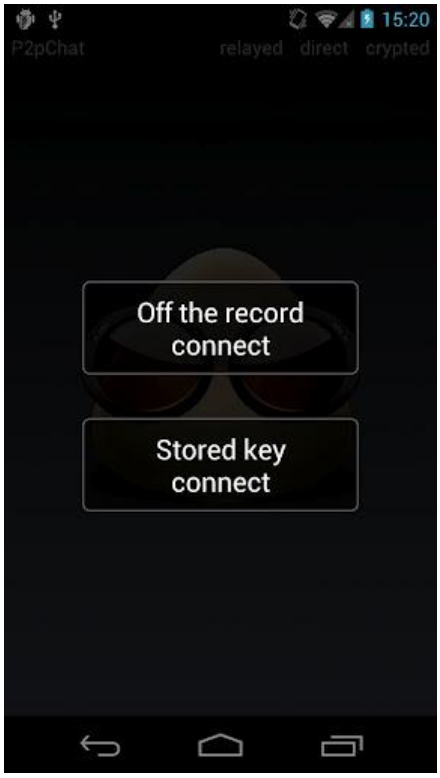


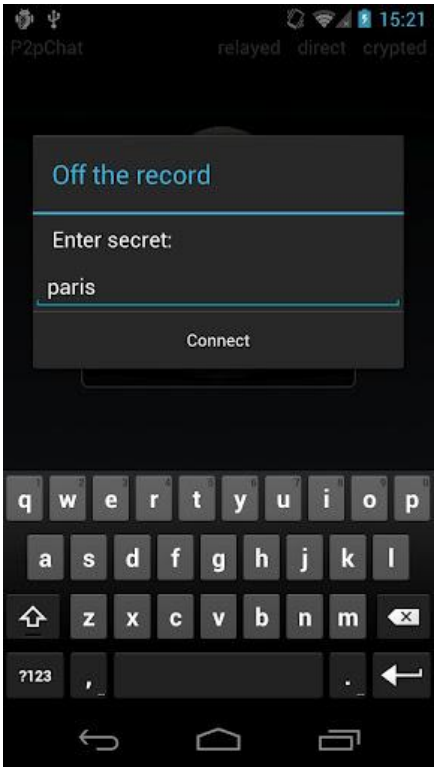
FIGURE 14: Superuser Android App screenshots

The first android chat application evaluated was *P2P Chat,* its logo is showed in FIGURE 15; features are presented in next paragraph.

FIGURE 15: Logo P2pChat

This is a secure peer-to-peer chat application. Create encrypted connections between Android devices behind remote firewalls. No need extra configuration or accounts, peer is established using a secret word; main screens are showed in FIGURE 16.





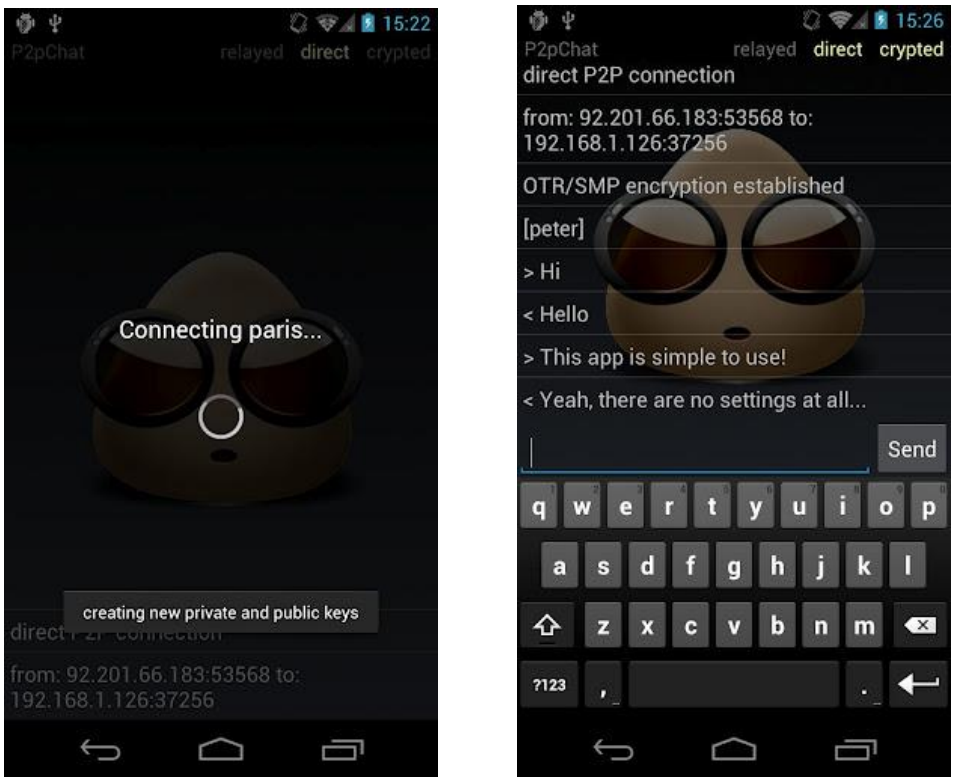


FIGURE 16: P2pchat Android App screenshots

P2pChat Android App is not available in Google Play, now. At the time of research started P2pChat gave the criterion about this kind of applications. Any Android devices were tested using this App; in some cases observed that is not possible to establish an ADHOC network. Modern Android devices have other wireless peripherals like: NFC, WIFI direct, Bluetooth, but Pear to pear protocol is not available, for that reason and few information related with this topic concluded that is necessary to look for another more popular way to build the App related to the present work.

The second analyzed application was Wi-Fi Talkie, an application that uses TCP sockets for information transfer. This application was evaluated used two phones under wireless environment and hotspot established. Wi-Fi Talkie was a good

example about what was locking for and had a heavy weight in final decision. Wi-Fi Talkie Tab is described in next paragraph; logo corresponds to FIGURE 17.

FIGURE 17: Logo P2pChat

Main features:

- Application doesn´t need Internet connection or Mobile network.
- Voice communication between smartphones.
- File transfer with speed of up 24 Mbps.
- Chat and personal messaging.
- Turn on Wi-Fi Hotspot to build a local Wi-Fi with smartphone.
- Advanced settings for VPN networks, etc.

Ways to use the application:

- Use like Walkie-Talkie.
- File transfers fastest than Bluetooth connection.
- Calls in the same wireless environment like: house, office, university, school.
- Calls in noisy places.
- Baby monitor

Limitations in Lite Version (free version that was used to test):

- Calls until 1 minute.

- Transfer files between Lites versions of Wi-Fi Talkie. Files size limitation: 2 MB.

Screenshots of the application are showed in FIGURE 18.

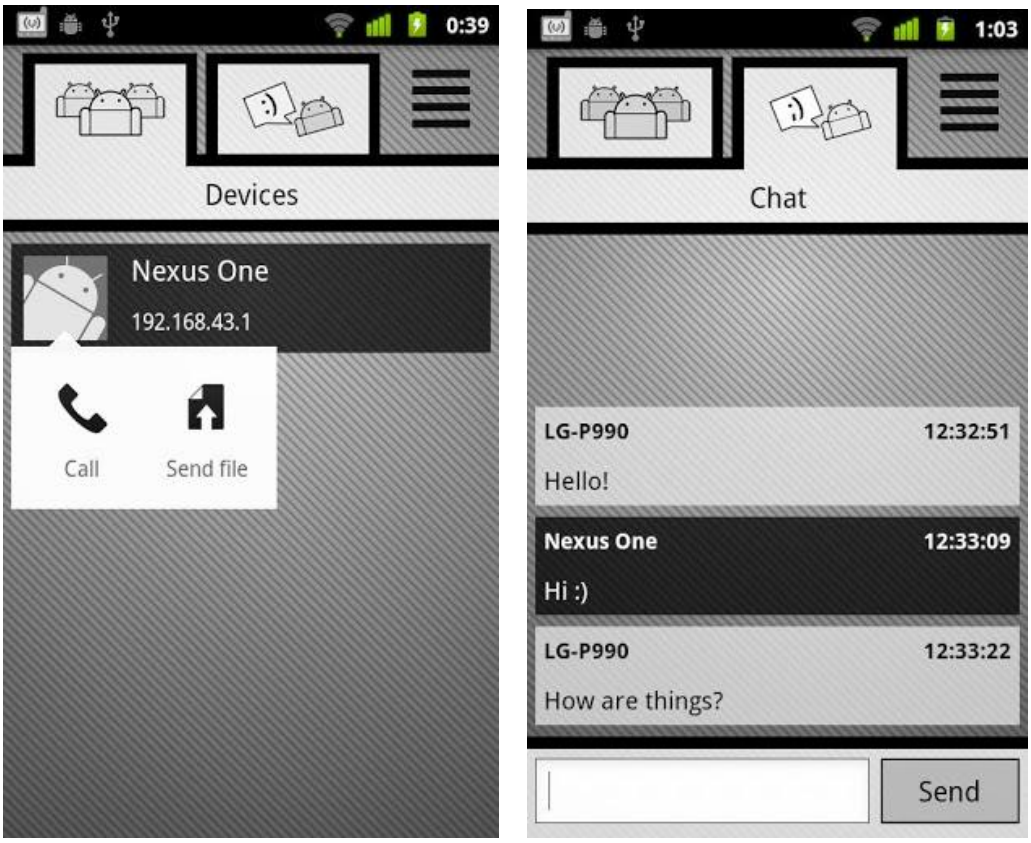


FIGURE 18: Wi-Fi Talkie Android App screenshots

In 2.3 paragraphs, the correct way to install Eclipse and Android platform was explained. A good internet connection, expend a lot of time reading tutorials and patience (download Android tools and updates) warranty a successful installation of the tools in the computer.

II.2 Build a chat between computers

"Hello world" was the first compiled program which permits collect first experiences with new environment. However compiling this program was possible to identify Java console (FIGURE 19); it´s a space where can debug programs within design an application layer, easy to use because console is an extra object that is possible to handle with methods.

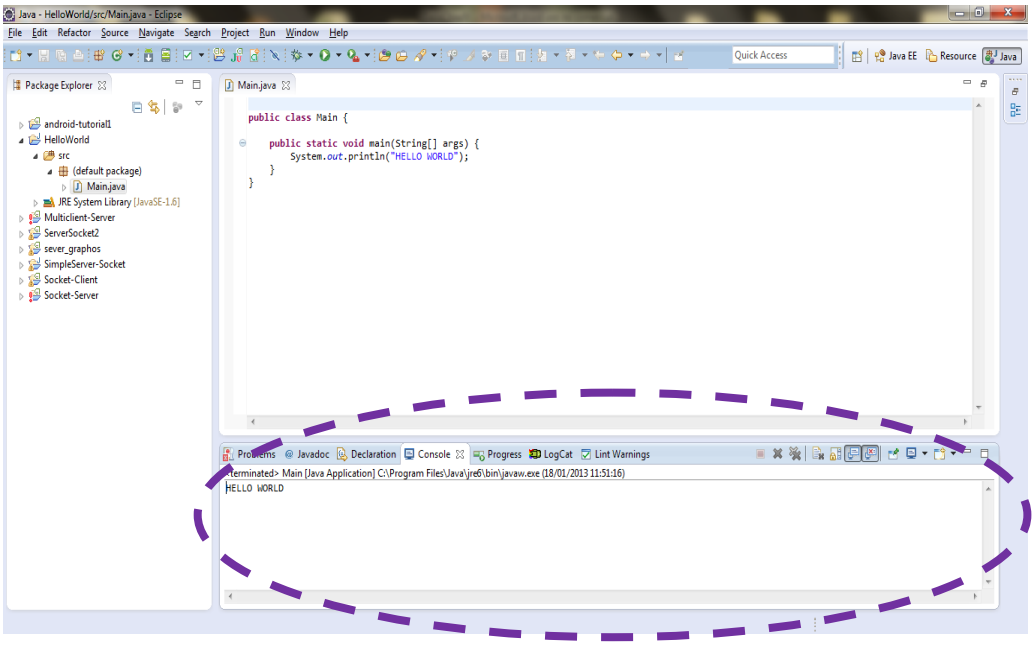


FIGURE 19: Eclipse Environment, Java Console: "Hello World" example

Inside of Eclipse world, it was decided to build and debug a first application (of course maiden a lot of previous tests) that will interconnect two computers through a WIFI wireless coverage using java console and client/server structure. Same platform, a lot of official and unofficial information was the ingredients about this first experience. Debug and test this software requires Firewall turn off. In this sense, have worked with sockets and ports which were described in 3.II.2 paragraph, open a socket and a port are tasks attributed to the server and connect to this port and

socket requires client job, everything under the same WIFI network. After this experience concluded that build a client will be an easy task, but server will require more effort, bad news because the goal of this work requires a server, no a client. Server flowchart is showed in FIGURE 21; this will be foundations of the rest of the work.

II.3 Build a client in Android smartphone

"Divide and conquer" paradigm inspired the next step of this work: Build a client in Android phone and use the same server developed in the computer, in first step to build a chat, client can speaks and server hear. However when begins with hardware configuration involves a lot of questions:

How to begin? Where is the way to download the application? Which is the way to debug application? Etc.

An interested tutorial clarified this doubts [Vogel]. "Hello world" application open paths to work with android. A screenshot of this experience are showed in FIGURE 20.

FIGURE 20: Hello world Android App

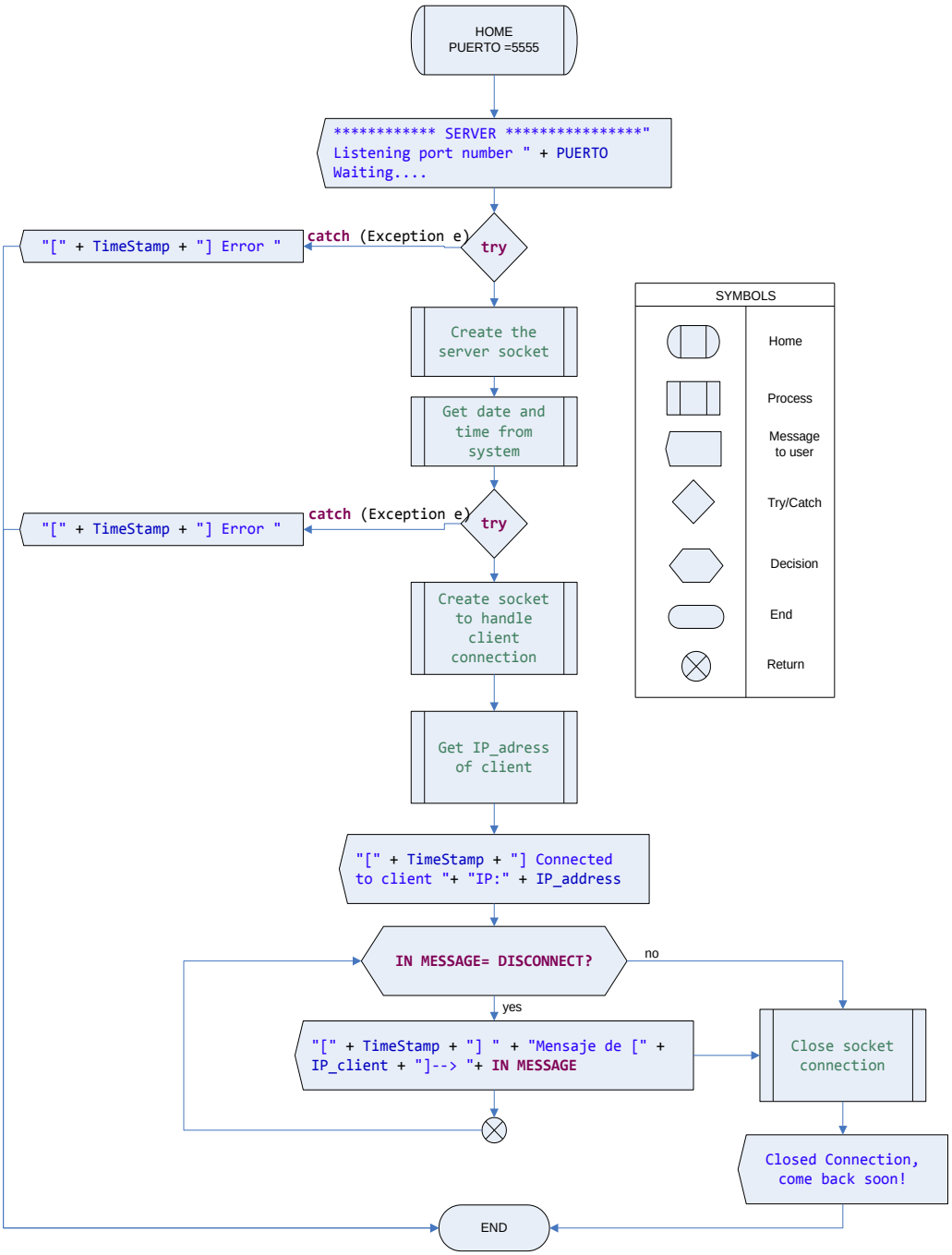


FIGURE 21: Server flowchart

Client Android Application has the same idea of the program implemented by computer. Layout, Manifest file, and the .java extension file are the key of Android develop. The requirements of client application were:

- Text boxes to configure server information. IP address and Port of the Socket are required information to establish a client-server structure.
- A button called CONNECT try to connect remote server. Attributes of button are: it is available only when settings were configured, after touch it; data are packed and sending the request wait for an ACK from server. It is transparent to the user, these transactions are made using TCP protocol messages how was explained in 3.2.1 paragraph.
- Once open socket a text box and a button called SEND are exposed to input a text message that will be send though virtual.
- DISCONNECT button to close port and establish a new connection.
- LOG OUT to close application.

Flow chart of the client program is added FIGURE 22; in the same way is presented Graphical Layout files on FIGURE 23.

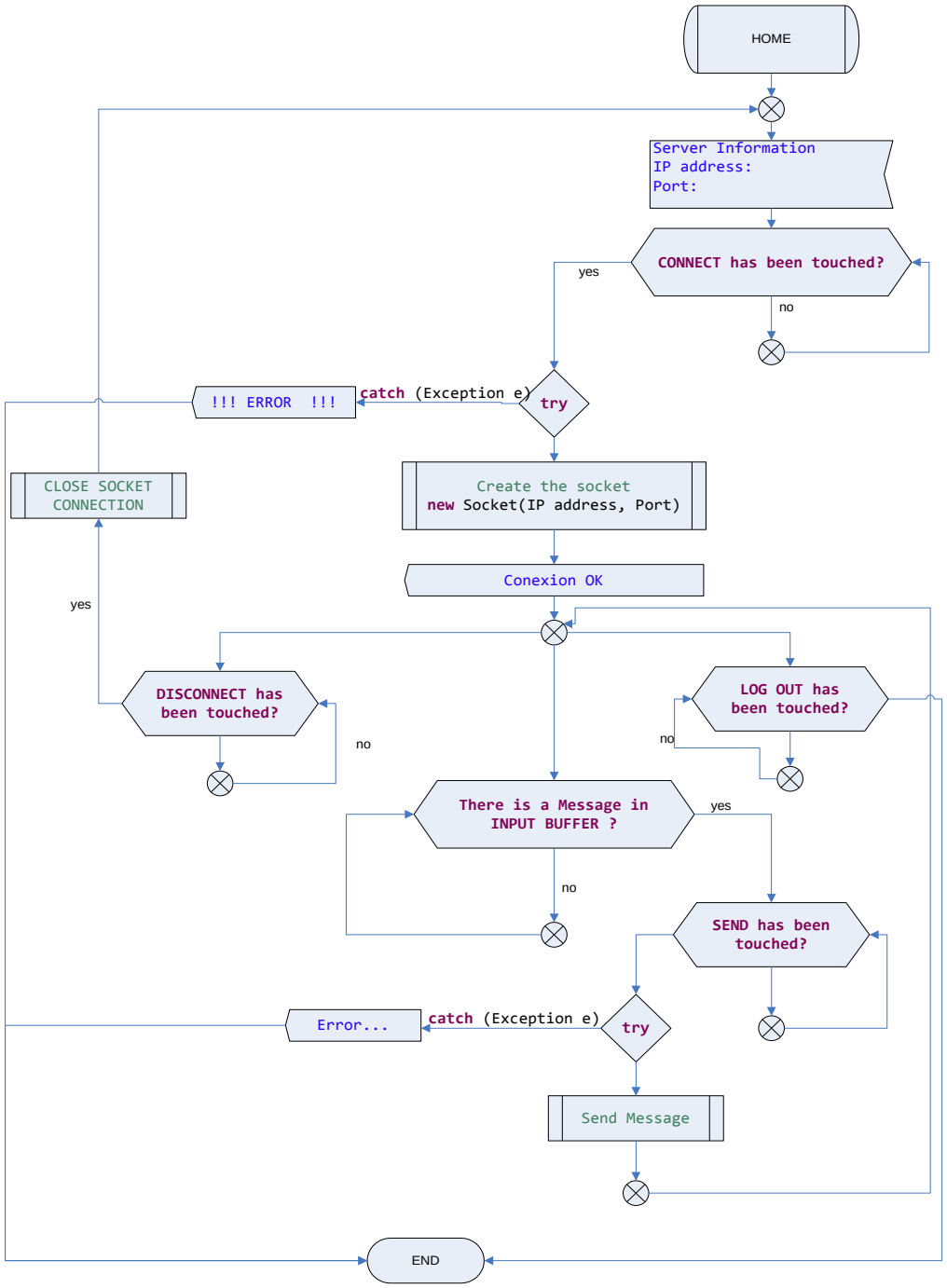


FIGURE 22: Client flowchart

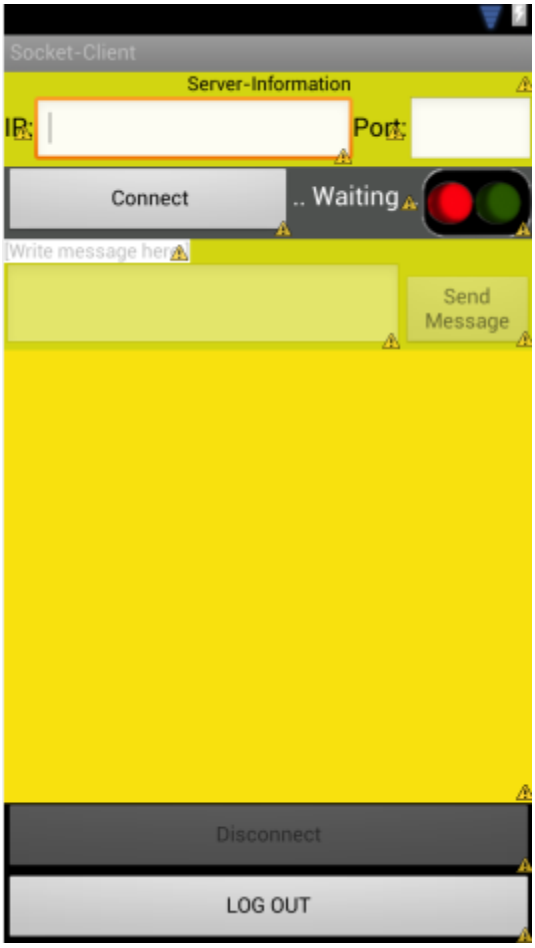


FIGURE 23: Layout Files: Graphical Interface

II.3.1 Evaluate client Android Application with server program in a computer

Test of the server and client connection requires next evaluation steps:

a) Computer server and Android client should be in the same wireless network.

b) Get IP server address (computer), for do that we apply "ipconfig" command in CMD of windows. FIGURE 24 shows this process.

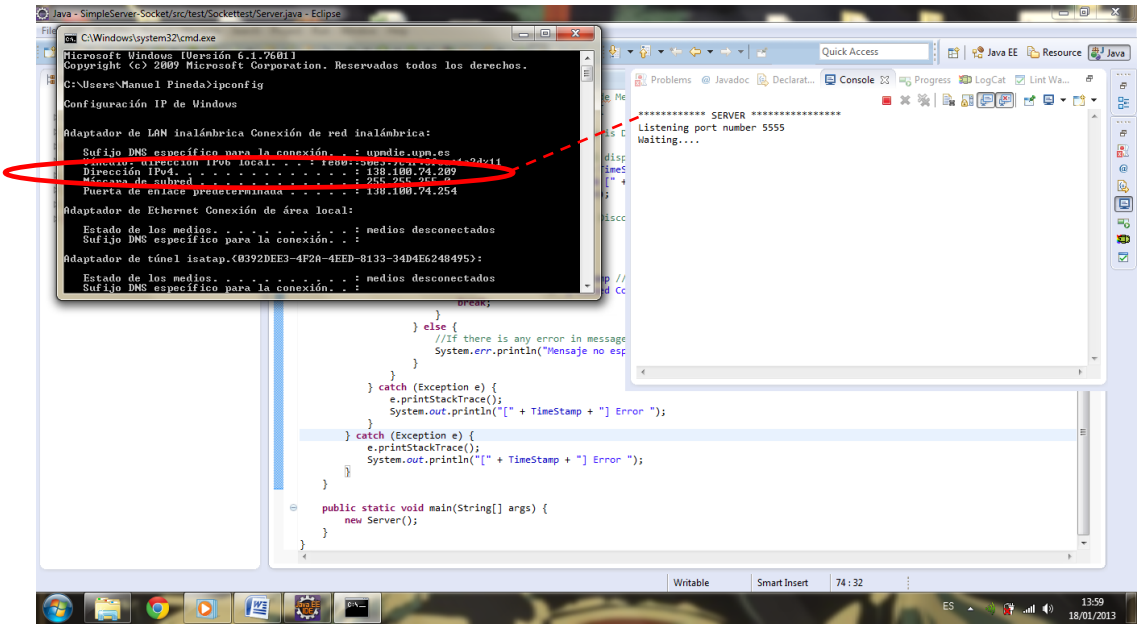


FIGURE 24: Get IP server address

c) Set information server in the smartphone (Screenshots in FIGURE 25):

d) Finally, Server can read messages from client. Example is showed in FIGURE 26.

In conclusion, a stable communication has been reached between Android smartphone and a java server running in a computer.

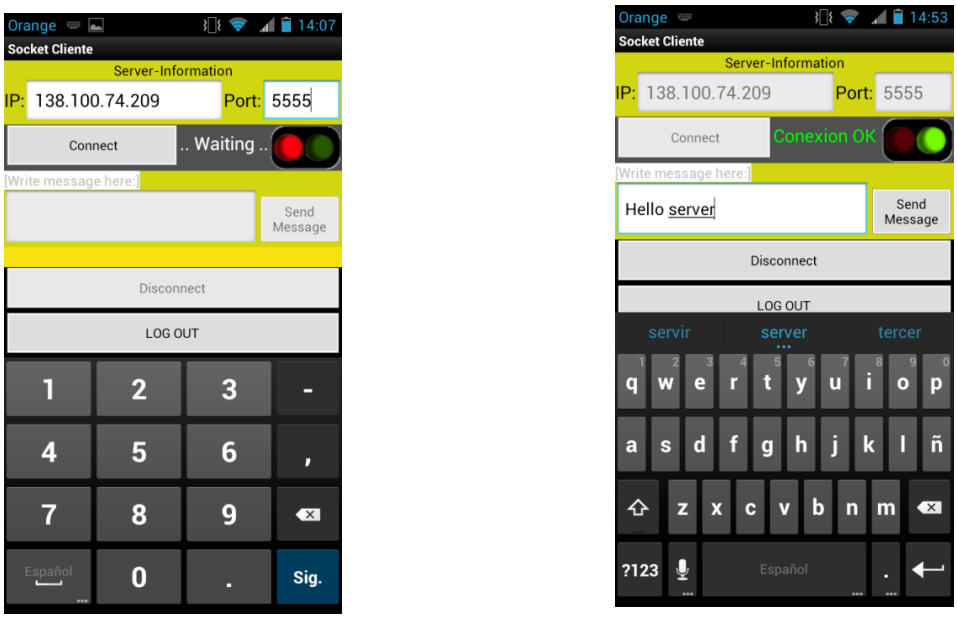


FIGURE 25: Get IP server address

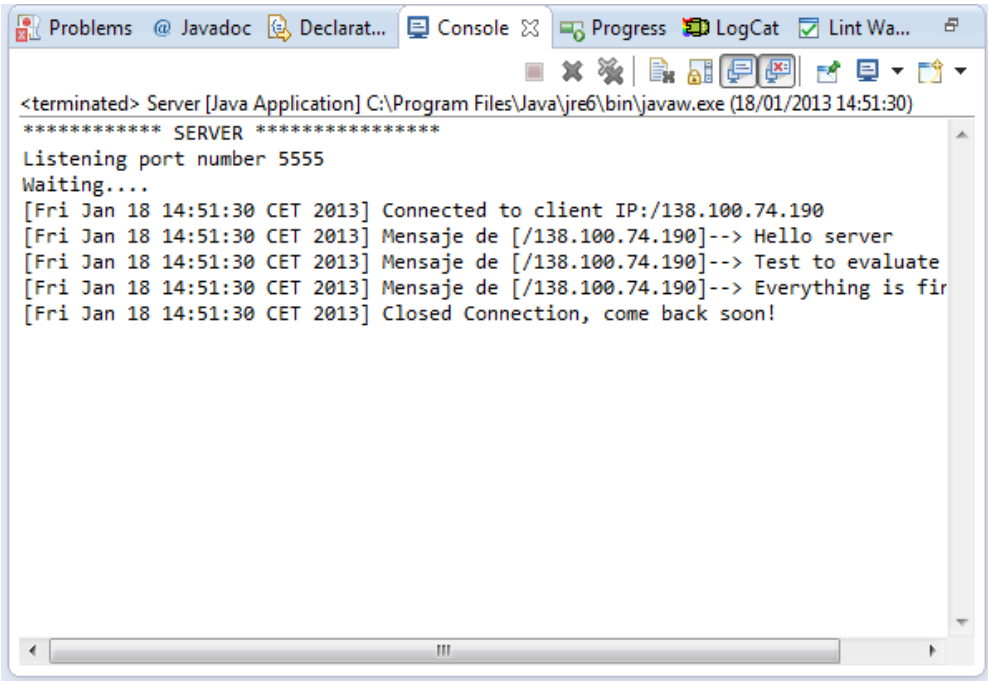


FIGURE 26: Java Console: Results of client connection

II.4 Build a Server Android Application

In the first instance was necessary set a text box that acts like java console (now yellow color) to registered events as: open/close socket server, received messages and errors that can be generated for try/catch sentence; paragraph 3.2.2 in chapter 3 referred to this design. As was explained, while server application is waiting for a client connection, is not possible to attend other systems demands, for example: update screen layout. After reviewed basic concepts about Android Apps, resolved that Android is based in threads, which is a kind of process that runs independently ones of the others, main thread is called Layout Thread because handle presentation layer and is configure by default. So a communication layer (Open and handle sockets) was transfer to an additional thread launched from presentation layer. A console to the client app is a good example. FIGURE 27 shows a screenshot to indicates correct performance of Console in Client Application.

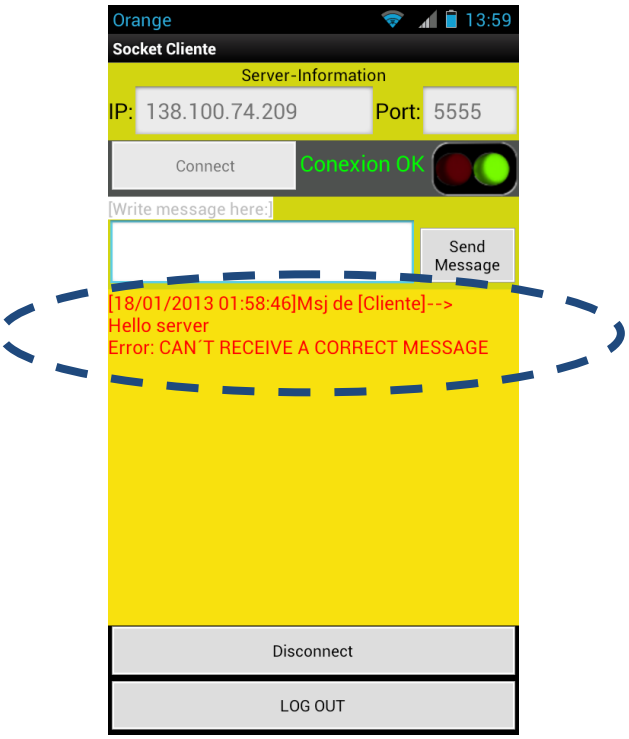


FIGURE 27: Client Application: Console

Talking about infrastructure, there is two possible ways to connect client with server:

a) *Using an Access Point* which has routing capacity and a DHCP server. AP builds the wireless network and DHCP assigns IP address to the hosts, in this part was helpful *Display WIFI Information Tool* Android application because displays IP address to set information server in client configuration. A short description about this application is included here. Logo is showed in FIGURE 28.

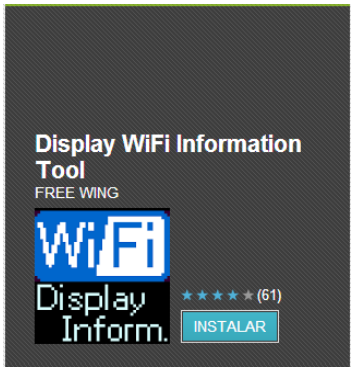


FIGURE 28: Logo Display WIFI Information Tool Android App

Features:

- Display WIFI IP Address, SSID, Connection Speed, etc.
- Very useful Simple Wi-Fi Software.
- It is an Android Toast, it is to say that is an application that shows information for a while, so can manually close Info.

Information Displayed:

- IP Address/ SSID/ Link Speed/ MAC Adrs. /Gateway Adrs./ Net Mask/ DNS1 /DNS2
- Tested Android versions: 1.5, 1.6, 2.1, 2.2, 4.0.4.
- Download link [Googleplay´02].

Example screenshot are exposed in FIGURE 29.

FIGURE 29: Display WIFI Information Tool

b) *Activating Mobile Hotspot,* it is possible to build a wireless network using smartphone; use DHCP protocol. It involves clients and a server operating in a client-server model which handles a quantity of dynamic IP address; main application is to share internet connection with clients, just what is needed. Configuration is very easy, it is required to set Network SSID and set a password. First direction of DHCP client will be 192.168.43.2, so by default the first direction is used for the device that builds network, in this case server smartphone with IP address: 192.168.43.1. FIGURE 30 shows Configure Mobile Hotspot screen.

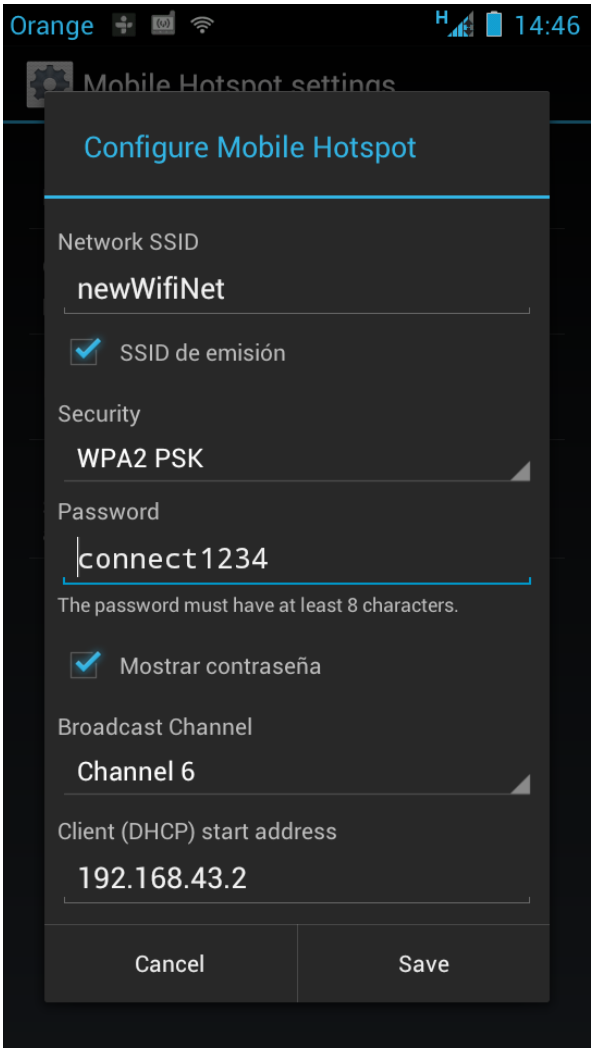


FIGURE 30: Mobile Hotspot Configuration

To finish console construction it was decided to catch IP address of the server and date/time to label events. IP address catching should consider possible infrastructures before explained. FIGURE 31 present final result of this objectives builder as was explained in 3.3 paragraphs that spoke about Additional subroutines implemented. FIGURE 32 showed events examples with additional subroutines.

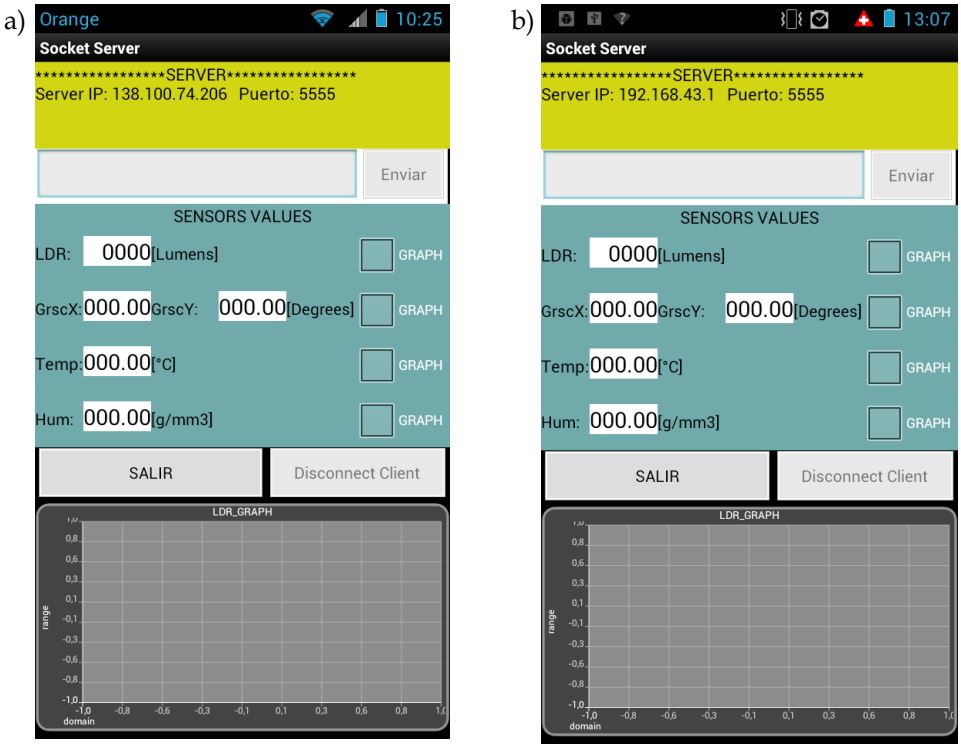


FIGURE 31: Console screenshots: a) using an Access Point

b) Activating Mobile Hotspot

A lot of tests were done to demonstrate communication performance. To complete chat model was necessary implement communication in two ways, it is server and client can speaks and listen for messages, this task are developed in this sense:

- *Server* should handle Output buffer of Socket, a simple task similar to Client text box implementation.
- *Client*, a hardest task because is necessary to carry the communication statements to a second thread, because hear the port could blocks Presentation Layer; handle Input buffer of sockets was an additional job.

FIGURE 32: Console screenshots: Catching Date/Time of events

FIGURE 33 shows screenshots of these results, a great goal of this work, this implementation is a complete chat between two android devices.

FIGURE 34 presents process handshaking.

FIGURE 35 shows examples of chatting.

FIGURE 36 is a picture of testing environment.

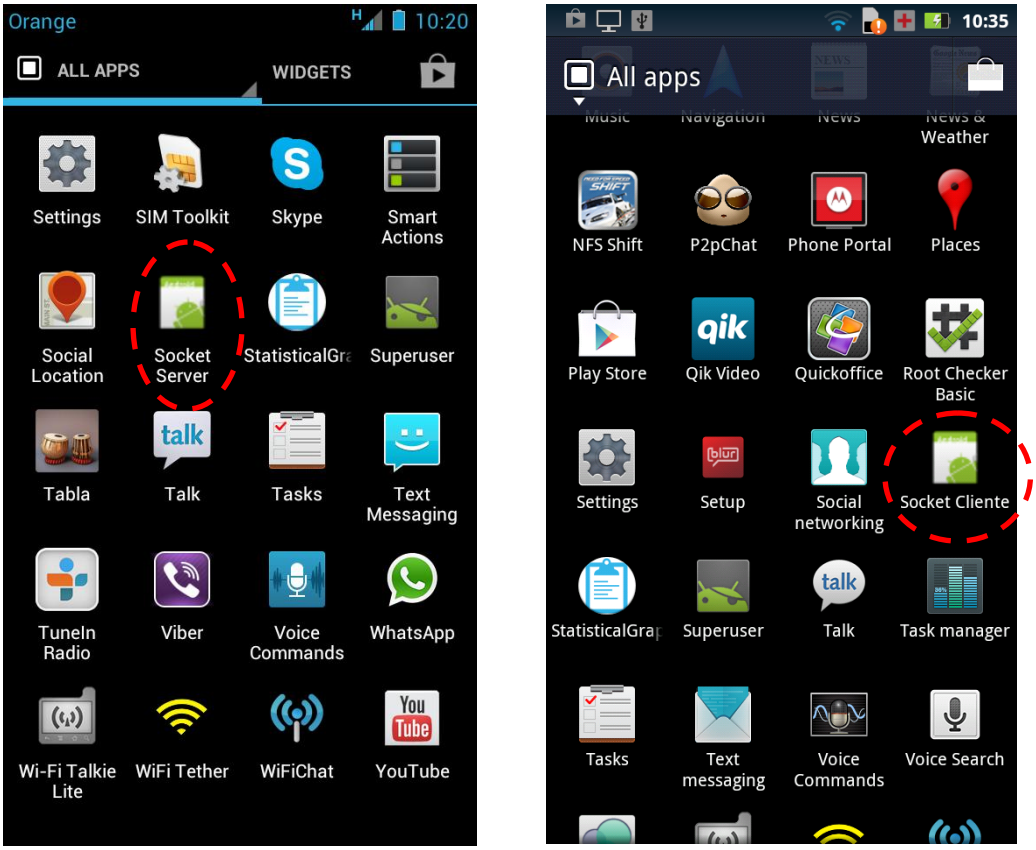


FIGURE 33: Desktop screenshots: Application Launch Icon of Server and Client

FIGURE 34: Application screenshots: Handshake

FIGURE 35: Application screenshots: Chat Examples (Part 1)

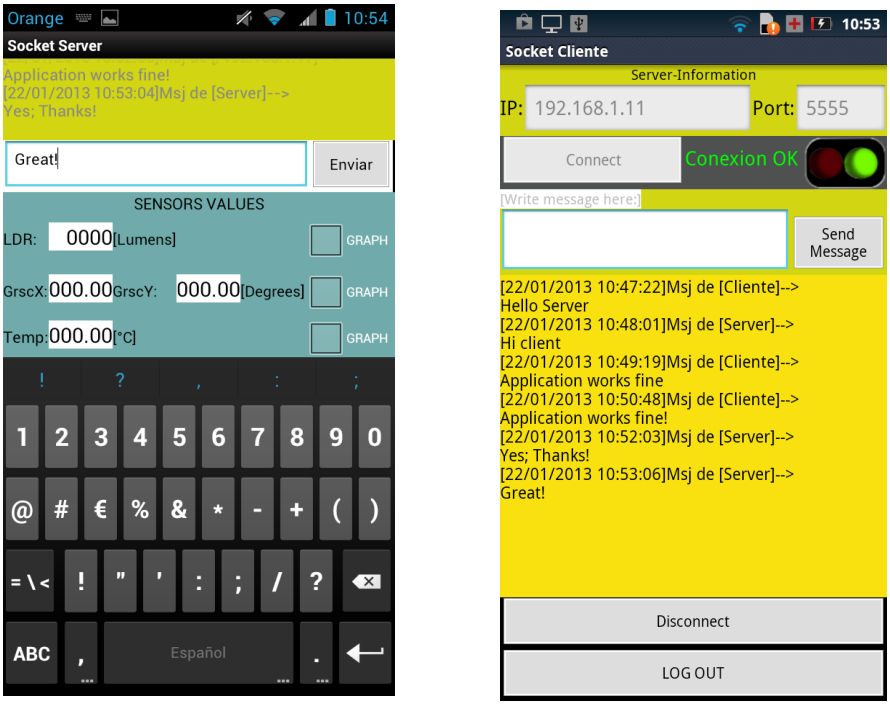


FIGURE 35: Application screenshots: Chat Examples (Part 2)

FIGURE 36: Testing environment: Client and Server Android Application

II.5 Connect Cookie with Server Android App

Cookie node is a configurable system as was described in chapter one I.2 paragraph, in this way generated a stable communication required additional settings side to side. From the Cookie Node point of view; the developer builder a program in C using µVision 3 to configure an ADUC 841 microcontroller (Analog Devices) part of Processing Layer. The program developed basically generates different commands to handle RN-131C chipset (Roving Networks WiFly radio module) available in Communication Layer and takes sensor values from the bus to transmit. The architecture of the node can be seen in FIGURE 37.

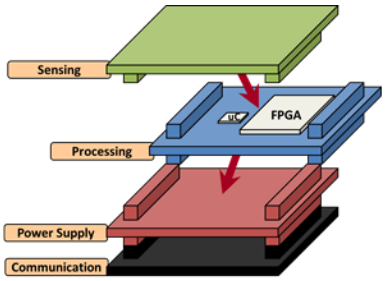


FIGURE 37: Cookies Architecture.

Tasks implemented to network configuration in program are listed:

a) Restore module to factory settings.

b) Basic Module configuration:

- UART (to interact with module thorough serial port)

  Baud rate: 19200 bps (After testing is the maximum rate that microcontroller can interact, actual architecture).

  Data bits: 8

  Parity: None

  Stop Bits: 1

- ECCHO: Disable.

- AUTOCONNECTION: Disable. Exist connection redundancy and it is not possible to send commands. After first configuration it will be available.

c) DHCP cache configuration. Module requests an IP address to DHCP server. Requested fields: SSID, Key, IP server, Protocol: TCP or UDP in this case TCP is setting.

d) Restart the module, to update RAM memory because last configuration is saved in a temporary file.

e) Connect module with Android Application.

- Open a Socket Client (it is possible to open a Server Socket too), pointed IP server and using Port.
- Wait for Connection Acknowledge from smartphone.
- Begin to send data captured by microprocessor.
- Sleep

It is possible to set WLan rate communication. There are different values available in module, as highlighted on *Table 2*. It was preferred 24 Mbps (default rate).

Table 2: Wlan rate Command Options

Value	Wireless Data Rate (Mbits/second)
0	1
1	2
2	5.5
3	11
4 - 7	Invalid
8	6
9	9
10	12
11	18
12	24 (default)
13	36
14	48
15	54

Node Cookie should connect to the network; get an IP direction from DHCP Server available in Access Point or Mobile Hotspot (Virtual DHCP Server) and establish a connection with Android Application Server, everything automatically. This part developed by Low Power WIFI group researches; as it was explained in last paragraph.

Other test was done: build the network with node Cookie, open a socket, acts like server and connect with Client Android Application, this had positives results. By side of Server Android Application develop followed three steps:

a) Connect/Disconnect node cookie and build a cycling application, that is when node Cookie disconnects should exists option to connect other node (or the same). A button called Disconnect client with attributes to disconnect (close socket and reopened) the client was implemented to give an extra possibility to user. FIGURE 38 shows client disconnection process.

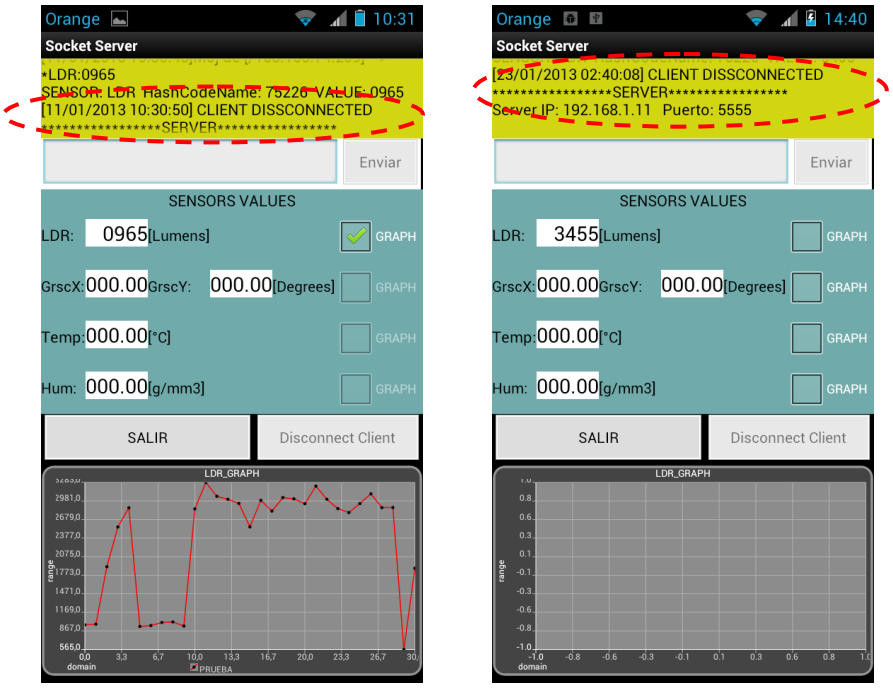


FIGURE 38: Client disconnection. Close sever socket and reopened

FIGURE 39 shows disconnection of client process after push *Disconnect Client* button.

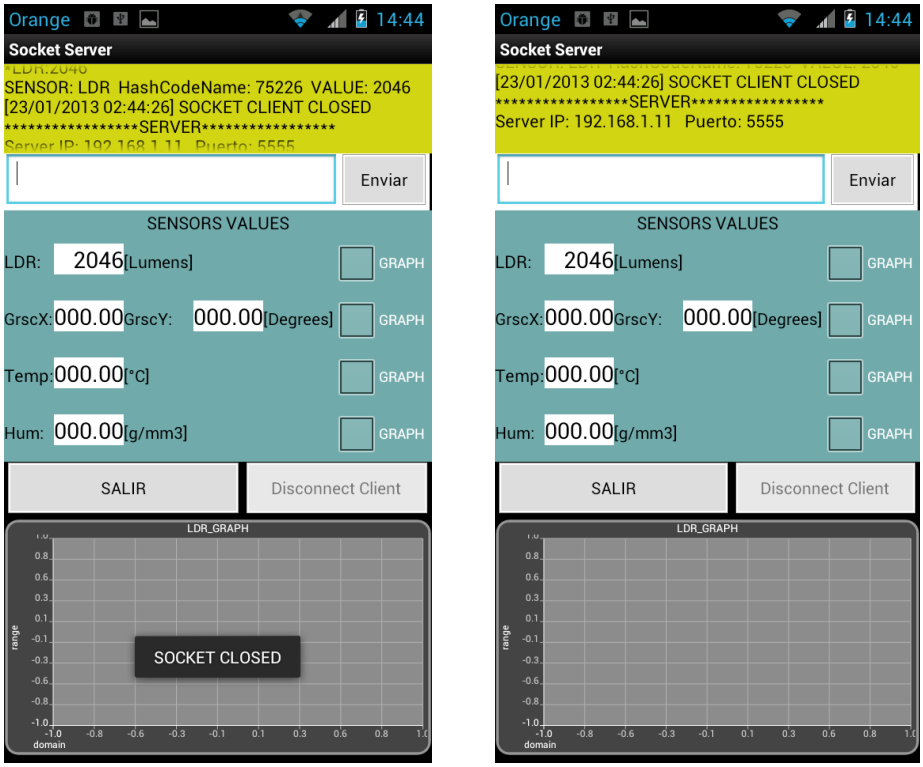


FIGURE 39: Client disconnection using Disconnect Client button

b) Build a screen to show sensors values, updated when a data arrives. In this case implemented three sources of data: temperature, rotation and humidity sensors. This part of the screen gives the user´s option to choose which variable is graphed. There is not the option to show all variables at the same time because each variable has own scale and range of values; for example LDR measure light from 0 to 4000 lumens but temperature not reach that values of degrees. Testing environment at this point includes node cookie connection with HyperTerminal of Windows to check that all messages generated by the node arrive to smartphone. FIGURE 40 explains better.

FIGURE 40: Testing environment: Acquisition sensor values

c) Build a histogram of sensor´s values to show tendencies. It means implemented a buffer of points to show using Android Plot libraries as was explained in item 3.5. Thirty points was an acceptable number to show tendencies of the variable, other test were executed with 5, 15, 20 points but the results were worse. FIGURE 41 displays results.

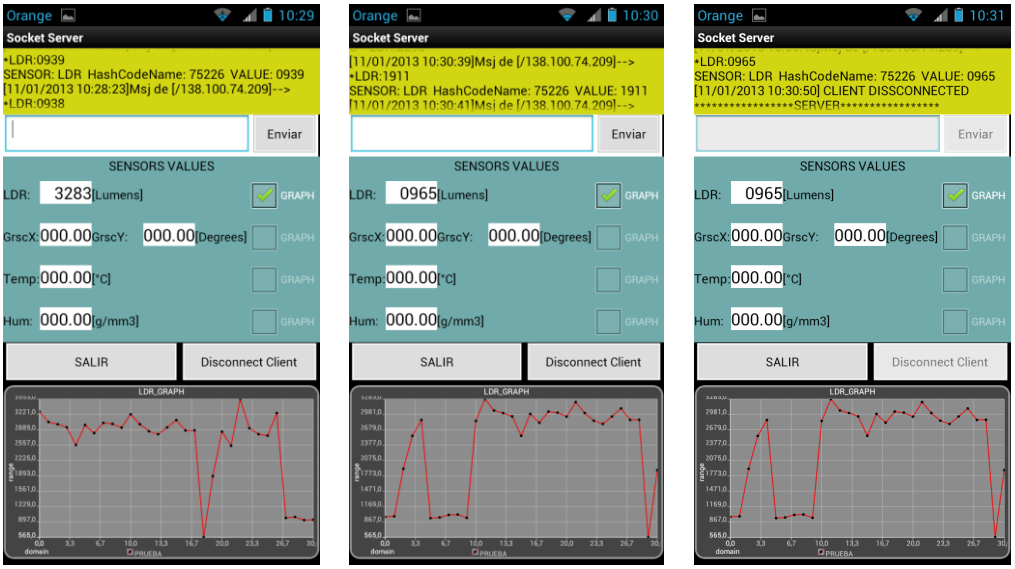


FIGURE 41: Testing environment: Acquisition sensor values

II.6 Log file test

As it was explained in 3.4 item, it was created a subroutine to generate a log file, of course a text file. It is saved in internal memory of the smartphone (but it is possible to save in SD card if it´s available and changing configuration). This file collects all history of data and handshake events. File Recovery from internal memory is a process that implies access to next address: Internal phone storage/mysdfile.txt. The file can get from the phone connecting to the computer too (FIGURE 43). Steps to recovery Log file are illustrated on FIGURE 42 from internal smartphone memory.

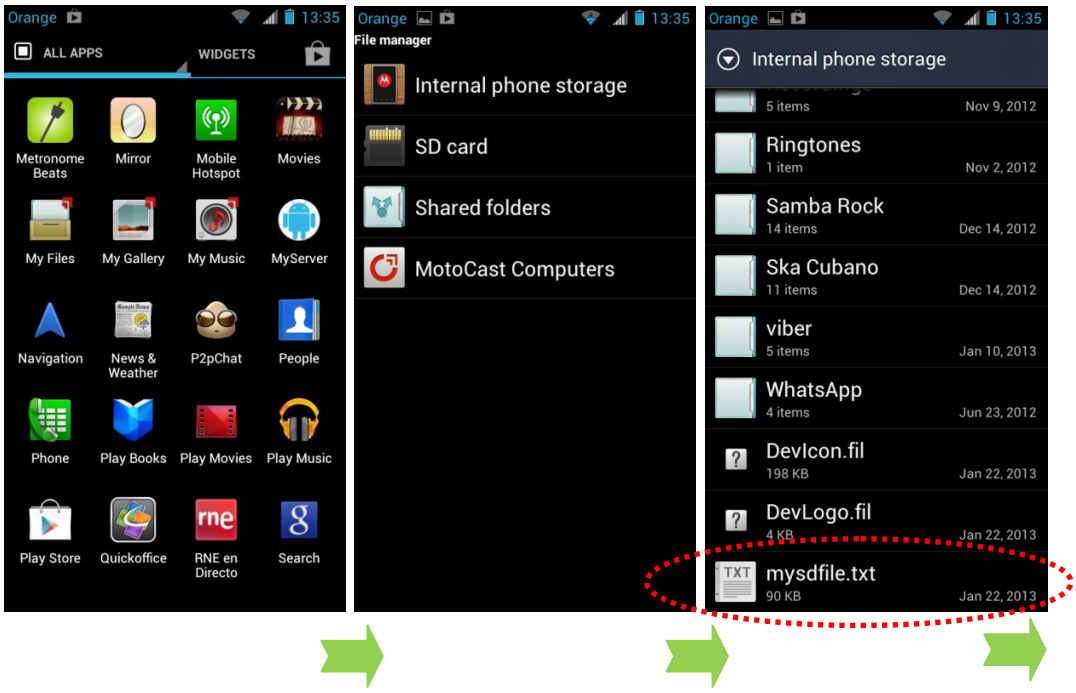


FIGURE 42: Log File recovery process from internal smartphone memory (part 1)

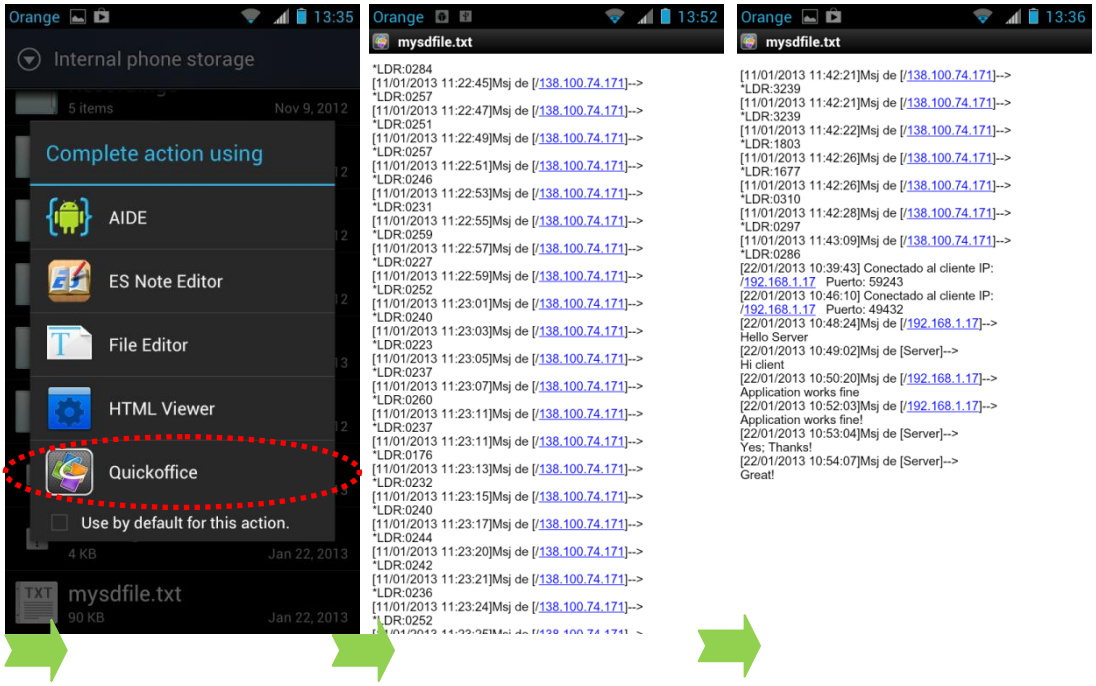


FIGURE 42: Log File recovery process from internal smartphone memory (part 2)

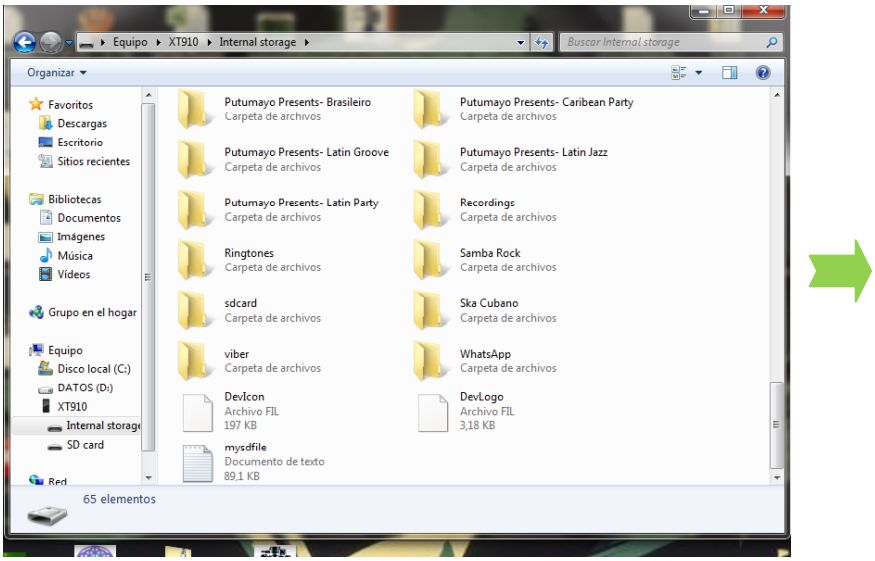


FIGURE 43: Log File recovery process using computer connection with smartphone (part 1)

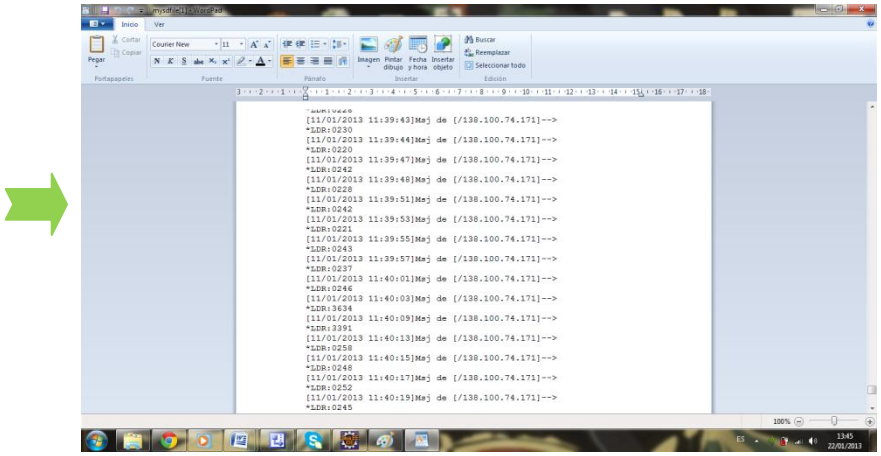


FIGURE 43: Log File recovery process using computer connection with smartphone (part 2)

II.7 Testing coverage and Power Consumption

An interested question is the coverage of a wireless communication. In this sense say that limitations are explain by IEEE 802.11 standard, however depends of many variables, which for example: working area, electromagnetic noise, infrastructure; etc. Protocol using is TCP, as is known it is a contention protocol, which devices will be the best effort to send and receive messages; of course in badly conditions it is possible to lose information. As a future line of research other protocols like UDP can be implemented and guarantee no data losses.

Power Consumption, ever is a hot topic in WSN, from the point of view of Server Android App say that is known tendency of smartphone manufactures reduce power consumption because that is a principle of mobility. Smartphone peripherals such as: WIFI greatly reduce battery of smartphone when this is connected to an Access Point. Worst results can get using Mobile Hotspot, an advice even remember this fact (FIGURE 44). A closer analysis usually gave a specific application environment but this work is oriented to generate an interface, not a strict study of power consumption issues.

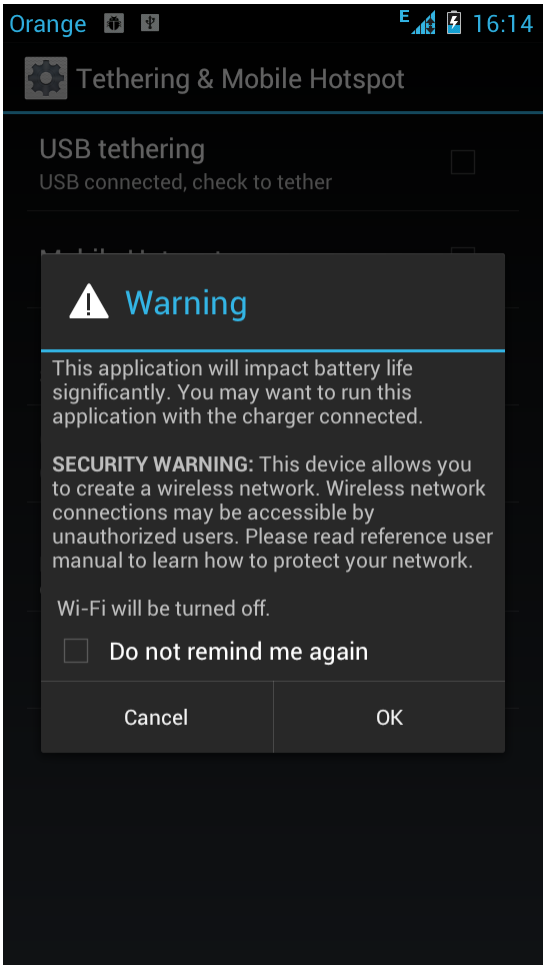


FIGURE 44: Mobile Hotspot advice about battery life

II.8 "Force Close" Handle

From point of view of developers is a serious problem on Android, no matter what hardware are using. It is a badly condition that indicates: Something is wrong in app execution. In this part it is important consider functionality states of the application, for example in Client App if CONNECT button is available and pressed without have Server Information can generate a Force Close. Many of this wrong behavior

are associated to infinite loops or incorrect user handle that a developer should consider. Avoid a Force Close is a way to guide user to follow a sequence because many of this apps are oriented to connection and a previous handshake may realized, advanced App implements handshake in backplane. Buttons, Input text boxes and any source of possible errors should be considered. Attributes of Android Widgets are a good help to avoid errors.

Server Android Application has the next states and considerations:

Application upload, while socket server is waiting for a client connection: input text box, *Send* and *Disconnect Client* button are disabled

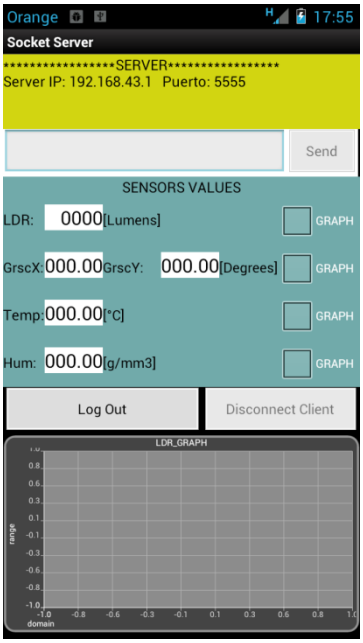
Client connected, a condition that permits interchange messages: input text box, *Send* and *Disconnect Client* button are enabled.

Client disconnected, server socket is closed and reopened to receive other client, in this sense input text box, *Send* and *Disconnect Client* button are disabled until a client connection come back.

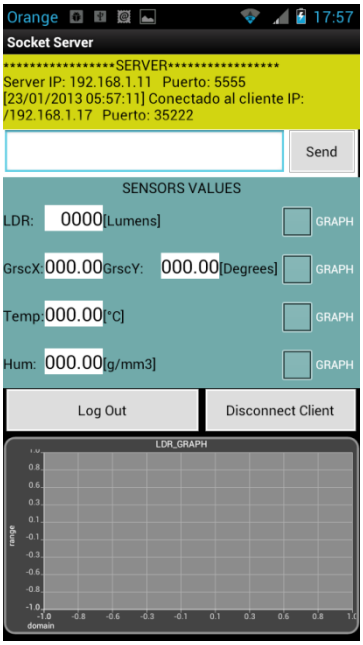
Log Out button can be pressed anytime because it is possible to logout of the application, if there is a server socket operating this is closed.

Graph button can be chosen any time because selects which is the variable to graph. It is possible to choose only one of this, for the reasons before explained (paragraph 4.2.5)

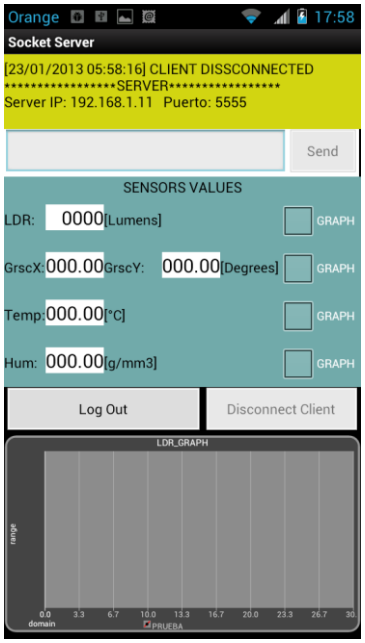
These states are showed in FIGURE 45:



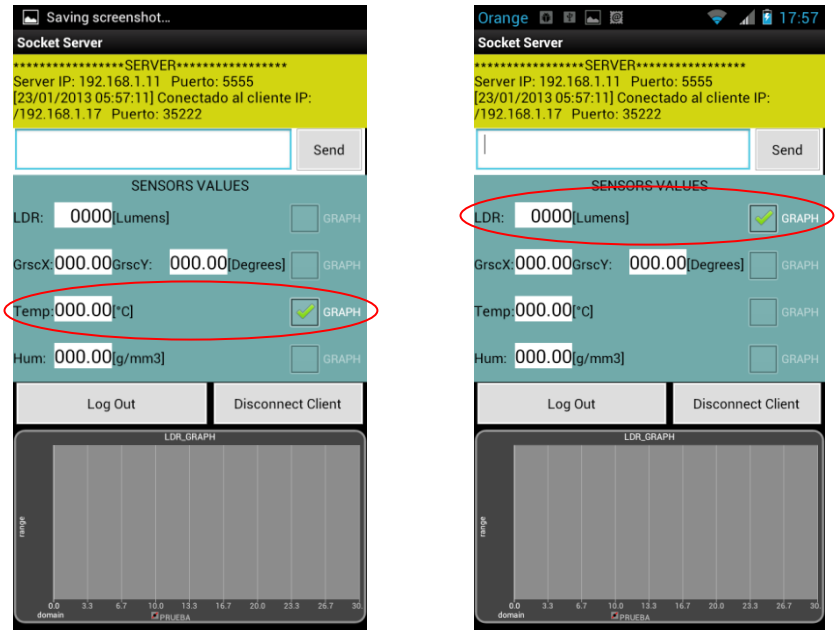
a) Application Upload



b) Client connected



c) Client disconnected



d) Graph button

FIGURE 45: Force Close Handle

Chapter 5

Conclusions and Future Scope

"Genius begins great works; labor alone finishes them."

Joseph Joubert

Conclusions and future scope

I.1 Conclusions

- In times that machines fit the human environment instead of forcing humans to enter theirs (ubiquitous computing) new generation of services such as offer Android System grow of potential of Wireless Sensor Networks (WSN). The application's main goal is to interact with a WSN, allowing a user to consult sensor states and have a low level of process and storage (histogram and log file). To do that was implemented a wireless interface through TCP sockets in a WIFI network, peripherals available in node Cookie and smartphone. Therefore the user will be notified under the same network coverage there are changes in the WSN. Apart the WSN, a base station is also needed for keeping the state of the WSN, store sensor's information and enable communication between the WSN and the developed mobile application via Web Services (email, web servers, etc).

- Eclipse tools offer a good development environment to build Android Applications; it is great communication software. It uses software development methodology which focuses on modeling and abstractions to close any concepts of common language instead of machine language. Compilation and Debugging was an easy task follow sequential steps, by other side Eclipse offers a free license and open code make it available and accessible. “Trial and error” was the main method to design. Java simplifies design process and promotes communication between entities that work in the same system. Objects, methods and class help to do things easier. Think in objects is a hard work when begins but with more expertise acquired day by day it is possible to generate interesting solutions. There are three basics files that a developer should consider at time to build an Android

Application: Android Manifest (AndroidManifest.xml), Layout (main.xml) and java program editor (.java) files.

Android Manifest is a configuration file where can set permissions to handle hardware and software resources. These permissions should be written in XML language.

Layout file handle activities presentation, it is the face that an applications shows to the user. It is possible to build this file in two ways: Using graphical interface or programming in XML language. There is a palette of widgets available to do easier things; each widgets have own attributes that to be set. Buttons, Text views, Text boxes, etc composed group of widgets that a developer can use.

Java program editor is a file written in java language where can set attributes to the widgets and generate more complex process, here it´s possible to configure the brain of the Android App.

- This Final project of Master degree has promoted research an individual analysis, handle actually tools to resolve technician problems and grow potential of Cookies platform. By other side contribute to integrate different platforms making real the Internet of things vision. In this way has been important aid and disposal of others researches groups of CEI. This way of working helps to form real scientists committed to a changing and demanding world, where there is a lot of knowledge waiting to be discovered or developed.

I.2 Future Scope

- From the point of view of networking we build a communication layer via Wi-Fi to Android devices: connect, disconnect nodes and transmit information in two ways. Actually we approach this communication to show sensor´s values, save it in a log file and brings statistic information. There are a lot of additional approaches that can develop and research.

It is possible study about build databases with the purpose of organize and manipulate information in better way. SQLite is an Open Source Database engine which is embedded into Android. SQLite reads and writes directly to ordinary disk files. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. Using a SQLite database in Android does not require any database setup or administration. It is required to define the SQL statements for creating and updating the database. Afterwards the database is automatically managed for you by the Android platform (more details in SQLite web site: http://www.sqlite.org/).

Respect of IoT (Internet of things) vision this application is the gate to share information to Internet, for example consider that a smartphone generally is on line trough WIFI or Mobile Network is possible to feed remote web server or build it inside of the smartphone (of course more restricted that a hardware dedicated to server). Web server or http server can be an industrial computer prepared and conditioned to remain connection to high speed network. Web server refers to a program, too. It is a program that attends and answers requests from browsers using HTTP or HTTPS protocols. In this sense a web server fed by this Android App can optimize WSN exploitation taking access anytime, anywhere.

Send and receive messages from Cookie nodes are a good way to share other kind of information about WSN; for examples it's possible to build a testing bench using Android resources. By other side begins to appear a lot of applications to control things (drones, for example) using Android Apps though Wi-Fi peripheral, so is a good way to introduce in robotics world.

Finally a huge spectrum of applications can be develop consider this Android App like a starting point to research about CCTV to security, monitoring environment variables (noise, temperature, humidity, gas emissions) etc, many of that implemented with WSN now.

- WIFI is only one of the possible wireless interfaces that can used to communicate WSN nodes with Android System (Serfass, D., "Wireless

Sensor Networks using android virtual devices and Near Field Communication peer-to-peer emulation", Dept. of Comput. Sci., Univ. of Arkansas at Little Rock, Little Rock, AR, USA). Modern applications can demand use of other technologies to resolve initial approach; at this point are interesting Wi-Fi Direct and NFC (Near Field Communication) standards.

Wi-Fi Direct, previously known as Wi-Fi P2P, is a standard that allows Wi-Fi devices to connect to each other without the need for a wireless access point [Wi-Fi DA]. This allows Wi-Fi Direct devices to directly transfer data between each other with greatly reduced setup. Wi-Fi Direct works by embedding a limited wireless access point into the devices, and using Wi-Fi Protected Setup system to negotiate a link. Setup generally consists of bringing two Wi-Fi Direct devices together and then triggering a "pairing" between them, using a button on one of the devices, or systems such as NFC. This technology is based in a server client infrastructure; one pair runs a wireless access point soft and choose who is the server and client, this handshake is does in backplane, so it is transparent for the user.

NFC is a set of standards for smartphones and similar devices to establish radio communication with each other by touching them together or bringing them into close proximity, usually no more than a few centimeters. NFC can be used in social networking situations, such as sharing contacts, photos, videos or files, and entering multiplayer mobile games (Pelly, Nick; Hamilton, Jeff, 10 May 2011. "How to NFC". Google I/O 2011).

From point of view of Android Developer these standards are available with objects, class and methods in Android Libraries to handle these interfaces (Android Developers), even in a more friendly way that was done present Android Application work but is necessary has appropriate hardware in the two sides of the communications. These peripherals can consider to future developments at cookies hardware, of course after a power consumption analysis, a hot topic in WSN.

References

[Lewis´08] Lewis, Jhon; Loftus, William; "Java Software Solutions Foundations of Programming Design" 6th ed.Pearson Education Inc., 2008.

[Escolar´09] Escolar Díaz María Soledad, Wireless Sensor Networks: State of the Art and Research, 2009.

[Advantech] Internet of things (IoT), Actualizing an Intelligent City *www.advantech.com.mx*

[Forster´12] Forster, A. Garg, K. ; Puccinelli, D. ; Giordano, Networking Lab., Univ. of Appl. Sci. of Southern Switzerland, Switzerland S. "FLEXOR: User friendly wireless sensor network development and deployment", World of wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium

[Allmendinger´12] Glen Allmenginder, Where is the money in the Wireless Sensor Networks? Harbor Research, May 4th 2012.

[Postscapes] Postscapes: Traking the Internet of Things, "A brief History of the Internet of Things" *http://postcapes.com*

[Casagras´07] RFID and the Inclusive Model for the Internet of Things, CASAGRAS, 2007

[Tozlu´11] Tozlu, S. Res. & Technol. Center - North America, Robert Bosch LLC, Palo Alto, CA, USA, "Feasibility of Wi-Fi enabled sensors for Internet of Things", Wireless Communications and Mobile Computing Conference (IWCMC)", 2011 7th International.

[Exupery´09] Antoine de Saint-Exupery, "Internet of Things: Strategic, Research, Roadmap", September 2009.

[Haller´09] Haller Stephan, SAP Research, "Internet of Things: An Integral Part of Future Internet", Prague, 2009.

[Mujica´12] Mujica Gabriel, "Hardware and Software Integration Platform to Testbed in Wireless Sensor Networks", Centre of Industrial Electronics, Universidad Politécnica de Madrid, March 2012

[moviThinking] What is a Mobile App? What is a Web-based mobile application or Web app?
http://mobithinking.com

[Eclipse] Eclipse home page: *http://www.eclipse.org/org/*

[JavaTuto] Java Tutorial: *http://docs.oracle.com/javase/tutorial/*

[AndroidDev] Android Developers:
http://developer.android.com/sdk/index.html

[AndroidPlot] Android Plot home page:
http://androidplot.com

[Motorola] Motorola home page:
http://www.motorola.com.mx/consumers/home

[Googleplay´01] Superuser download link:
https://play.google.com/store/apps/details?id=com.noshufou.android.su&hl=es

[Vogel] Lars Vogel, any articles about Android programation
http://www.vogella.com/articles/Android/article.html

[Googleplay´02] Display WIFI Information link download:
https://play.google.com/store/apps/details?id=jp.ne.neko.freewing.DispWifiInfo