Master in Informatics Engineering

Internship

Wise Energy

Internship Report

Intern:
Carlos Afonso Pereira de Carvalho Mota
cmota@student.dei.uc.pt

Supervisors:
Rui Pedro Paiva
DEI

Sérgio Cardana
WIT Software

Rui Oliveira
WIT Software

July 12, 2012

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

This page was intentionally left in blank.

Acknowledgements

First and foremost, the present report represents the end of a five year chapter as a student of Informatics Engineering at the University of Coimbra that had not been possible without the continuous help and guidance from my parents and grandparents to whom I owe everything I am today – a special and eternal thanks.

To both my advisors: Rui Pedro Paiva for the continuous preoccupation, encouragement and those last-minute reviews and to Sérgio Cardana for accompanying and carrying about my internship and lately to being able to solve those *while(true)* moments;

To WIT Software for creating this opportunity, the environment and most of all to its employees that gladly agreed to participate voluntarily on the usability studies.

Last but not least, to my cousin for the continuous encouragement during these last days of writing and, of course, to a special group of friends that have been present for as long as I can remember – thank you.

This page was intentionally left in blank.

Abstract

We live in a technological era. In our homes we have dozens of appliances and devices that are continuously draining the electric grid. As we grow in number, we start facing several limitations on the amount of energy that is available to use and the quality of the infrastructures that are responsible for their transmission from the power plant through our homes. If this rate is maintained we can start facing energy shortages.

Although utilities are investing in new forms of renewable energy and in the creation of a smart grid which will be responsible to adapt the electric grid to fulfil today's needs, it will take several years to be fully operational. It is mandatory to take actions now.

Consumers can act by reducing their household energy consumption, but to do so, they need to learn how to save. It is necessary to provide a deeper insight into which appliances/devices are using too much power, to monitor the current usage, to access past data in order to analyse improvements and to control remotely the behaviour of every one of these equipments.

This document represents the work done during the year-long internship for Master in Informatics Engineering graduation, at WIT Software, on this problematic. Moreover, during the development of Wise Energy Android application, the company found an inherent value on the charting components that had been developed and asked to design them has an external library for later integration on other Android applications.

Developing an application capable to run and provide the same user-experience across different android releases and smartphones/tablets presented itself as a challenge. Due to limitations either from the platform or the device it was necessary to redesign several features in order to include as much users as possible.

Keywords

"Energy", "Energy Savings", "Energy Efficiency", "Wise Energy", "Smart Energy", "Smart Grid", "Smart Meter", "Smart Monitor", "Home Control", "Home Monitoring", "Home Automation", "Android Charts", "Data Visualization".

This page was intentionally left in blank.

Table of Contents

Index of Figures

This page was intentionally left in blank.

Index of Tables

This page was intentionally left in blank.

Glossary

For the use in this document the following definitions are presented:

Acronym	Description
Android	[1] Android is a software stack for mobile devices that includes an operating system, middleware, and key applications.
Android SDK	[1] The Android SDK provides the tools and libraries necessary to begin developing applications that run on Android-powered devices.
Ant	[2] Apache Ant is a Java library and command-line tool whose mission is to drive processes described in build files as targets and extension points dependent upon each other.
Dalvik	[3] Dalvik is the managed runtime used by applications and some system services on Android.
GE Brillion Technology	"Brillion is GE's innovative technology that enables cooperative communication between networked devices and the smart grid." [4]
INSTEON	"Reliable home control and automation technology." Uses "both the existing wires (power lines) in the home and radio-frequency communication." [5]
Time-Of-Use (TOU) or variable pricing	"Rate that a utility company is charging for electricity use which changes according to the time of day." [4]
X10	"Is a communications "language" that allows compatible products to talk to each other using the existing electrical wiring in the home." [6]
Z-Wave	"Is an efficient, lightweight wireless technology designed for residential control applications." [7]
Zigbee	"Wireless technology developed as an open global standard to address the unique needs of low-cost, low-power wireless M2M networks." [8]

This page was intentionally left in blank.

Acronyms

For the use in this document the following acronyms are presented:

Abbreviation	Description
ADB	Android Debug Bridge
ADT	Android Development Tools
AIDL	Android Interface Definition Language
AMI	Advanced Metering Infrastructure
AMR	Automated Meter Reading
ANR	Application Not Responding
API	Application Programming Interface
APK	Android Package
DDMS	Dalvik Debug Monitor Server
HAN	Home Area Network
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
HVAC	Heating Ventilation Air-Conditioned
IDE	Integrated Development Environment
IP	Internet Protocol
IPC	Inter-Process Communication
OS	Operating System
PAN	Personal Area Network
PV	Photovoltaic
SDK	Software Development Kit
SDK	Software Development Kit
SMS	Short Message Service
SVN	Subversion
TOU	Time-Of-Use
UI	User Interface
URL	Uniform Resource Locator

This page was intentionally left in blank.

1. Introduction

We are always saying to ourselves… we have to innovate. We've got to come up with that breakthrough.
– Bill Gates, Microsoft co-founder.

1.1. Documentation

The present document and all its appendixes are part of the work done at WIT Software during the one year internship, for the Department of Informatics Engineering of the University of Coimbra.

This work was supervised by Rui Pedro Paiva, PhD, professor at the Department of Informatics Engineering of the University of Coimbra, and Engineer Sérgio Cardana and Engineer Rui Oliveira, from WIT Software.

Attached to this document are the following appendixes:

- WiseEnergy_-_Appendix_A_-_State_of_the_Art
  This document corresponds to a detailed description of the current solutions and technologies in the area of home energy management.
- WiseEnergy_-_Appendix_B_-_Android_Charting_Tools
  This document corresponds to a detailed analysis of the android charting tools.
- WiseEnergy_-_Appendix_C_-_Requirements_Specification
  This document corresponds to a description of the Wise Energy Android application and Charting API requirements. It contains the initial user stories, the functional, the quality and the design requirements and the research conducted on energy – domain area knowledge.
- WiseEnergy_-_Appendix_D_-_Application_Mockups_Wireframe
  This document corresponds to the Wise Energy Android application mockups and wireframes designed.
- WiseEnergy_-_Appendix_E_-_Architecture
  This document corresponds to the description of the Wise Energy Android application and Charting API architecture as well as the technical decisions made during the entire development process.
- WiseEnergy_-_Appendix_F_-_Software_Quality
  This document corresponds to the description and analysis of the software quality tests defined to evaluate the Wise Energy Android application.
- WiseEnergy_-_Appendix_G_-_Usability_Tests
  This document corresponds to the specification of the tasks written for the usability tests and describes its results.

Additional research was made in order to analyse the value that energy saving solutions can bring towards the consumer and utilities, which of those are already offered by utilities and what wireless protocols currently being used for HAN's. On this last topic the Zigbee protocol received special focus once it is the one used by the system *smart plugs* – the plug-in modules which allow direct communication with the connected devices.

1.2. Document overview

The present document is divided according to the following eight chapters:

- **Introduction**
  This section presents an overview of the internship, including contextualization, motivation, goals and risks of the current project.
- **State of the art**
  On this section it is described a list of products and services that are the result of a continuous research and analyses in the area of energy saving solutions.
- **Requirements**
  This section describes the features supported by the Wise Energy Android application, the Charting API and the technologies used during the development and testing process.
- **Architecture**
  This section describes the application architecture and provides an overview of the technical decisions made during the project development.
- **Project Planning**
  This section describes the software development methodology followed along with the initial and revised project planning.
- **Software Quality**
  This section describes the tests made to assure the project quality and to confirm that all of the defined requirements were met.
- **Conclusion**
  This section sums up the work done during the year-long internship along with the lessons learnt during the entire development process, the contributions made and the outlined future work.
- **References**
  This section states the references used on the present report.

1.3. Context

The internship took place at WIT Software, a "software company that develops software products for Mobile Operators and the Mobile Internet" [9] and it is part of the Wise Energy project which aims to create energy management solutions that will help consumers to create an energy efficient environment.

1.4. Motivation

During the past decades we have seen the world shifting towards a society that is truly dependent on energy to survive. Most people do not even realize about the amount of energy they are actually wasting – even in standby mode an electrical device is draining power from the grid. As we surpass the 7 billion people barrier on the planet and as technology evolves it is mandatory to make a more efficient use of the Earth's resources and still to reach to the entire population.

Otherwise considered the greatest engineering achievement of the twentieth century by the America's National Academy of Engineering, the electric power grid uses the same architecture and technology as when it was first created more than one 120 years ago. This leads to an outdated power grid incapable of supporting tomorrow's energy demands. Therefore it is mandatory to bring it back to twentieth one century by the creation of a smart grid.

The smart grid will work as the backbone of the entire electric system – from the power plants to the consumer's home it will be possible to track energy usage, detect failures, respond to energy demands, etc.

This concept does not end at the distribution level. Utilities are starting to update the old electric meter to a new smart meter which allows several features, including automatic readings, two way communication direction – this is particularly important to warn the consumer when the energy use is reaching the bill budget and during high peak demands to remotely turn off systems like the HVAC that drain too much power, allowing for a more reliable grid.

Until today electric bills almost did not display any insight into the home's energy use – there we only had access to an estimated energy value for the time frame between the current and past bill, the energy cost and in some utilities a chart with past consumption data. Since this information is an estimation, until today there was no reliable solution to analyse the home's energy consumption. This information is vital to understand where we are wasting energy and how can we save it.

However, having this information available on the bill report is not enough. The consumer will only have access to it from time to time slowing down the saving process.

What is the best way to provide this service? What can we add?

The smartphone is probably the only electronic device that is always in our pocket – whether it is inside or outside our homes, making it the perfect candidate to hold an energy management service. From it, we will be able to access in real time the home's energy use, the past usage data, to receive alerts/notifications when the consumption is too high and even control all appliances and electronic devices from it. This is possible to achieve either at home, the office, etc. – it only requires an Internet connection in order to access the service.

Reducing energy use will automatically reduce CO_2 levels which in turns will reduce the home's carbon footprint and of course the price of the electric bill.

1.5. Goals

The goals of this project can be divided into two major interconnected parts:

- Internship, which focus on the experience and knowledge gains for the intern.
- Project, that corresponds to the implementation of all the defined requirements.

1.5.1. Internship

From the internship point of view the goals are to consolidate the knowledge about Software Engineering and mobile development technologies, giving special emphasis to the Android platform, and to gain experience while developing software in a corporate environment with real life clients and all its implications, and learn from it.

1.5.2. Project

Today's solutions for saving energy present several limitations:

- Most of them are restricted to online access (web portal). It is more common for a user to carry a smartphone than her personal computer; hence a mobile application is the easiest solution to provide access to the home's energy consumption.
- Do not offer detailed information about how much energy a certain household appliance and/or device is using nor is it possible to control them.
- Do not monitor energy production.
- Are incapable to estimate how much energy will be used during the current time frame and how much it will cost.

The above limitations must be solved in order to provide a deeper insight into the home's energy consumption and production. Moreover, it is necessary to develop a set of home automation tools that will provide the consumer the possibility to control and access every household appliance and device reducing the overall energy use without having a significant impact on her lifestyle.

The application was developed for the android platform mainly due to its continuous growth on the smartphone market.

During the second semester, WIT Software found an inherent value on one of the project modules, which was developed to support the drawing of charts on the device's screen. Moreover, it added the possibility

to directly interact with them via user-interaction gestures. A Charting API library was developed, which provides full customization of the graphs components and offers a wide range of charts. Moreover, a sample project was created in order to demonstrate its potential.

The Charting API library project was independent from Wise Energy and it is currently being used in other Android applications developed by the company.

The Wise Energy Android application and the Charting API requirements are defined on section 3 of this document.

1.6. Risks

Risks can highly influence the outcome of a project. It is necessary to create strategies to overcome or prevent them. An analysis should be made periodically in order to avoid unpredictable results.

The following list represents the risks appointed during the project development along with the solutions created to overcome them:

- **Offline API.**

  Since the application relies on a third party API to make data requests, it is not possible to guarantee that the service will have 100% uptime.
  - This risk cannot be eliminated.
  - Mitigation plan: the application detects when the API is offline and retrieves all the necessary data from the local database notifying the client that there has been a connection problem to the API and that the current visualization mode is "offline". All the updates must be saved locally.

- **API requests.**

  Since the data presented on the application is requested wirelessly to an external server it is possible that a particular response becomes invalid due to server-side or transmission errors.
  - This risk cannot be eliminated.
  - Mitigation plan: the application should always analyse the response content for possible errors. If there are no structural problems the header status code should be matched into one of the defined response codes – if it corresponds to the success status the application view should be updated.

- **No Internet connection**

  In order to make the API's request it is necessary for the device to be connected to the Internet.
  - This risk cannot be eliminated.
  - Mitigation plan: the application should detect if there is an Internet connection available before making any request. In case there is not, the user should be informed and asked if she pretends to use the application in offline mode.

- **Different Android releases and devices**

  There is a wide variety of Android powered devices, each one with its own hardware specifications and OS release.
  - This risk cannot be eliminated.
  - Mitigation plan: the application must be supported by all of the Android devices currently on the market with the Eclair release (version 2.1) and above. Due to API limitations it is acceptable that on smartphones with older Android OS's some of the features would not be available. These must be deactivated and under no circumstances can cause an application crash.

2. State of the art

What we're going to have to do at a global scale is create a new system, so we need energy miracles. We have to drive full speed and get a miracle in a pretty tight timeline.
– Bill Gates, on 2010 Ted conference.

In the year 2000, the America's National Academy of Engineer elected the "vast networks of electrification" as the greatest engineering achievement of the twentieth century [10]. According to the academy, the electric grid was the element that made possible the development of all the other engineering advancements like the automobile, airplane, computer, space exploration, etc. Today's power grid architecture is very similar to the one designed and built more than one hundred and twenty years ago, using the technology available at that time. Nowadays, the grid is on its limits to support the current uninterruptible energy demand, incapable to transmit large amounts of power over long distances – this is especially important for renewable energy sources where usually the power stations are located far away from the cities. There are other limitations that need to be considered like security threads that can arrive from energy suppliers or cyber-attacks, the stability of the grid, the number of devices we can connect, etc.

In order to bring the electric grid to the twenty first century it is mandatory to evolve its infrastructure, thus creating the name and concept of "smart grid", a new improved electric grid, more resilient, capable of detecting failures, on the edge to self-healing and load management, predict peak times, etc.

Since "smart energy" started to be a buzzword several companies decided to contribute for this concept providing several applications for utilities, consumer's homes and companies that point towards a common goal: reduce energy consumption. Here we have a vast new line of "smart products": smart meters, smart monitors, smart automation, etc. They provide a deeper insight into consumers energy use creating a more energy-aware society.

In a few years functionalities like controlling every household device and monitor their energy use will become ubiquitous. In other words, consumers will embrace this technology that it will be used daily without even noticing.

In this section is presented a summary of the products analysed on the appendix: WiseEnergy_-_Appendix_A_-_State_of_the_Art.:

- Opower
- Google PowerMeter
- Microsoft Hohm
- first:utility
- ISA iMeter
- Onzo
- Energy Aware
- Nest Labs
- PassivSystems
- Android
- Eragy
- GE
- EnergyHub
- GridPoint
- Serious Energy
- OGE
- Green Energy Options
- Smarthome
- alertMe
- GreenWave Reality
- Motorola

2.1. Energy efficiency solutions

These products were selected according to the following metrics:

- Monitor energy use, it can be from connecting directly to a smart meter or to an external device with the same end function.
- Control every household appliance or device from a controller, online or via a smartphone.
- Detailed energy reports, which are the result of powerful algorithms used to analyse the consumers home energy use.
- Energy saving solutions for consumers, businesses and utilities.

2.1.1. Opower

[11] Opower is a platform for the utility industry. Their products provide a new insight into the home's energy use helping consumers to analyse how much energy and money they can save.

Their product line covers the following products:

- Home Energy Reports – provides personalized information on the bill report about the home's energy usage along with energy saving recommendations.
- Online Energy Management Tools – an online platform that provides a deeper insight into the home's energy consumption.
- Energy Alerts Platform – sends notifications to clients when there energy use is approaching the bill's budget limit.

2.1.2. Google PowerMeter

[12] PowerMeter was an online energy monitoring tool developed by Google with the purpose of providing a better insight to consumers about their home's energy consumption.

2.1.3. Microsoft Hohm

[13] Microsoft Hohm is an online tool designed to provide a better insight into the home's energy usage which helps consumers to understand where they are spending too much energy and how they can reduce.

2.1.4. first:utility

[14] first:utility offers three solutions in order to decrease the consumers energy usage: smart meters, online support and an energy monitoring unit. The company believes that with the right tools the consumer will make better energy saving decisions that will lead to a decrease on power consumption which is translated into a reduction of the home's carbon footprint and a lower electric bill.

2.1.5. ISA iMeter

[15] The iMeter is an energy saving solution for costumers that pretend to reduce their energy usage.

The product line is composed by:

- Power, water and gas meters, which are responsible to measure the energy/water/gas that passes through the inbound sensor.
- A display that shows, in real time, energy use and cost.
- A transmitter that sends the measured data via ZigBee to the display and to an iMeterBox which in turns is connected to the internet and transmits this information to ISA servers in order to present it on their online web portal Enerbook – a social network where users can access their past consumptions and share recommendations for energy reduction.

Additionally, there is the iPlugMeter that monitors energy consumption of a specific plug and allows to remotely controlling appliances, for example to alternate between on and off status.

2.1.6. Onzo

[16] Onzo develops energy saving solutions for utilities: the smart energy kit and a set of analytic tools capable of processing large amounts of energy readings.

The kit was built targeting consumers that still have the old electric meter installed providing them the possibility to monitor their energy use. Therefore, in the set is included a sensor (measures electricity flow), a transmitter (sends the data wirelessly) and a display (shows the current consumption data).

Clients with a smart meter can log directly on the web portal.

Onzo also created an energy analytics platform capable of detecting which appliances are consuming energy solemnly based on the global readings – this is particularly important since it can be used not only to send notifications when the client is using too much energy but also when a certain appliance is not working correctly.

2.1.7. Energy Aware

[17] Energy Aware The Power Tab™ is a simple display for homes with a smart meter that shows the current energy use, the price and TOU pricing period.

2.1.8. Nest Labs

[18] Nest™ Learning Thermostat™ is a self-programming thermostat that learns the home's right temperature through the user's (initial) regulation.

It takes one week – configuration period, to learn what the daily right temperature is and maintains it through the whole season. It learns every time someone regulates the thermostat, updating usage patterns and temperature schedules, leaving the home more comfortable and saving energy.

Nest connects to the Wi-Fi, so it's possible control remotely – from the web or a smartphone.

2.1.9. PassivSystems

[19] PassivSystems has a product line dedicated only to energy savings – the PassivEnergy:

- PassivHub, has the double function of working as a gateway, responsible for establishing and communicating with every household device and connect to the Internet in order to send its data into PassivLiving™ for online access.
- PassivController, a remote display that supports remote control for the HVAC and solar PV system.
- iOS app, offers the same functionality as the PassivController on an iOS device.
- PassivLiving™, displays the home energy consumption usage online.
- Other products like temperature sensors, meters, etc. that contribute to a smarter and safer environment.

2.1.10. Android

[20] Android@Home is a new branch of the Android project that aims to reach home automation. Users will be able to control lights, irrigation systems, etc.

2.1.11. Eragy

[21] Eragy offers energy solutions for utilities, homes and companies interested in reducing their consumption usage.

For consumers they provide myEragy Home Energy Monitoring, an online platform that displays the home consumption levels and cost.

For utilities and companies, Eragy offers the same features as for costumers but in a larger scale – monitor every consumers home, receive notifications of energy usage outside a normal range and compares past data in order to quantify how much energy someone is saving.

2.1.12. GE

[22] GE uses the GE Brillion Technology along with GE Nucleus and smart meters in order to provide energy efficiency solutions for the customers.

The Nucleus works as a gateway which communicates with the home smart meter and/or other smart appliances, capable of receiving and storing consumption data in near real-time and projecting bill costs for the next years.

GE also developed a group of appliances integrated with their Brillion Technology – displays, dishwasher, fridges, etc. that communicate with Nucleus providing energy use information per device and present several features that help the consumer to save money – for example, the refrigerator defrosting cycles are scheduled to be executed automatically when the electricity is cheaper. Additionally, GE developed a software tool and and iOS application that communicates directly with Nucleus in order to monitor the home's energy.

2.1.13. EnergyHub

[23] EnergyHub product line consists in several products that together create an energy efficient HAN. The Home Base connects wirelessly to the home's smart meter and to the sockets (plugs) and strips in order to receive, in real time, the current energy use. It also provides a wireless thermostat making it possible to control remotely from the web portal – Mercury Smart Thermostat Platform, or an Android/iOS device.

For homes that do not have a smart meter consumers can acquire a Wireless CT Sensor that provides the same functionalities.

2.1.14. GridPoint

[24] GridPoint Home Energy Management Solution provides a set of tools to utilities for reducing power demand by, remotely, shutting down consumers HVAC systems or other draining appliances in order to stabilize the grid – load management; and for the consumer a new insight into the home's consumption usage, providing information that will lead to power savings.

2.1.15. Serious Energy

[25] Provides a set of solutions for energy management targeted to buildings that go from applications designed to monitor, analyse and control power consumption to green solutions like soundproofing products.

2.1.16. OGE

[26] OGE is an American utility that offers an online web portal (myOGEpower) to their clients, so they can have access to the home's energy consumption data daily.

2.1.17. Green Energy Options

[27] Set of products for utilities and consumers that provide a deeper insight into the home's energy usage and production:

- In-home displays that communicate with the home's smart meter and/or other energy monitor device and show the current power consumption. Additionally, it is possible to turn on/off up to six connected appliances.
- Solo PV which displays the energy being produced.
- MyEnergy is an online web portal that displays how much energy is being used and an iOS application that apart from monitoring energy allows turning on/off a group of devices.

2.1.18. Smarthome

[28] Smarthome offers a wide range of products that makes possible to remotely control every appliance and device in the home. Turning on/off or adjust dimming of a specific equipment requires to connect this device to an external controller that is plugged into a power source. These controllers are connected to the home's Wi-Fi allowing the user to control them via the in-home display or an iOS device (MobiLinc).

2.1.19. alertMe

[29] alertMe provides three lines of products related with energy savings:

- SmartEnergy which is a set of products that communicates with a monitoring device in order to obtain, in real time, the home's energy use. This information is transmitted to the hub which in turns sends to the in-home display and to the alertMe servers in order to be available online.
- SmartMonitoring, is targeted to provide an extra layer of security. Supports activate/deactivate the home's alarm remotely and warn the user every time a problem is detected (through the use of sensors – smoke, gas, etc.).
- AlertMe for iPhone is an application that connects the smartphone to the AlertMe hub which in turn provides access to products mentioned above: monitor the home's energy, control devices and receive security alerts.
- Web portal, shows information in real-time about the home's energy consumption, cost prediction, carbon footprint, home temperature and connected appliance status and power usage.

2.1.20. GreenWave Reality

[30] GreenWave Reality provides a set of products for utilities and consumers that allow to monitor and control household appliances and devices in order to save energy and money without having significant impact on the client's lifestyle.
The product line is divided into two categories:

- Energy Management Products, which is a set of plug-in devices, in-home devices and a hub. Together they allow to control and monitor the home's energy consumption and connected devices
- Intelligent Lighting, a set of light bulbs that contain the NXP Green-Chip™ and use JenNet-IP which makes it possible for the hub to communicate with the bulbs via IP instead of Zigbee, Z-Wave, X10 or other HAN protocol.
- iOS application, monitor and control the home's energy use from the iPhone/iPad.

2.1.21. Motorola

[31] Motorola's 4HOME platform provides a set of solutions to control, monitor and secure the home:

- 4Home Control Home Management allows to control the status of a variety of appliances and devices.
- 4Home Security and Monitoring, monitors and secures the home.
- 4Home Energy Analysis and Management, monitors and analyse energy consumption readings.

Additionally, it is possible to add a large set of devices which use the same open standards providing a new number of features.

2.2. Comparison

In order to compare and contrast the analysed products it was necessary to define a set of metrics.

Smart metering
If it is possible to communicate directly with the home smart meter or it is necessary to acquire an energy monitoring device.

Smart monitoring
If it is possible or necessary to communicate with an energy monitoring device in order to measure the home's power consumption.

Energy per device
If it is possible to access to the household appliances and devices in order to control them and/or monitor their energy use.

Energy produced
If it is supports monitoring the amount of energy produced by a renewable energy source.

Energy analysis
The application is capable to analyse the consumption readings and provide a deeper and accurate insight into the home's energy usage.

Energy comparison
If it is possible to compare the energy usage in two different time frames.

Estimate energy and cost
If the application is capable to project the amount of energy that will be used at the end of the month and how much it will cost.

Real-time data
If the data obtained from the smart meters and/or the monitoring device is displayed in real time. Due to delay times in the communication and since it is an acceptable value, every product that has an interval equal or less than three seconds (near-real time) was included in this field.

Automation
If it is supports access and control the household appliances and/or devices.

Neighbours' comparison
If the application compares the home's energy use with its neighbours.

Data visualization
Corresponds to the solution found to display the monitored data – if it easy to read and access or not.

Recommendations
If there is a recommendation engine that displays a set of energy saving solutions.

Notifications
If the user is warned when the home's energy usage surpasses a certain defined value.

Bill information
The application provides access to the current and/or past energy bills.

Platform
Shows the platforms where the product is available.

Functionality	Opower	PowerMeter	Hohm	first:utility	iMeter	Onzo	Energy Aware	Nest Labs
Smart metering	✓	✓	✓	✓	✓	✗	✓	✗
Smart monitoring	✓[1]	✓[1]	✓[1]	✓[1]	✓	✓	✗	✓
Energy per device	✗	✓[1]	✗	✗	✗	✓	✗	✗
Energy produced	✗	✗	✗	✗	✗	✗	✗	✗
Energy analysis	✓	✗	✓	✓	✗	✓	✗	✗
Energy comparison	✓	✓	✓	✗	✗	✓	✗	✗
Cost information	✓	✓[2]	✓	✓	✓	✓	✓	✗
Estimate energy and cost	✗	✓	✗	✗	✗	✗	✗	✗
Real-time data	✗	✓	✓[1]	✗	✓	✓	✓	✓
Automation	✗	✗	✗	✗	✗	✗	✗	✓
Neighbours' comparison	✓	✓	✓	✗	✗	✓	✗	✗
Data visualization	✓	✓	✓	✓	✓	✓	✗	✗
Recommendations	✓	✓	✓	✗	✓	✓	✗	✗
Notifications	✓	✗	✗	✗	✓	✓	✗	✗
Profiles	✗	✗	✗	✗	✗	✗	✗	✗
Platforms	Online	Online	Online	Display/Bill report/Online	Display	Display/Bill report/Online	Display	Online/iOS/ Android

Table 1 - Products comparison 1

[1] Support, but it is necessary to purchase a monitoring device.
[2] Depends on the partnership with the utility or if the user enters this information.

Functionality	PassivSystems	Android	eragy	GE	EnergyHub	GridPoint	Serious Energy
Smart metering	✗	?	✓	✓	✓	✓	✓
Smart monitoring	✓	?	✓	✓[3]	✗	✗	✓
Energy per device	✗	?	✗	✓[3]	✓[4]	✗	✓
Energy produced	✓	?	✓	✗	✗	✗	✗
Energy analysis	✗	?	✗	✓	✗	✗	✗
Energy comparison	✓	?	✗	✓	✓	✓	✓
Cost information	✗	?	✓	✓	✓	✗	✗
Estimate energy and cost	✗	?	✗	✓	✗	✗	✗
Real-time data	✓	?	✓	✓	✓	✓	✓
Automation	✓	✓	✗	✓	✓	✗	✓
Neighbours' comparison	✗	?	✗	✗	✗	✓	✗
Data visualization	✗	?	✓	✓	✗	✗	✓
Recommendations	✗	?	✗	✓	✗	✓	✗
Notifications	✗	?	✓	✓	✗	✗	✗
Profiles	✓[4]	?	✗	✗	✗	✗	✗
Platforms	Online/Display/ iOS	Android	Online	Online/iOS	Online/iOS/ Android	Online	Online

Table 2 - Products comparison 2

[3] Available only for appliances that use GE Brillion Technology
[4] Only for HVAC systems

Functionality	OGE	GEO	Smarthome	alertMe	GreenWave	Motorola	Wise Energy
Smart metering	✓	✓	✗	✗	✗	✓	✓
Smart monitoring	✗	✓	✓	✓	✓	✓	✓
Energy per device	✗	✗	✓	✓	✓	✓	✓
Energy produced	✗	✓	✗	✗	✗	✗	✓
Energy analysis	✗	✗	✗	✗	✗	✓	✗
Energy comparison	✓	✓	✓	✓	✓	✓	✓
Cost information	✗	✓	✗	✓	✗	✓	✓
Estimate energy and cost	✓	✗	✓	✓	✗	✗	✓
Real-time data	✗	✓	✗	✗	✓	✓	✓
Automation	✗	✓	✓	✓	✓	✓	✓
Neighbours' comparison	✓	✗	✗	✓	✗	✗	✗
Data visualization	✓	✓	✓	✓	✓	✗	✓
Recommendations	✓	✗	✗	✗	✗	✗	✓
Notifications	✗	✗	✗	✓	✗	✓	✓
Profiles	✗	✗	✓	✗	✗	✓	✓
Platforms	Online	Online/ Display/ iOS	Display/iOS	Online/ Display/ iOS	Online/ Display/ iOS	Online/ TV/ Display/Android	Android

Table 3 - Products comparison 3

The Wise Energy project provides the necessary insight into the home's energy consumption and production. Since data is displayed in near-real time consumers can analyse the result of their actions when turning on/off a specific household appliance and/or device. Moreover, the application provides an accurate projection of how much energy it is predicted to be used at the end of the month and how much it will cost.

Furthermore, since several equipment's can be connected at the same time, the user has the possibility to control a set of them within a single gesture either by activating any of the four available profiles: home, out of home, going home, sleep or a user defined one.

Although, many of the previous products are only available to the iOS platform, according to [32] almost half of the smartphones on the market are powered by Android, which makes it the right development platform for an energy saving application which aims to reach the highest number of users.

3. Requirements

The most difficult part of requirements gathering is not the act of recording what the user wants; it is the exploratory development activity of helping users figure out what they want.
– Steve McConnell, author of Code Complete.

3.1. Introduction

The first stage of software development projects is the definition of the project requirements. These will be used as a guideline during the entire development phase. In this work, the first approach was to write small user stories that reflected the system behaviour which led to a more formal definition.

During the application development, WIT Software found an inherent value on the charting components that were been developed for the Wise Energy project and requested to design them as an external library for later integration on other Android applications. A new set of requirements were defined and it is presented on section 3.3 of this document.

Note: More detailed information about the application and library requirements as well as the user stories and the domain area research made is available on the following document annexed to this report: WiseEnergy_-_Appendix_C_-_Requirements_Specification.

3.2. Wise Energy

Wise Energy is an Android application focused on home energy efficiency. It provides near-real time information about the current energy being used and produced, access to past values, supports controlling the connected modules by turning them on/off, scheduling standby periods, etc. along with other set of features defined on the present document. The requirements defined on this section correspond to the ones defined for the Wise Energy project.

3.2.1. Functional Requirements

This section describes which features the application should address and correspond to a deeper analyse and elaboration made on the previous defined user stories (defined on the WiseEnergy_-_Appendix_C_-_Requirements_Specification document).

They are divided according to the following groups:

- Alerts.
- Authentication.
- Categories.
- Energy readings.
- Plugs.
- Profiles.
- Subscription.
- Tariff.

Alerts

The following requirements correspond to the set of operations available for alerts.

Name	Requirement
Get all alerts	The application must be able to get a list of all the available alerts.
New alert	When the application receives a new alert from the server it must immediately notify the user.
Remove an alert	The user must have the possibility to remove an alert from the list.
Mark an alert as read/unread	An alert can be marked as read/unread.

Authentication

All the functionalities present on the application can only be accessed after the user is authenticated.

Name	Requirement
User login	A user must be able to login with a valid username and password.
User logout	A user must be able to terminate a session.
Timeout	The session expires if the user is inactive for a certain period of time and the "remember me" option is not set.
Timeout with "remember me"	The application automatically re-authenticates the user if the "remember me" option is active.

Categories

The following requirements correspond to the set of operations available for categories.

Name	Requirement
Get all categories	The application must be able to get a list of all the available categories.
(Sub-) Category icons	The application must be able to match the (sub-) categories received from the server with a set of available icons.

Energy Readings

The following requirements correspond to the set of operations for data visualization.

Name	Requirement
Request energy readings	The application must be capable to request the following energy related data: • Energy consumption • Energy production • Power • Cost prediction The same must be applied when the unit is changed from currency (€) to energy (kWh) as defined on FR-GB-01.
Request energy readings (filtered)	When a request is specific for a certain plug, the application must be capable to build it accordingly. This will avoid wasting time parsing the response and selecting the relevant information.
Display energy usage and production on a bar chart	The application must be capable to display the energy use and production on a bar chart. Moreover, this information can be grouped by different time stamps – 15 minutes intervals, a day or a week.
Display the energy usage divided by TOU rates on a stacked bar chart.	The application must be capable to display the energy usage divided by TOU (Time-Of-Use) rates on a stacked bar chart.
Swipe to change chart type	The application must support swipe for an easier navigation between two charts.

Global

The following requirements correspond to a set of functionalities that must be available to the entire application.

Name	Requirement
Requests unit	The user must be able to select the unit type in which she pretends to see the data: • System currency (for example: euros). • Energy unit (kWh).

Plugs

The following requirements correspond to the set of operations available for managing plugs.

Name	Requirement
Get all plugs	The application must be able to get a list of all the connected plugs.
Add to favourites	The user must be able to get a list of all the connected plugs that are tagged as favourites.
Turn on/off a plug	The user must be able to change the current plug status.
Modify the plug's category	The user must be able to attribute a different category to a plug. Since every category has an associated image, the plug icon will change.
Modify the plug's name	Modify the plug's name.
Standby scheduling	The user must be able to define a standby scheduling for a specific plug. During the defined time frame if the energy measured is bellow a defined threshold the device will be turned off. When this period ends it will be turned back on. The entire process automatic.
Standby threshold	The user must be able to define a standby threshold. During the scheduling time frame, if the energy being used is bellow that value the device will automatically be turned off.
Block a plug	It must be possible to lock a certain plug. If this option is set, the device status cannot be changed and it is not possible to modify its schedule.

Profiles

The following requirements correspond to the set of operations available for profiles.

Name	Requirement
Get all profiles	The application must be able to get a list of all the available profiles.
"User" profile	A default user profile with all the connected plugs added with their current configuration must be available.
Profile configuration limitation	It is only allowed to have one profile for a specific configuration. Even with different profile names.
Select a profile	The user must be able to select a different profile from the list.
Remove a profile	The user must be able to remove a specific profile from the list.
Update an existing profile	The user must be able to update the change a profiles name and its plugs configuration.
Create a profile	The user must be able to add a new profile to the list.

Subscription

This section corresponds to the requirements for the subscription settings.

Name	Requirement
Get the client subscriptions	The application must be able to retrieve a list of all the subscriptions available to the client.
Load subscription-dependent content	The application must only load the content for the available subscriptions.

Tariff

The following requirements define the application functionalities related with the tariff scheme and the home's energy information.

Name	Requirement
Display instant usage	The application must display the current energy consumption.
Display contracted power	The application must display the user's contracted power.
Animated energy gauge	The gauge arrow presented on the main activity must rotate according to the current energy consumption and the max power value defined.
Tariff type	The application must display the current tariff type.
Tariff colour scheme	The energy gauge presented along with the text colour must vary according to the current tariff. The corresponding values are defined in an external document.

3.2.2. Quality Requirements

This section focus on the quality requirements defined for the Wise Energy Android application. They define the system constraints and properties and are grouped into the following categories:

- Compliance.
- Performance.
- Platform compatibility.
- Reliability.
- Robustness.
- Security.

Compliance

This section covers quality requirements related with regulatory compliance.

Name	Requirement
Application design	Unless a design decision goes against one of the defined requirements of this chapter, the application should follow, at all times, the Android architecture and structure.

Performance

This section covers quality requirements related with performance.

Name	Requirement
Requests specification	Requests must not be done from the UI thread.
Request-response time	Every request must have a timeout associated.
Idle status	When the device is idle (the screen is turned off/ locked) the background services should obey the same status – stopped. This will save the device's battery life.
Detect widget on home screen	In order to save resources, the application must be able to detect if the widget is active on the home screen.
Detect user on home screen	In order to save battery life, the widget should only make data request when the user is on the home screen.
Wi-Fi and 3G networks	The application must be capable to detect either the device is connected via a Wi-Fi or a mobile network in order to adjust the background requests (services) time interval.

Platform compatibility

This section covers quality requirements related with platform compatibility.

Name	Requirement
Application Android OS target	The application must be supported for Android devices above the 1.6 release (codename Donut).

Reliability

This section covers quality requirements related with reliability.

Name	Requirement
Corrupt data detection	If the data received is corrupted it should be discarded and a new request should be made.
User information	The result of a request must be consistent with what the application UI shows.
Terminate session	When the user logouts the application it is necessary to clean all the associated resources.
Activity flow and user actions	The activity flow and the user actions should remain constant across the entire application.

Robustness

This section covers quality requirements related with robustness.

Name	Requirement
Fault-tolerant	The application must be fault-tolerant. When a request fails, data is invalid, internet connection is lost, etc., all these events must be treated gracefully without creating any sort of impact on the application.

Security

This section covers quality requirements related with security.

Name	Requirement
Credentials encryption	The application must encrypt the user's password.
Data access from external sources	The available data (from requests, cached, shared preferences, database, etc.) must be accessible only from inside the application.

3.2.3. Design Requirements

The design requirements defined on this section correspond to the application design which is available on the appendix WiseEnergy_-_Appendix__D_-_Application_Mockups_Wireframe. This document is part of the throwaway prototyping phase and includes: screen map (a hierarchy tree with every activity/view that will compose the application as well as the navigation pattern chosen), mockups and wireframes.

Name	Requirement
Tab widget visibility	If the user is authenticated and the device is on portrait the tab widget must be visible at all times.
Pressing back from a sub-activity	When the user presses back from a sub-activity, the application must redirect the user to the previous activity on the stack.
Tab widget click	There are three possible scenarios for this requirement: • If the user presses on a different tab widget than the one is currently at it the application should load that view. • If the user presses the tab widget where she is currently at, nothing happens. If the user presses the tab widget and the current view is a sub-activity from that same tab, the application should return to the activity that it is hierarchically above.

3.2.4. Domain Area Knowledge

This section summarizes the knowledge needed to acquire during the project development.

Since Wise Energy deals with information from other domain areas, like physics, it was fundamental to conduct a background research about several subjects, namely energy and power and how this two correlate. This information is available on the following document annexed to this report: WiseEnergy_-_Appendix_C_-_Requirements_Specification.

3.3. Charting API

The Charting API is an Android library initially developed for the Wise Energy application in order to add graphical support to data visualization. At this moment, it is independent from the initial project and is being used on other Android applications developed by WIT Software.

It provides full customization of the graph components and offers a wide range of charts. A sample project was created in order to demonstrate its potential (more information is available on the WiseEnergy_-_Appendix_E_-_Architecture, section 2.2.2).

3.3.1. Functional Requirements

This section corresponds to the functional requirements defined for the Charting API library. They describe which features the project should address and correspond to a deeper analyze and elaboration made on the previous defined user stories.

They are divided according to the following groups:

- Chart types.
- Global.

Chart types

The following requirements correspond to the set of available charts on the library.

Name	Requirement
Bar graphs	The library must be capable of drawing a bar chart.
Stacked bar graphs	The library must be capable of drawing a stacked bar chart.
Grouped bar graphs	The library must be capable of drawing a grouped bar chart.
Line graphs	The library must be capable of drawing a line chart.
Doughnut graphs	The library must be capable of drawing a doughnut chart.
Pie graphs	The library must be capable of drawing a pie chart.
Charts orientation mode	The functionalities available on portrait (device vertically) must also be present on landscape mode (device horizontally).
Different datasets size	The library must be capable to detect cases where different series have different datasets sizes and adapt them to properly draw the defined chart(s).
Null/empty datasets	The library must be capable to detect cases where one or more datasets are null or empty without raising exceptions. The user should have the chance to define if she wants to display a “no data message” or not.

Global

The following requirements correspond to the set of general features available on the library.

Name	Requirement
Add series	It must be possible to add more than one series to the library. Moreover, it should support having different types of chart series.
Remove series	It must be possible to remove a specific series from the library. Additionally, it is a nice to have an option to remove all the series from a specific chart type.
Update series	It must be possible to update a specific series.
Multiple charts selection	When there is more than one series it must be possible to choose which one the selection applies.
Gesture detection	The library must be capable to detect different type of user gestures: • Single and multiple gestures at the same time. • Click, touch, drag and pinch events.
Select a smaller timeframe window	The user must be able to select a smaller window via multi touch gestures.
Scrollable content	The user must be able to scroll horizontally through the chart content when the series size is larger than the screen dimensions or a viewport is defined.
Real time data	The library must support a continuous dataflow of values.
Morphing two or more series	The library must support morphing two existing graphs into one or a graph with a new series.
Auto-adjust to fill the screen dimensions	The library must be capable to generate a chart that will automatically resize itself to fill the screen dimensions.

3.3.2. Quality Requirements

This section focus on the quality requirements defined for the Charting API library. They define the system constraints and properties and are grouped into the following categories:

- Platform compatibility.
- Security.

Platform compatibility

This section covers quality requirements related with platform compatibility.

Name	Requirement
Application Android OS target	The application must be supported be Android devices superior to the 1.6 release (codename Donut).

Security

This section covers quality requirements related with security.

Name	Requirement
Data access from external sources	The available data to render the chart (series) must be accessible only from inside the application.

3.4. Technologies

The following sections describe the technologies used during the project development.

3.4.1. Pencil

[33] Pencil is an open source tool designed for making diagrams and GUI prototyping. In order to design the Android mockups, it was necessary to download an external stencil available on [33].

3.4.2. SmartDraw

[34] SmartDraw is a software tool that supports the creation of software architecture diagrams. It was used to draw a large part of the report UML diagrams.

3.4.3. eUML2

[35] eUML2 is a plug-in for eclipse that supports the design of UML 2.1 diagrams.

3.4.4. PowerDesigner

[36] PowerDesigner is a data modeling software tool used to design the application database.

3.4.5. Hudson

[37] Hudson is a continuous integration engine that allows building and testing software projects continuously and monitoring the execution of external-ran jobs.

3.4.6. Python

[38] Python is a high-level programming language that was used on the project testing phase. It was used due to its ease integration with tools like Monkey Runner and allowed to create several python scripts that

are responsible to launch the Android emulator, on headless mode, with a specific set of parameters defined on a separated configuration file and navigate through the application in order to simulate the user actions and compare with a predefined set of images which correspond to the expected result.

3.4.7. Android SDK

[39] Android is an Operating System designed by Google for smartphones and more recently for tablet. Although, it is currently on the Ice Cream Sandwich (codename for the 4.0.x version) most devices still use the 2.3.3 (Gingerbread) as it is possible to see on the following figure.

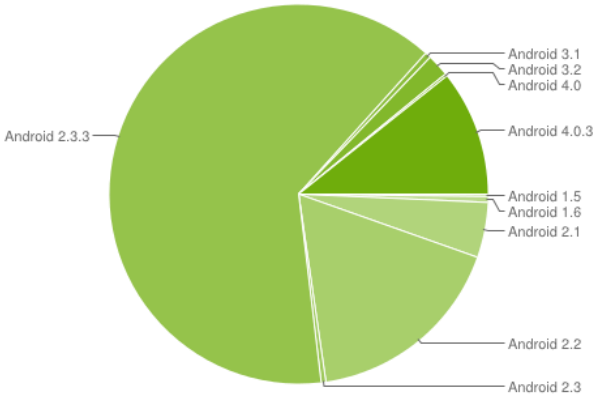


Figure 1 - Android: platform versions distribution

Note: The data presented on Figure 1figure 1 was measured by Google and corresponds to a 14-day period analyses, ending on July 2, 2012, of every Android device that accessed Google Play.

During the last Google I/O conference (end of June 2012) a new version was released – Android 4.1, codename Jelly Bean. Although, there is no official update for any of the supported devices it has been very well received by the Android community.

Other tools outside the Android SDK were used during the project testing phase and can be found described on the Software Quality chapter. They were omitted from the present chapter once they were used more as support tools which covered some limitations found during application testing.

This page was intentionally left in blank.

4. Architecture

With good program architecture debugging is a breeze, because bugs will be where they should be.
– David May, Professor of Computer Science at the University of Bristol.

The architecture is an essential part of the development of any project. It defines the structure, behaviour and views of the system providing a solid guideline towards its implementation.

This information must be detailed and represented on an architecture document where other developers can access and easily understand how the system works from different points of view. Only then it is possible to extend or upgrade a specific feature.

Note: The present section is an overview of the information available on the appendix Wise_Energy_-_Appendix_E_-_Architecture. This document contains a full description of both the Wise Energy Android application and the Charting API library architecture which follows the IEEE 1471 standard for "Recommended Practice for Architectural Description of Software-Intensive Systems" (IEEE-sdt-1471-2000 [40]). Moreover, on this document it is also presented the technical decisions made during the project development.

4.1. Architecture Overview

The present section is divided into three sub-sections:

- **Project integration**
  Provides an overview of the external services used in order to access the home's energy data.
- **Android Application Overview**
  Provides an initial overview of the project architecture.
- **Module Views**
  Describes the principal module views of the project – decomposition view, uses view and data model.
- **C&C Views**
  Corresponds to the Client-Server view and the Model-View-Controller used to define the application component-and-connector styles.

4.1.1. Wise Energy

Project Integration

Wise Energy is integrated with an external service responsible for measuring the home's energy data and adds full support to remotely control all the devices connected to a central hub – controller. Wise Energy accesses this data via a set of API calls made to an external server – figure 2.

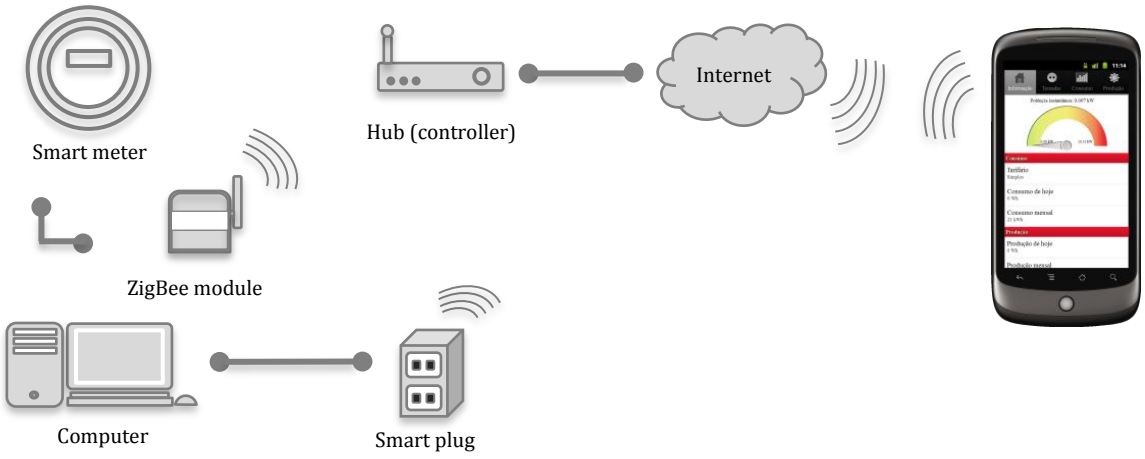


Figure 2 - Wise Energy: External services

Energy readings can be obtained from two different sources:

- Smart meter, which is responsible for measuring and recording the home's (global) energy consumption. It is connected to a ZigBee module responsible for periodically sending new data to the local controller. Newer versions support communication directly with the HAN's without the need of an additional interface.
- Smart plugs correspond to the hardware interface required to communicate with the home's devices and appliances. It measures and records the energy being used and supports remote control – turning on/off.

The communication between the smart plug, the smart meter (with the ZigBee module) and the controller is made via ZigBee – a standard wireless protocol designed for low cost and power networks.

The hub is connected to the HAN's and works like a gateway. It provides access to the energy efficiency system data and control to authenticated users through the Wise Energy application.

Android Application Overview

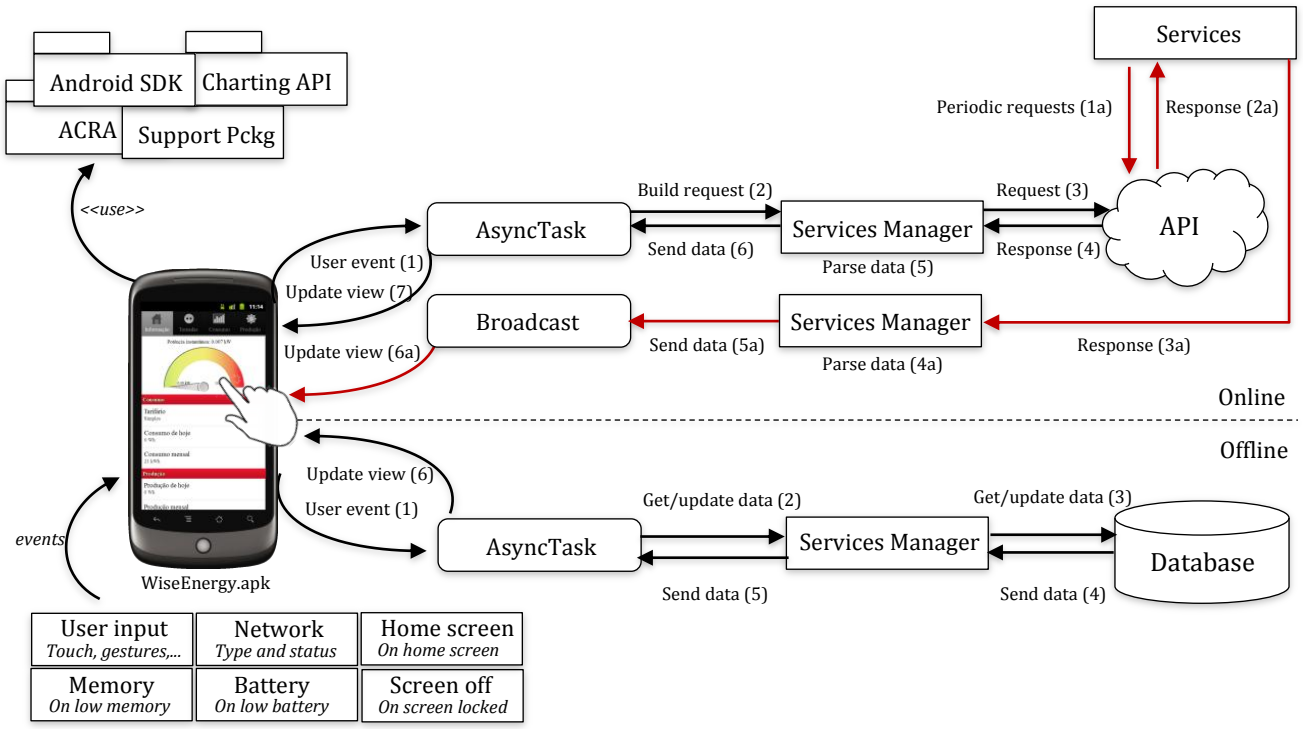


Figure 3 - High-level architecture of the Wise Energy application

The project uses a set of external libraries in order to provide additional features as it is the case of:

- The Charting API that was necessary to develop in order to accomplish the requirements defined for data visualizations on the Wise Energy project.
- The Android Support Package which offers a certain level of backward compatibility to older OS releases.
- ACRA a software quality tool responsible for logging the application errors and sending these reports to a defined email account.

Wise Energy was developed to use the lowest resources possible both from the device and the external server to which it communicates. The application adapts itself to the network type and quality: faster Wi-Fi connections deal with a higher number of requests per minute than slower and via 3G. The values attributed to each type are defined on an external configuration file (located on the *assets* directory) so it can be easily modified. Moreover, when the devices switch from a 3G network to Wi-Fi or the signal intensity increases or vice-versa the application is responsible for updating the requests rate. The same

logic is used when the device is running on low battery. The application also detects when the device screen is on/off and the keyboard is locked or not. During off status, on both situations, the background services responsible for continuous calls to the server are stopped avoiding unnecessary requests. Since the user cannot see the application, there is no need to update its content, this will only consume the device resources and increase the server load. The same principle applies when the running program is the Wise Energy Home Screen Widget.

One of the main features of using Wise Energy is to have access to the home's energy use and production in near-real time[5]. The system communicates with an external server via a set of API calls which can either be used to retrieve data from the server or to update the existing one (for example, defining a standby schedule for a specific module). To provide high availability when the device is not connected to the network, an offline module was created in order to store locally (database) the information about past energy consumption and production, power, the modules configurations, profiles, etc. Moreover, implementing a database to store the data from previous energy readings requests allowed to reduce the number of calls made and consequently the server load and device resources.

Based on the fact that tablets are gaining market share an additional built was made in order to support these devices. This new version uses the hardware acceleration functionality available only for releases superior to Android 3.0 which provides smoother animations and faster rendering (graphics).

To sum up there are currently two different builds of the Wise Energy application:

- Smartphone release, which supports Android from 1.6 to 2.3.3 and the new 4.0.x version.
- Tablets release, which supports and it is optimized for the Android 3.x.

Module Views

This section represents the module structures of the project. Each module is composed by a set of classes which were grouped according to their specific set of tasks (jobs), this leads to a more clear and well-organized code.

Decomposition View

The decomposition view represents how the code will be organized in terms of modules and sub modules and provides an initial overview of the entire system. Figure 4 corresponds to the application UML package diagram.

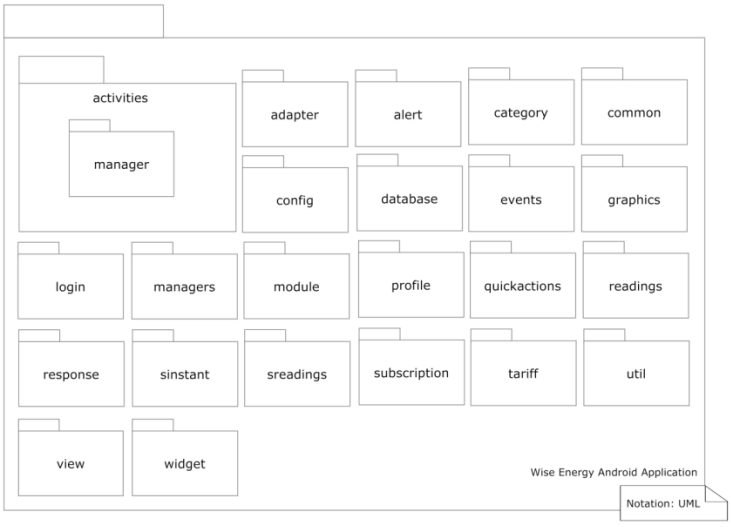


Figure 4 - Wise Energy: UML Package diagram

The application is grouped into several modules, each address one or several specific features. These are described on table 4 and represented in detail on figure 5.

[5] As mentioned before, the network type and the signal intensity can make the requests' rate time vary.

Module	Function
activities	It is responsible for launching the application through the *LoginActivity* where the user can enter her credentials in order to login. When she is validated it is created a *TabHostActivity* responsible for managing which activities will be shown in each tab: Information, Plug Management, Consumption Graphics, Production Graphics, Power Graphics, Cost Prediction Graphics, Profiles Management, Settings and Help. Most of these screens are built under a hierarchy tree, meaning that once the tab is clicked a main activity is created and from there, on user interaction, secondary activities (sub-activities) can be launched. On the application it is possible to find level 3 depth.
activities.manager	Since one of the requirements states that at all times if the device is on portrait mode the tab widget must be visible, it was necessary to redesign the native navigation between activities – activity flow. This is due to the lack of support from the tab components when dealing with sub-activities – only one activity is allowed per tab. The solution passed by using activity groups, for almost each tab, and singletons in order to save the activity state when navigating between different views. For this reason, most of the references defined on the Manifest file point to this location.
adapter	This module contains several adapters that are responsible to define the content and behaviour of certain objects. Additionally, it allows recycling views when a list element changes, contributing to a higher performance.
alert	Responsible for defining the XML structure, to implement the API for retrieve all the alerts from the server, and the new ones once the application is running and to parse the received response – XML file.
category	Responsible for defining the XML structure, to implement the API for retrieve the modules categories/subcategories and to parse the received response – XML file.
common	Defines the objects' structure and corresponds to the content data that will be used to launch/update the UI. It also contains an *AnimatedDrawable* class in which it is defined the behaviour of a loading animation that is used to inform the user that a request/response is being processed.
config	Provides access to the application configuration files and is responsible to save persistent data.
database	Provides access and all the necessary methods needed to interact with the local database. This content is private to the application and its main function is to work as a backup – allowing fetch/update data, when there is no Internet connection available.
events	Responsible for defining the XML structure, to implement the API for retrieve the scheduling events associated to a module and to parse the received response – XML file.
graphics	Work as a manager between the application and Charting API library developed. Depending on the type of graph requested, this modules parses the energy readings received, defines the graph structure (select which components are going to be shown – labels, captions, etc. and which actions are supported for the user to interact with – gestures, real time, etc.) and style (series color, minimum and maximum size, etc.) and returns the generated graph: • Bar chart (consumption global/per module and production). • Stacked bar chart (consumption global/per module TOU rates). • Line chart (measured and contracted power). • Multi bar chart (cost prevision).

login	Responsible for defining the XML structure, to implement the API call that is used to validate the user's credentials and to parse the received response – XML file.
managers	Contains the *ServiceManager* which provides the interface to all data requests. It is called by the activity when a request needs to be made and returns its content in order to update the UI. It ensures that when there is no Internet connection available, the data is fetched/ updated from the local database. Additionally, it is responsible to create and bind the application with the running services when the user is authenticated.
module	Responsible for defining the XML structure, to implement the API call for retrieving the modules data and to parse the received response – XML file.
profile	Responsible for defining the XML structure, to implement the API to retrieve the user profiles and to parse the received response – XML file.
quickactions	An external module developed by a user named Lorensius [8] that uses Android Quick Contact Badge and extends it to support user defined interactions.
readings	Responsible for defining the XML structure, to implement the API call for retrieving past energy readings and to parse the received response – XML file.
response	Defines the XML status response structure which corresponds to its status code, description, etc. That is used to confirm if the request was successful.
sinstant	Service responsible for requesting new data periodically and to notify the evolving process that needs to update their UI content.
sreadings	Service responsible for requesting new data periodically and to notify the evolving process that needs to update their UI content. Contrasts with the previous service in the type of API calls made and the idle time between requests.
subscription	Responsible for defining the XML structure, to implement the API call for retrieving the user's subscriptions and to parse the received response – XML file.
tariff	Responsible for defining the XML structure, to implement the API call for retrieving the energy tariff and to parse the received response – XML file.
util	Set of methods that are used throughout the code.
view	Contains some Android native methods that were necessary to overwrite in order to comply with the requirements or to provide a better user experience. One of these examples is the *VerticallScrollView* class which was necessary to redesign in order to support multiple gestures: swipe (left and right) in order to visualize different graphs and vertical scroll (up and down) to access the energy consumption detailed data on smaller devices (low and medium-resolution devices).
widget	Holds the widget content and service classes.

Table 4 - Wise Energy: packages description

This page was intentionally left in blank.

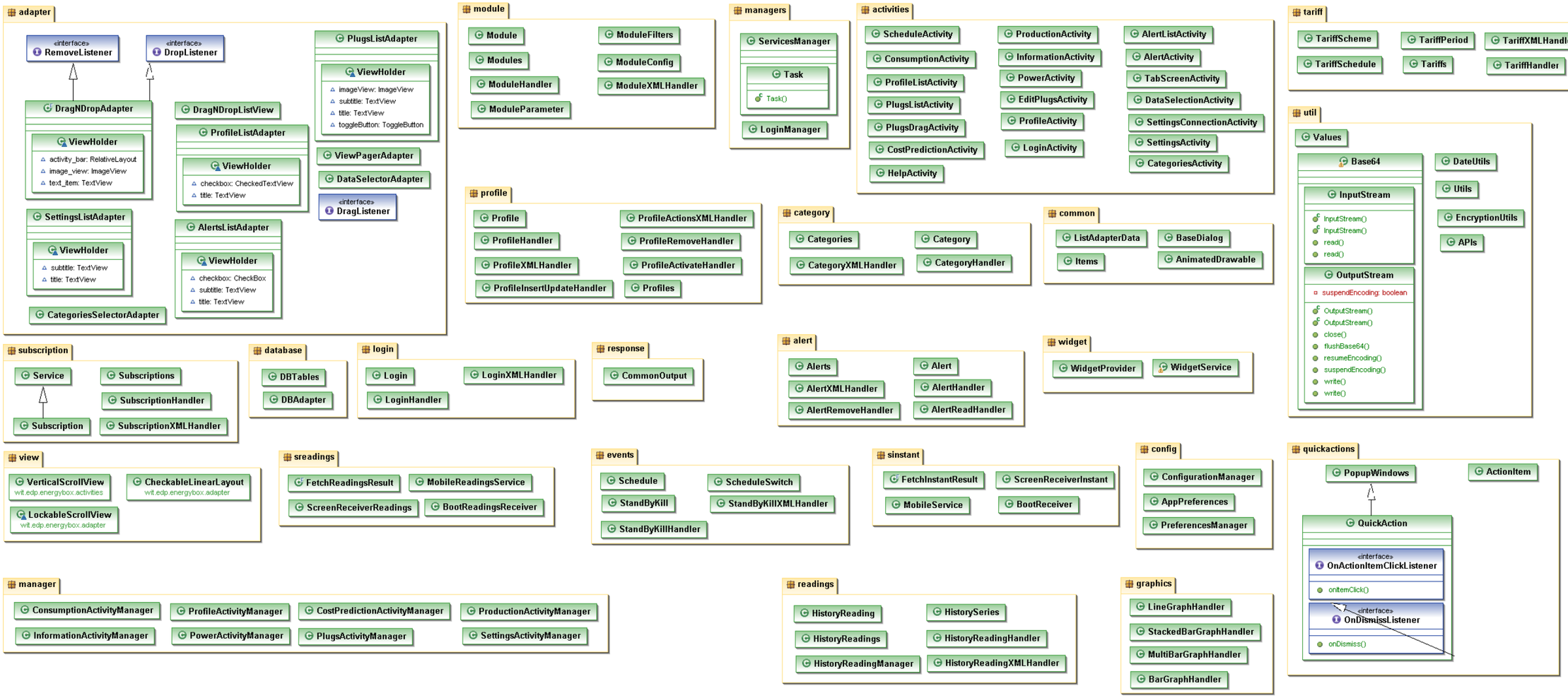


Figure 5 - Wise Energy: detailed package diagram

This page was intentionally left in blank.

Uses View

The uses view is an increment of the decomposition view with a *depends-on* relation defined between different packages. This connection is established between two packages when one depends on the content of the other.

This style has been proved particular useful during different phases of the project development:

- To understand the effect that a particular change can create on the system.
- To analyse which modules are required for a specific part of the system to work.
- To debug the application.
- To create testing cases.
- To support continuous integration of different modules into the application without changing its state.

In order to understand better the uses view represented on figure 6 some modules and their dependencies were filled with colour:

- The packages represented in yellow correspond to the code necessary for requesting, handling and parsing a specific API call. The *response* package corresponds to the response header object that is necessary to be analysed in order to confirm the status request of a specific request made to the server.
- The modules represented in pink correspond to the system services. These run continuously if the application is active and are responsible for making periodically requests to the external server in order for views to be updated later by the controllers. Additionally, they support several events that can affect their behaviour – for example, if the screen is off the services are automatically stopped in order to avoid wasting the system resources and ultimately to reduce the server load.
- The *widget* module is represented in grey since it works almost independently from the main application – it does not rely on its services to fetch data. Instead, in order to avoid wasting resources (widgets consume a lot) it was designed to work only the strictly necessary. Only when the user is at the home screen and the screen is on and unlocked, the service schedules its own services to request data.
- The modules represented in green correspond to the application controllers – *activities* and *activity managers*. They are the ones responsible for updating the views content.
- Without any background there are the *configuration*, *managers*, *util*, *common*, *adapter*, *quick actions*, *graphics* and *view* modules that are used across the entire application. This last one does not have any dependency since it is called directly from the XML layout resource files. It defines the behaviour that a particular view component should have and usually corresponds to Android classes that were necessary to redesign.

This page was intentionally left in blank.

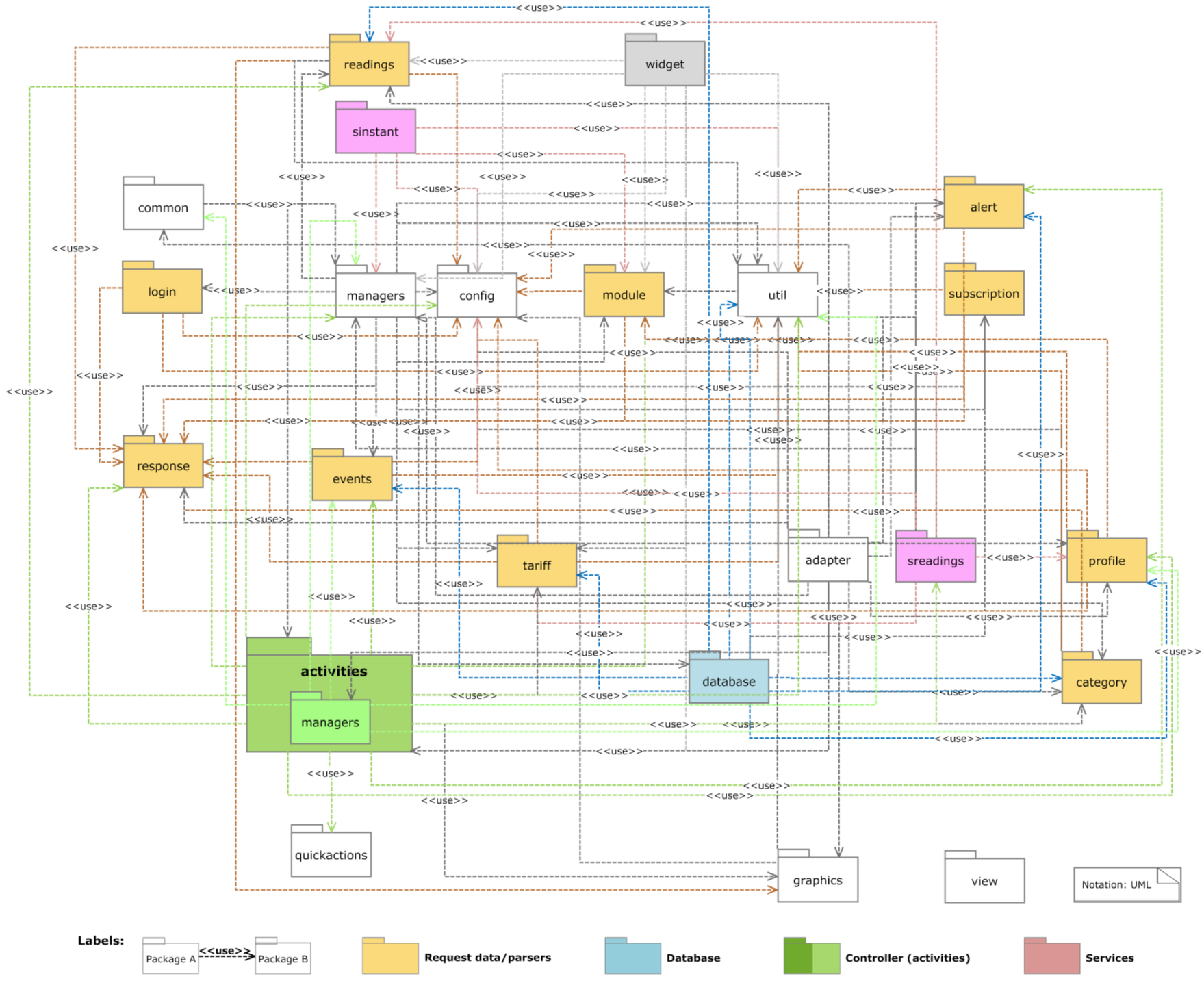


Figure 6 - Wise Energy: UML Package diagram (dependencies)

Data Model

The data model corresponds to the application database view. The database scheme is available on the entity-relationship diagram on figure 7.

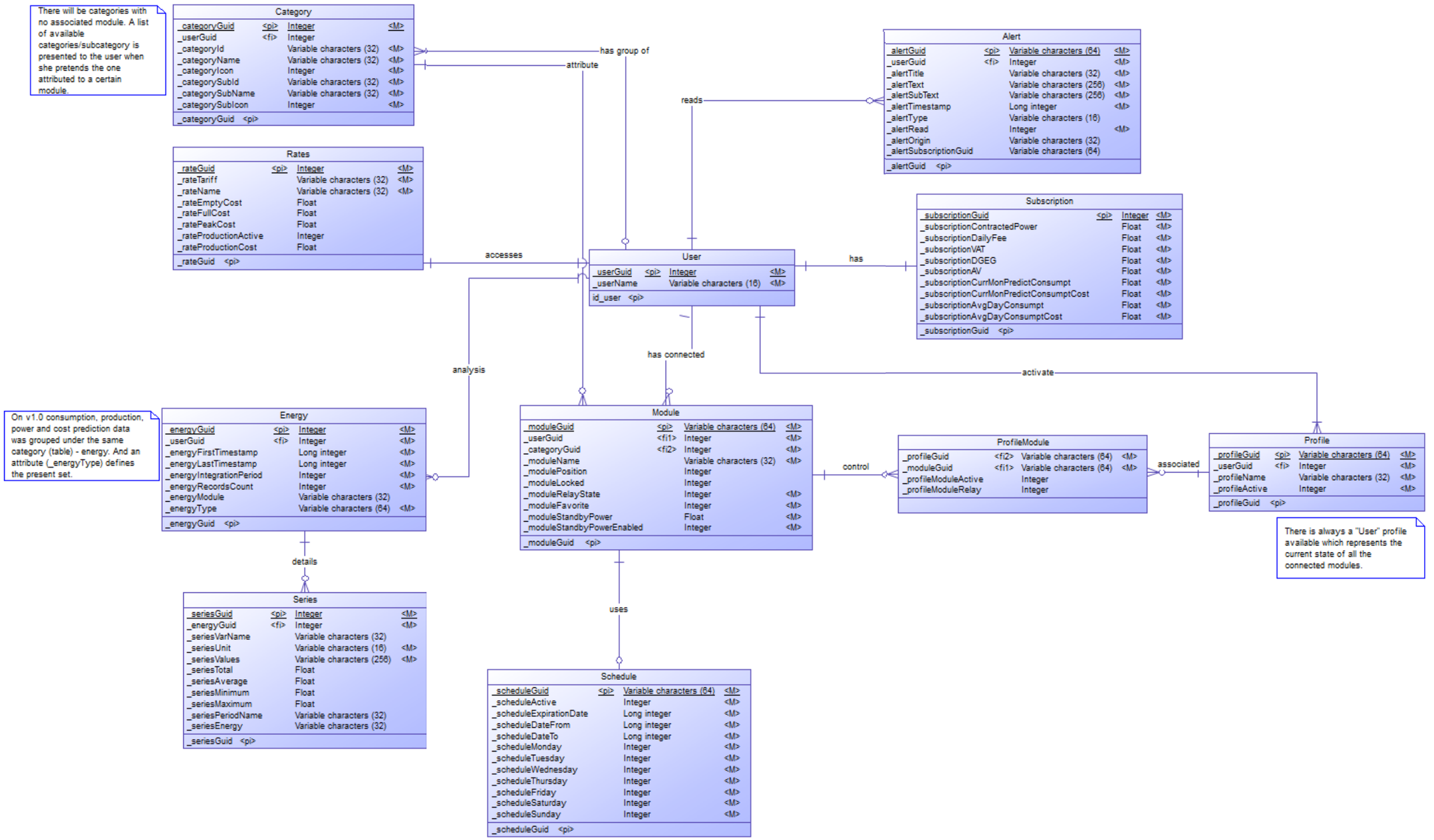


Figure 7 - Wise Energy: Entity-Relationship diagram

Table	Function
Alert	This table stores the information about an alert. The identifier used (primary key) has the same value as the global identifier which is sent from the server along with the data request. This way it is easier (and faster) to search for a specific alert. There are three different columns which contains the alert information: title, text and subtext (which contains more detailed information than the available on text column). In order to maintain consistency across the application this structure follows the same as the one received on the XML response for a given alert. Read corresponds to the current alert state. Timestamp is the time in seconds from when the alert was generated. The type corresponds to its classification which can be one of the following: • Information, which indicates for example the end of an active schedule. • Warning, warns the user that for example a specific module is using too much energy. • Alert, notifies the user that for example the home's energy consumption is over the defined threshold. Origin corresponds to the device responsible for sending this alert.
Category	Corresponds to the category types available for each one of the modules. This table is connects with *Modules* table once each one of the connected modules has an associated category defined by the id from the columns category and subcategory (information sent from the server) and it is matched locally to one of the correspondent name and icon defined on the configuration files. If one of this id's does not have a reference, it is then attributed the name and icon "other". Moreover, it connects with *User* since it is necessary to display a list of available categories in a view in order to modify the one already attributed to a specific module. It is also worth to mention that the icon type is defined as an integer since it represents a specific reference to its location (via *R.java* file) which is maintained across the entire application and the number of runs. This way there's no delay time associated with this process. Since every category can have more than one subcategory, the defined id for this table is generated automatically by the database system via Auto increment. Additionally, a view using the category and subcategory id was created in order to facilitate the data access.

Energy	This table was redesigned in order to store all the energy-related data making it easier to access and handle inside the application. It contains data from: • Global energy consumption. • Energy consumption per plug • Energy production • Power • Cost prevision. This table is connected to *Series* which contains more detailed information about energy. This separation was created since it is possible to have *Energy* data without *Series* being available – which often occurs when the user selects a time period when there is no information registered. A view using the first timestamp, last timestamp, integration period and type was created in order to facilitate the data access. The integration period used differs depending on the device orientation – portrait/landscape.
Module	The module is identified by a global id that is sent from the server when new data is being retrieved. This table connects with: • User, since modules are associated to a certain account. • Schedule, which corresponds to the standby time period. If this is active and a specific module is consuming energy which is under the defined threshold value the system automatically turns it off during the defined period. • Category corresponds to the module category and subcategory. For example: living room lamp, room computer, etc. The user has the chance to organize the modules order by either separating them into two different categories: favourites and others, or to manually drag them in into a user-specific order. The "relay status", the "locked" and the "standby power enabled" columns correspond to the current module status (if it is on or off), if it is possible to change it and if the standby schedule is active or not, respectively.
Profile	A profile defines a specific configuration of a set of modules – it is possible to add, remove and change a plug status. When a profile is active the system automatically changes its state to the one defined. Therefore, the *Module* table is connected to *User* (it is account dependent) and to Module via a weak entity. This one is responsible to store the module state that the user defined and if it should be updated or not. **Note:** the "User profile" which corresponds to the system default profile with all the connected modules added and its current state it is not added to the database.
ProfileModule	It is a weak entity and represents the modules added to a specific profile. It defines their relay status and if it should be changed or not. It is responsible to connect the *Module* and *Profile* table.
Rates	*Rates* has a single instance which contains the client energy tariff, the utility rates and if the home's is enabled to produced energy or not. It is only updated (or created on a first run) when the user is authenticated.

Schedule	Defines a time period for when then standby will be active (or not): • Initial and final date (in seconds). • Expiration date. • Status • Week days in which the schedule is going to be active.
Series	This table stores the energy readings data. A single instance on energy can reference at most three different series. This is due to the division made on TOU rates where each tier corresponds to a different serie.
Subscription	This table contains a single entry which corresponds to: • The fees applied over the energy bill. • The average day consumption (in the system currency/energy). • The months' predicted cost and energy. • The home's contracted power. This data is updated every day.
User	This table works as interface between the credentials entered on the application and the available data on the local database. The user credentials are stored on a *SharedPreferences* file (a more detailed information is available on the Data Storage section). When the user is authenticated the application manager checks if there is any entry on the database which corresponds to the username entered – on the *User* table. If it already exists it loads all the available data to the application, otherwise it creates a new entry for storage. This table was also created separately in order to provide modularity to the application.

In order to facilitate the data access it was created the following views, additionally:

- ViewProfileModule, that retrieves the modules data associated with a specific profile.
- ViewCategories, that selects the all the subcategories that are related with a specific category.

And the following tables, which deal with a large quantity of data accesses, were indexed in order to improve its speed:

- Alert
- Module
- Profile

Note 1: *sqlite* does not have a separate Boolean storage class. Instead these values are stored as integers: 0 for False and 0 for True.

Note 2: there is an additional table which represents the device location – *android_metadata*, which is automatically created by the system and since it has no relevance on the current project was discarded from figure 7.

C&C Views

The component-and-connector style specifies the connections available between the application components and the communication protocol used.

Client-Server View

The client-server view provides a better insight into the application's producers and consumers and their interaction, in order to retrieve new data.

Wise Energy communicates with an external server via a set of API calls. For each one it is necessary to create a specific XML object that will be sent via an HTTP request to the external server. These can be made either by the running services, responsible for keeping the application data updated, or via the *ServicesManager* class which is responsible for making the initial requests for when the application is loading. For example: to retrieve the account subscriptions (since these are one time requests it is not necessary to add them to the services), or to an user event that triggers a server update or synchronization.

When a round of requests is made the Services notify the *ServicesManager* class that there is new data available that need to be processed and sends to the correspondent views. Since these two components run on different process (each one has its own address space) this means that there is no direct access to the each other's memory space. Therefore, in order to establish a communication between both it is necessary to implement an AIDL interface (which corresponds to a lightweight implementation of the IPC protocol).

On startup (after authentication) the *ServicesManager* is automatically connected (binded) to the background services and kept listening for incoming data retrieved from the server. When the responses are received the services notify the listeners that there is new data available. Once it is retrieved the manager processes it (parses and checks for errors) and notifies the current running activity that there are fields that require update. Moreover, this data is added to a local database in order to become available to the entire application (subsequent activities).

Figure 8 represents the client-server class diagram.

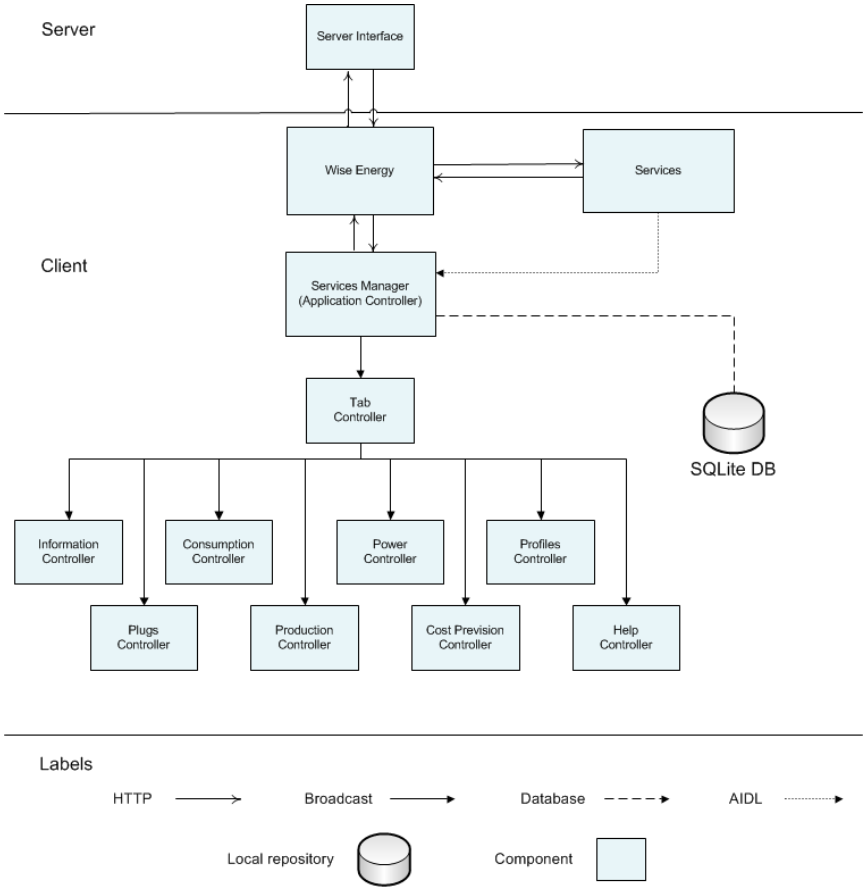


Figure 8 - Wise Energy: Client-server diagram

Model-View-Controller

The application MVC architecture is presented on figure 9 and 10. They describe all the steps involved when a specific action is triggered by the user when she is interacting with the application. The scenario chosen is the selection of a different time period for a better analysis on the home's past energy use.

In the scenario described on figure 9Figure 9, users access the application by interacting with the views – which correspond to every UI component available on the window – layout, widgets, fields, drawables, etc. When for example a toggle button is pressed the system generates an *onClick()* event which is captured by the current running activity – controller. The component state is evaluated and the correspondent action is taken which usually involves making a call to the external server or local database (in case the device is not connected to the Internet). This is carried out by the application handlers like the *EnergyReadingsHandler*, or *ModulesHandler* (models) – they are also responsible for receiving the response and sending it to the caller. When this data becomes available the views are notified and its content is updated.

Although, on the previous scenario it involved a user action resulting in a view update, there are other alternatives to update its content. There is a set of running services that are periodically requesting new data to the external server. When a batch of responses is received the connected binder (*ServicesManager* class) is notified and retrieves this information in order to parse and notify the current view that there is new data available (a more detailed description is provided in the Wise Energy request section).

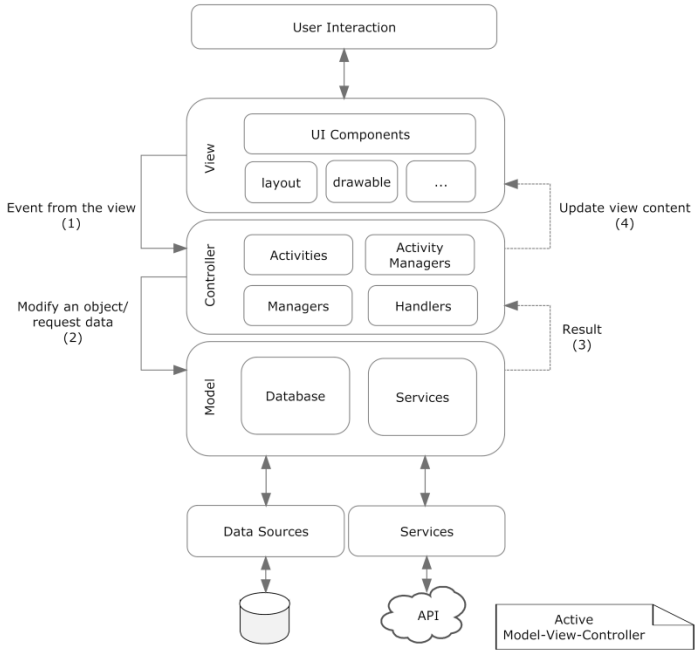


Figure 9 - Wise Energy: Active MVC diagram

On figure 10 the user decides to choose a different time period. This action corresponds to the flowchart initial state and it is done by entering two different dates. When the last date is entered the controller – in this case the *DataSelectionActivity*, receives a notification event (*onClick()*) and attempts to validate the input data. If both periods are correct the activity launches an *Asynctask* which instantiate the *ServicesManager* class that will be responsible to pass the correct parameters to the classes (methods) that will create an energy readings request (model). When the XML object is created it is sent via HTTP (an HTTPS mode is also supported) to the external server, and a listener will wait for its response. Upon reception the incoming data is read and analysed in order to check if there is any server-side or transmission errors; if there is none, the header status is retrieved and compared with the expected success code. If both values match the message body is parsed and stored on an energy readings object. Additionally, during this process the content is also analysed to check for missing information; if it passes, the view is notified that there is new data available and therefore it is necessary to update its fields.

If during this last step (the status code does not match), the view is directly notified by the controller that an error had occurred, and a *Dialog* with this information is displayed on the screen. Moreover, if the error corresponds to an invalid response or parameter the response, a new request is made to the server in order to retrieve the information that is missing. It is worth to mention that this process has a limited number of retries after which is displayed the above *Dialog*.

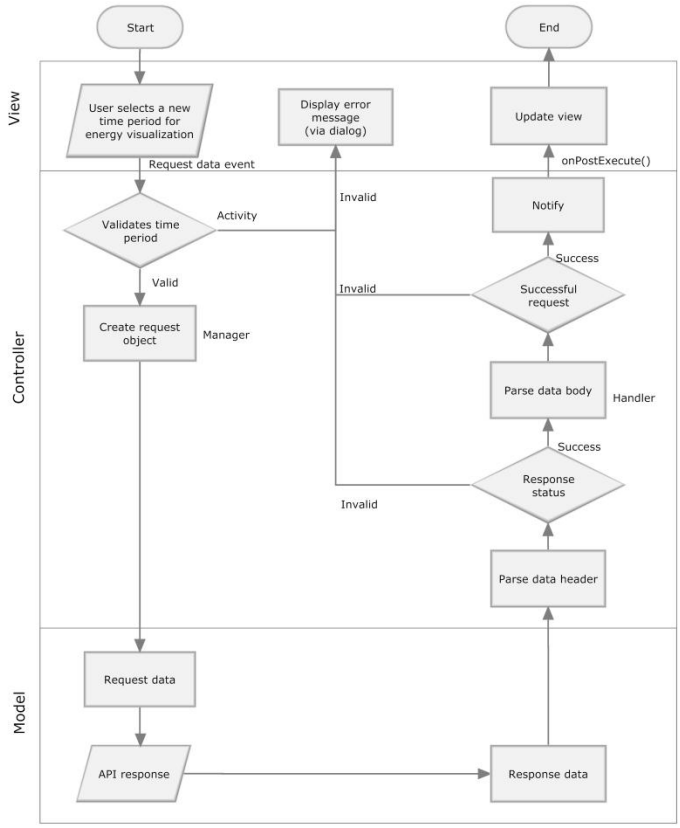


Figure 10 - Wise Energy: Active MVC flowchart

4.1.2. Charting API

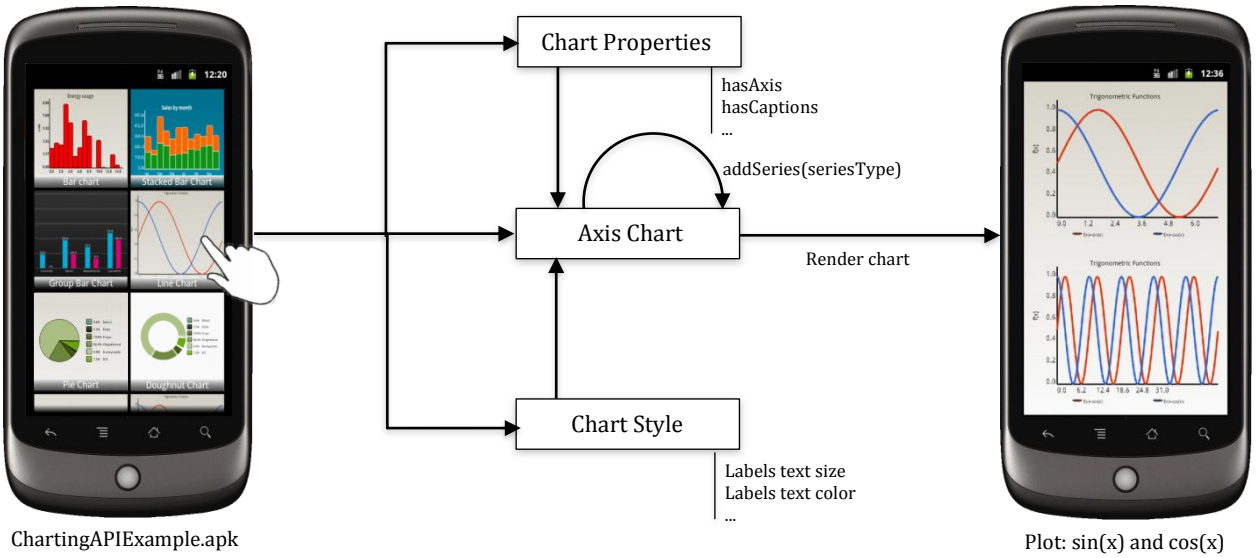


Figure 11 - High-level architecture of the Charting API Library

Note: the devices screen represented on Figure 11 correspond to screenshots from the ChartingAPIExample. A sample project created to demonstrate the potential of the developed external library – in this case the plot of the trigonometric functions: sin(x) and cos(x).

The Charting API is an Android library initially developed for the Wise Energy application in order to add graphical support to data visualization. At this moment, it is independent from the initial project and is being used on other Android applications developed by WIT Software.

In order to keep consistency along the library the drawing process is the same for all the available charts:

- **Define the chart properties.**
  This corresponds to the selection of which components will be draw. For example, the graph displayed on Figure 11 has captions (series name), title on top, the yy axis values are aligned to the right, etc.
- **Define the graph style**
  After defining the chart properties it is necessary to define their attributes. For example, the colour and stroke width used to draw the axis, the title size, etc.
- **Add one or more series**
  The developer can add a multiplicity of series, even from different types. On Figure 11 the chart contains two different line series – one representing the sin(x) function and the other the cos(x).

It is worth to mention, that if none of the first two steps is defined the library will use the default values defined to draw the graph.

Module views

Each module is composed by a set of classes which were grouped according to their specific set of tasks (jobs), this leads to a more clear and well-organized code.

Decomposition View

The decomposition view is usually the first style to be designed. It represents how the code will be organized in terms of modules and sub modules and provides an initial overview of the entire system. Figure 12 corresponds to UML package diagram designed for the Charting API library.

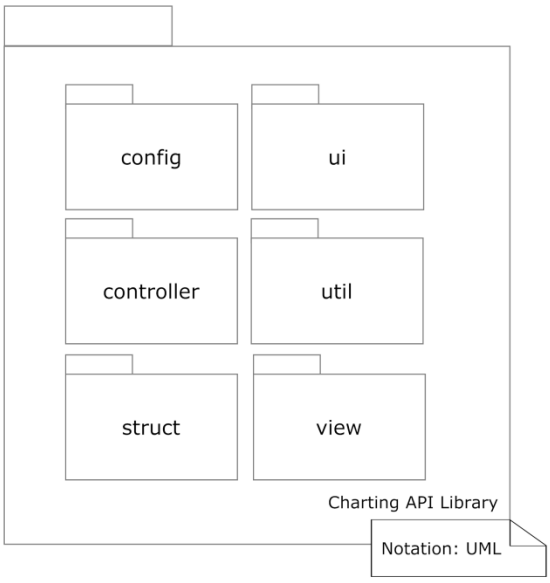


Figure 12 - Charting API: UML Package diagram

Module	Function
config	Stores de graph default values. It works as an interface between the classes and the attributes defined on the xml resources files (*colors.xml* – for the graph colors, *dimens.xml* – for its dimensions and *strings.xml* for generic text). This is particularly needed for some of the project objects since android only allows to access these values from classes who extend *android.content.Context* adding an unnecessary layer of complexity.
controller	Stores the controller written to handle user input events. Supports different types of gestures – single, multiple, pinch, drag, etc. It was developed to support Android releases as back as the 1.6 which only detects single touch events. Additionally, it is also capable to detect the position of different fingers on the screen and associate a continuous event (for example a drag) to a specific initial point, to measure the pressure on the screen (if it is supported by the device), to identify different gestures, etc.
struct	Contains the graphs properties, style and structure.
ui	Set of classes developed to provide an easier access to paint and text paint components as well as some additional methods which provide additional support to measure text width and height (typeface dependent). Holds the style and color information about how to draw geometries, text and bitmaps.
util	Set of methods that are used throughout the code.
view	Contains the different charts classes. Each class implements its own method which allows drawing the defined chart. Current chart types available: • Bar • Stacked bar • Grouped bar • Line • Doughnut • Pie

Table 5 - Charting API: packages description

Uses View

The uses view is an increment of the decomposition view with a *depends-on* relation defined between different packages. This connection is established between two packages when one depends on the content of the other.

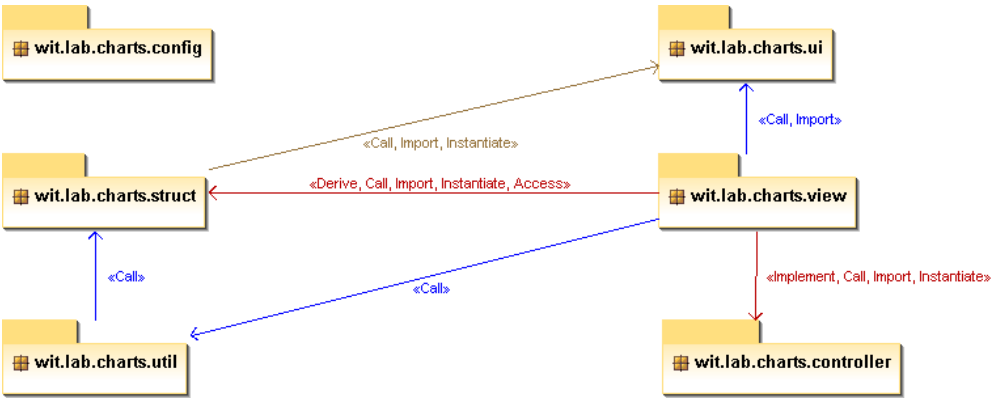


Figure 13 - Charting API: UML Package diagram (dependencies)

Aspects Style

Note: For a complete analysis of the project's class diagram (figure 14) and more detailed information about the classes written, their methods and connections, this information is available on the Wise Energy generated Javadoc files.

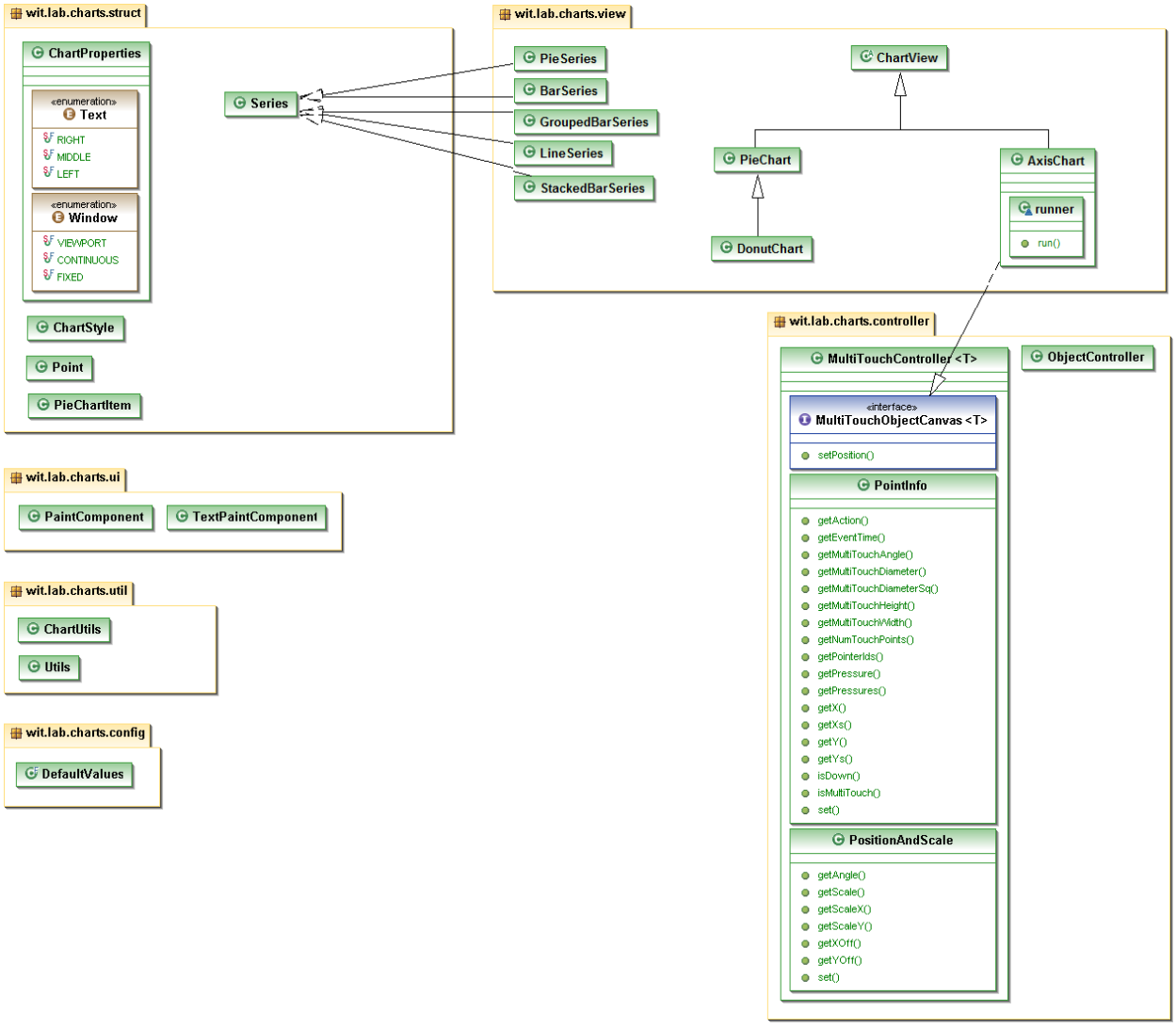


Figure 14 - Charting API: Class diagram

4.2. Software Development Metrics

At the end of the internship the following products were developed:

- Wise Energy Android application.
- Wise Energy Android application (optimized for tablets).
- Charting API library.
- Charting API sample project.

Moreover, a set of tests were written for the Wise Energy project in order to assure the project quality.

The results presented on this section – tables 6 to 12 correspond to the analysis made by EMMA [42] – a Java code coverage tool, during the final testing phase and the run of a small python script made to count the number of files available on a specific folder and the number of lines of code each file has. It is worth to mention that the script needed to be written since EMMA is only able to analyse the project source code. Moreover, it was created to ignore the lines that were composed by spaces, tabs, the new line character and single brackets.

The Wise Energy project contains files with different formats and syntax that are used across the application and define the communication between services, the resources used, etc. This corresponds to the AIDL, assets and resources folder.

AIDL stands for Android Interface Definition Language and it allows defining the programming interface that the *ServicesManager application* class and the running services (*SInstant* and *SReadings*) agreed upon in order to communicate with each other using InterProcess Communication (IPC).

The assets correspond to the configuration properties. This file defines all the system settings like the API url, request period, currency and energy units, etc. and since it does not rely on the application source code it makes the process of changing one of the project properties extremely easy.

The resources folder is then divided into a set of sub-folders that describe:

- **Animation**
  Corresponds to a set of pre-determined operations which will later be used to animate the application view components.
- **Drawable**
  Holds a set of bitmap files used to create the application views. Additionally there is also a group of xml files that define the background of several elements. Components are grouped into *ldpi* (low), *mdpi* (medium), *hdpi* (large) and *xhdpi* (extra-large) which are used for tablets.
- **Layout**
  Defines the application views. In order to support different device orientations this folder is divided into portrait and landscape mode, each own with its own layout files.
- **Menu**
  Defines the content of the “options” menu.
- **Values**
  Contains the *string*, *styles*, *dimens*, *colors* and *theme* resources. It is worth to mention that there two different strings.xml files one defines the content in Portuguese and the other in English (international).
- **XML**
  Contains the widget attributes: update time, layout, dimensions, etc.

Project: WiseEnergy	
Total packages:	24
Total executable files:	131
Total classes:	288
Total methods:	2348
Total executable lines:	11932
Total aidl files:	6
Total aidl lines:	28
Total asset files:	2
Total asset lines:	73
Total resources files:	89
Total resources lines:	3576
Total bitmaps files:	239

Table 6 - Software development metrics: Wise Energy (application)

Table 7 represents a version built optimized for tablets. Since these devices have a larger screen and use a more recent release of Android, it is necessary to update the existing layout to face these new dimensions and take advantage of the new functionalities available on Honeycomb (version 3.x). Nevertheless, a large part of the application code can be reused. Table 7 represents the number of executable, AIDL, asset and resources lines added to the initial project *versus* the ones that were possible to reuse.

Project: WiseEnergy Tablet	Code reuse	New code
Total executable lines:	10927	4904
Total aidl lines:	28	0
Total asset lines:	73	0
Total resources lines:	1925	1418

Table 7 - Software development metrics: Wise Energy (tablet application – code reuse)

Project: WiseEnergy	Smartphone	Tablet
Total packages:	24	24
Total executable files:	131	131
Total classes:	288	293
Total methods:	2348	2374
Total resources files:	73	97
Total aidl files:	6	6
Total asset files:	2	0
Total resources files:	61	28
Total bitmaps files:	239	309

Table 8 - Software development metrics: Wise Energy (tablet application – files)

Project: WiseEnergy Test	
Total packages:	3
Total executable files:	11
Total classes:	37
Total methods:	431
Total executable lines:	289
Total resources files:	0
Total resources lines:	0

Table 9 - Software development metrics: Wise Energy (test)

Project: WiseEnergyUITest	
Total packages:	1
Total executable files:	4
Total classes:	4
Total methods:	7
Total executable lines:	119

Table 10 - Software development metrics: Wise Energy (UI test)

Project: ChartingAPI	
Total packages:	6
Total executable files:	19
Total classes:	25
Total methods:	361
Total executable lines:	1769
Total resources files:	170
Total resources lines:	7
Total bitmaps files:	9

Table 11 - Software development metrics: Charting API (library)

Project: ChartingAPIExample	
Total packages:	7
Total executable files:	33
Total classes:	57
Total methods:	480
Total executable lines:	2748
Total resources files:	40
Total resources lines:	1276
Total bitmaps files:	16

Table 12 - Software development metrics: Charting API (application)

5. Project Planning

Hofstadter's Law: It always takes longer than you expect, even when you take into account Hofstadter's Law.
– Douglas Hofstadter, Gödel, Escher, Bach: An Eternal Golden Braid.

5.1. Software Development Methodology

The first step towards a successful software development starts with the methodology adopted for the entire process.

It is a fact that the software industry is always evolving. In order to keep up it is necessary to adopt an *agile* methodology for software development that provides a more flexible environment in which the team is not restricted to follow a top-down approach, but on the contrary every time it is justifiable it is possible to upgrade the previous steps. This is particularly important since it is not uncommon that new issues arise during the implementation phase. Furthermore, since the deadlines are not as demanding as in other methodologies (e. g. on *The Waterfall* model) the team has a higher chance to contribute with new ideas and solutions towards the project.

The methodology followed it is not included into any of the most common agile methods. Instead it uses several features of different practices:

- **Product Backlog:**
  A prioritized set of features to be implemented based on the requirements.
- **Meetings:**
  Every time a problem appeared or a specific requirement needed to be analyzed, it was arranged a short meeting. These usually take only a few minutes and allow discussing the problem found before the end of the sprint providing a faster resolution.

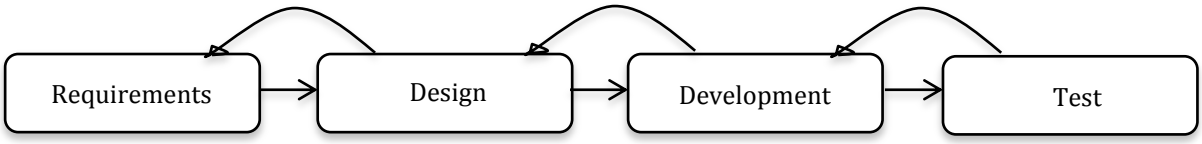


Figure 15 - Overview of the different phases of software development

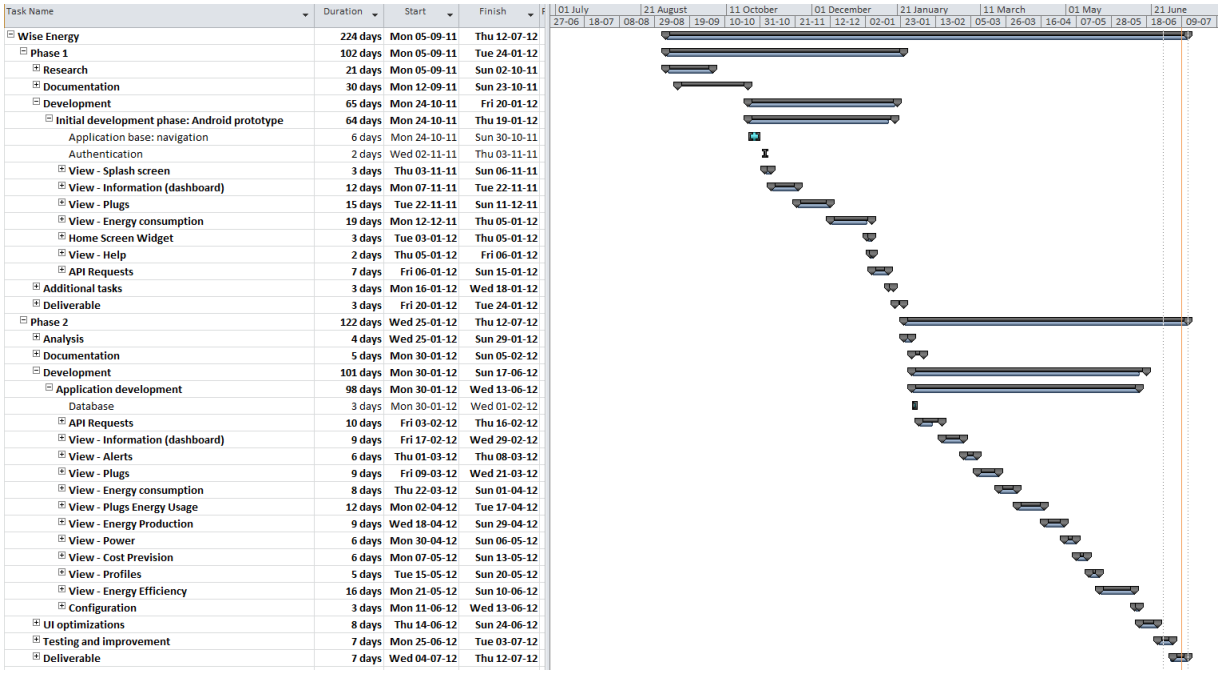
All the documentation and code produced are under version control, on WIT Software's repository, according to the companies' policy, and are updated daily.

5.2. Plan

After the state of the art and the requirements were completed, it was defined the initial plan. The planned spanned over 10 months and reflected the development of an energy efficiency application solution for the Android platform – Wise Energy.

Once the current project corresponds to the internship work the plan was later divided into two separated parts reflecting the FCTUC deadlines. This corresponds to the academic first and second semester.

Figure 16 represents the initial defined plan. A Gantt diagram was also designed in order to provide a better time overview over the entire project.



Task Name	Duration	Start	Finish
Wise Energy	224 days	Mon 05-09-11	Thu 12-07-12
Phase 1	102 days	Mon 05-09-11	Tue 24-01-12
Research	21 days	Mon 05-09-11	Sun 02-10-11
Documentation	30 days	Mon 12-09-11	Sun 23-10-11
Development	65 days	Mon 24-10-11	Fri 20-01-12
Initial development phase: Android prototype	64 days	Mon 24-10-11	Thu 19-01-12
Application base: navigation	6 days	Mon 24-10-11	Sun 30-10-11
Authentication	2 days	Wed 02-11-11	Thu 03-11-11
View - Splash screen	3 days	Thu 03-11-11	Sun 06-11-11
View - Information (dashboard)	12 days	Mon 07-11-11	Tue 22-11-11
View - Plugs	15 days	Tue 22-11-11	Sun 11-12-11
View - Energy consumption	19 days	Mon 12-12-11	Thu 05-01-12
Home Screen Widget	3 days	Tue 03-01-12	Thu 05-01-12
View - Help	2 days	Thu 05-01-12	Fri 06-01-12
API Requests	7 days	Fri 06-01-12	Sun 15-01-12
Additional tasks	3 days	Mon 16-01-12	Wed 18-01-12
Deliverable	3 days	Fri 20-01-12	Tue 24-01-12
Phase 2	122 days	Wed 25-01-12	Thu 12-07-12
Analysis	4 days	Wed 25-01-12	Sun 29-01-12
Documentation	5 days	Mon 30-01-12	Sun 05-02-12
Development	101 days	Mon 30-01-12	Sun 17-06-12
Application development	98 days	Mon 30-01-12	Wed 13-06-12
Database	3 days	Mon 30-01-12	Wed 01-02-12
API Requests	10 days	Fri 03-02-12	Thu 16-02-12
View - Information (dashboard)	9 days	Fri 17-02-12	Wed 29-02-12
View - Alerts	6 days	Thu 01-03-12	Thu 08-03-12
View - Plugs	9 days	Fri 09-03-12	Wed 21-03-12
View - Energy consumption	8 days	Thu 22-03-12	Sun 01-04-12
View - Plugs Energy Usage	12 days	Mon 02-04-12	Tue 17-04-12
View - Energy Production	9 days	Wed 18-04-12	Sun 29-04-12
View - Power	6 days	Mon 30-04-12	Sun 06-05-12
View - Cost Prevision	6 days	Mon 07-05-12	Sun 13-05-12
View - Profiles	5 days	Tue 15-05-12	Sun 20-05-12
View - Energy Efficiency	16 days	Mon 21-05-12	Sun 10-06-12
Configuration	3 days	Mon 11-06-12	Wed 13-06-12
UI optimizations	8 days	Thu 14-06-12	Sun 24-06-12
Testing and improvement	7 days	Mon 25-06-12	Tue 03-07-12
Deliverable	7 days	Wed 04-07-12	Thu 12-07-12

Figure 16 - Initial plan and Gantt diagram

5.2.1. First Semester

The first weeks of the semester were focused on providing a new insight into the "smart energy" field, answering questions like why and what makes is it smart, and analyzing the value it can bring towards the client and utilities; and to research for energy saving solutions and automation products available. Since this area had been receiving a lot of attention and many companies continuously releasing new products, for several occasions during the internship it was necessary to update the initial document of the state of the art.

During the requirements phase, the user stories defined on several meetings with the product owner where analyzed and mapped into the application requirements and grouped according to its specification – functional, quality and design requirements. After being defined, they were converted into mockups and submitted to the project manager. After his analysis, the wireframes were built along with the supported user interaction gestures (click, touch, drag, pinch, etc.) as well as the final aesthetics and sent to the PM for revision. Once approved, they were submitted to the product owner for further acceptance.

It is also worth to mention that during the requirements elicitation and analysis it was fundamental to conduct a background research about several subjects, namely energy and power and how this two correlate.

This information is available on the annex: WiseEnergy_-_Appendix_D_-_Requirements_Specification.

After defining all of the above, the new goal was to create a working prototype until the end of the first semester containing the following features:

- Design the application layout for the login, dashboard, plugs and consumption view.
- Display the current and past energy usage and production.
- Access and edit all the plugs connected to the gateway.
- Communicate with an external API in order to retrieve the home's energy consumption data.
- Support an offline mode for the users to access the application even when there is no Internet connection available.
- Develop a prototype of a widget that would display the daily energy use and production.

All of the above were achieved, on schedule.

5.2.2. Second Semester

For the second semester it was initially defined to follow the work done previously and implement other functionalities in order to get the final Android application:

- Design the following application views and controllers:
  - Alerts list and content.
  - Categories.
  - Plugs management and details.
  - Data selection.
  - Energy production.
  - Energy efficiency.
  - Contracted and maximum power measured.
  - Cost prevision.
  - Settings.
  - Help.
- Revise and update the dashboard layout.
- Fully develop the Home Screen Widget.
- Optimization across different devices: smartphones/tablets.
- Testing.

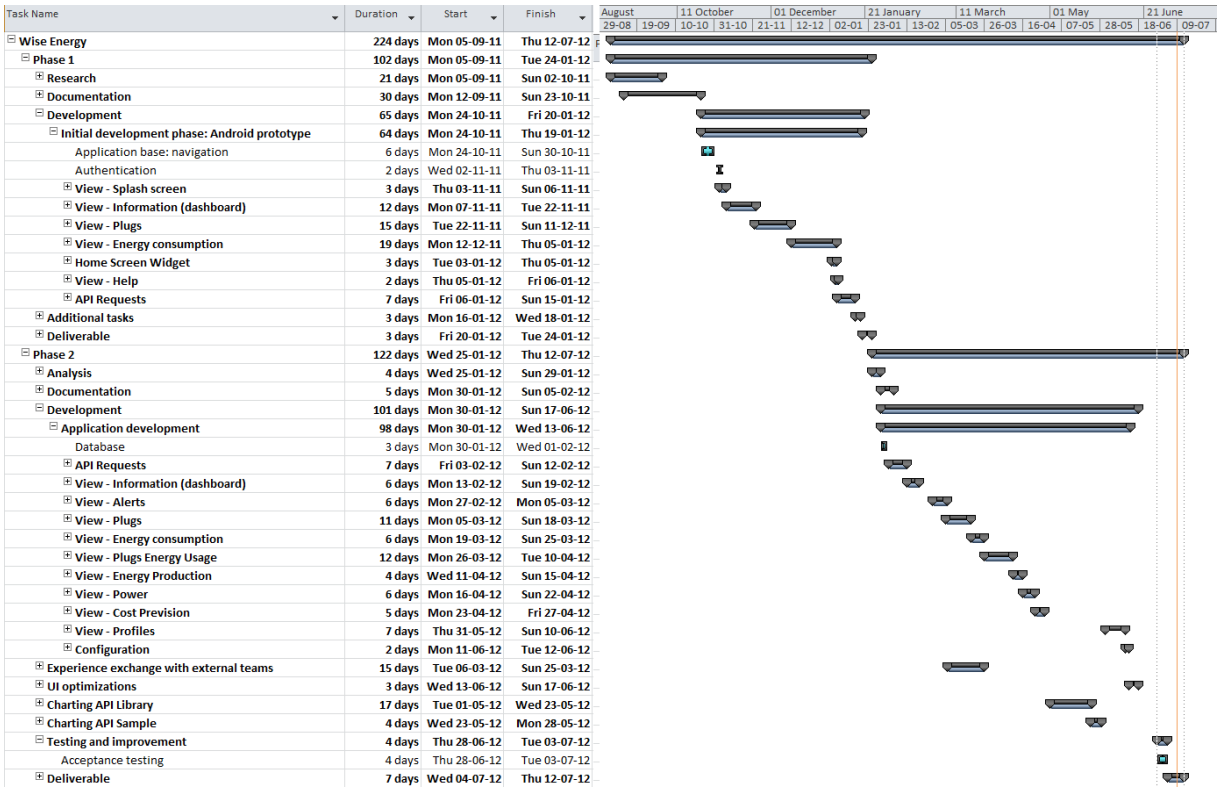
Revised planning

The work planned for the second semester suffered some modifications due to the company internal policies. Some priorities were redefined and are presented next:

- **Charting API library and sample project**
  During application development, WIT Software found an inherent value on the charting components that were been developed for the Wise Energy project and requested to design them as an external library for later integration on other Android applications. An additional sample project was created in order to demonstrate the Charting API full potential.
- **Android experience exchange with external teams**
  Moreover, WIT software decided that it would be a good working experience to be integrated in a team and working on a project in production phase. This provided an earlier and useful insight into application testing and optimizations which had later been applied to Wise Energy.

In order to cope with the previous modifications the energy efficiency module was discarded from the application once the value it initially represented had be replaced by other features that were implemented on the scope of the Wise Energy Android application project. Additionally, the time allocated for some of the features needed to be shortened in order to face these modifications. Figure 17 represents the revised project plan.

The entire remaining plan was fully accomplished.



Task Name	Duration	Start	Finish
Wise Energy	224 days	Mon 05-09-11	Thu 12-07-12
Phase 1	102 days	Mon 05-09-11	Tue 24-01-12
Research	21 days	Mon 05-09-11	Sun 02-10-11
Documentation	30 days	Mon 12-09-11	Sun 23-10-11
Development	65 days	Mon 24-10-11	Fri 20-01-12
Initial development phase: Android prototype	64 days	Mon 24-10-11	Thu 19-01-12
Application base: navigation	6 days	Mon 24-10-11	Sun 30-10-11
Authentication	2 days	Wed 02-11-11	Thu 03-11-11
View - Splash screen	3 days	Thu 03-11-11	Sun 06-11-11
View - Information (dashboard)	12 days	Mon 07-11-11	Tue 22-11-11
View - Plugs	15 days	Tue 22-11-11	Sun 11-12-11
View - Energy consumption	19 days	Mon 12-12-11	Thu 05-01-12
Home Screen Widget	3 days	Tue 03-01-12	Thu 05-01-12
View - Help	2 days	Thu 05-01-12	Fri 06-01-12
API Requests	7 days	Fri 06-01-12	Sun 15-01-12
Additional tasks	3 days	Mon 16-01-12	Wed 18-01-12
Deliverable	3 days	Fri 20-01-12	Tue 24-01-12
Phase 2	122 days	Wed 25-01-12	Thu 12-07-12
Analysis	4 days	Wed 25-01-12	Sun 29-01-12
Documentation	5 days	Mon 30-01-12	Sun 05-02-12
Development	101 days	Mon 30-01-12	Sun 17-06-12
Application development	98 days	Mon 30-01-12	Wed 13-06-12
Database	3 days	Mon 30-01-12	Wed 01-02-12
API Requests	7 days	Fri 03-02-12	Sun 12-02-12
View - Information (dashboard)	6 days	Mon 13-02-12	Sun 19-02-12
View - Alerts	6 days	Mon 27-02-12	Mon 05-03-12
View - Plugs	11 days	Mon 05-03-12	Sun 18-03-12
View - Energy consumption	6 days	Mon 19-03-12	Sun 25-03-12
View - Plugs Energy Usage	12 days	Mon 26-03-12	Tue 10-04-12
View - Energy Production	4 days	Wed 11-04-12	Sun 15-04-12
View - Power	6 days	Mon 16-04-12	Sun 22-04-12
View - Cost Prevision	5 days	Mon 23-04-12	Fri 27-04-12
View - Profiles	7 days	Thu 31-05-12	Sun 10-06-12
Configuration	2 days	Mon 11-06-12	Tue 12-06-12
Experience exchange with external teams	15 days	Tue 06-03-12	Sun 25-03-12
UI optimizations	3 days	Wed 13-06-12	Sun 17-06-12
Charting API Library	17 days	Tue 01-05-12	Wed 23-05-12
Charting API Sample	4 days	Wed 23-05-12	Mon 28-05-12
Testing and improvement	4 days	Thu 28-06-12	Tue 03-07-12
Acceptance testing	4 days	Thu 28-06-12	Tue 03-07-12
Deliverable	7 days	Wed 04-07-12	Thu 12-07-12

Figure 17 - Revised plan and Gantt diagram

Outcome

At the time this report was written, both the Wise Energy Android application and the Charting API library are fully functional. All the defined features were implemented as well as the tests made by the developer and the acceptance ones from WIT Software Quality Assurance Team. There were built two different versions – one optimized for smartphones and the other for tablets (Android 3.x support). Moreover, the charts library is currently being integrated in other projects developed inside the company.

6. Software Quality

Be a yardstick of quality. Some people aren't used to an environment where excellence is expected.
– Steve Jobs, Apple Co-founder.

In order to improve the software being developed, it is necessary to define and measure quality. The two metrics followed are the ones proposed by Crosby [43] where he defines that quality is a "conformance to requirements" and by Juran and Gryna [44] who described it as "fitness for use". The first parameter implies that requirements must be well defined and strictly followed during the development phase – quality of conformance. If a certain feature does not correspond to the defined specifications, then there is an absence of quality on the software and it must be corrected as soon as possible. "Fitness for use" defines quality in terms of design, conformance, availability, safety and field of use which leads to a viewpoint closer to the customer.

To sum up, software quality tries to measure how well software is designed – quality of design, and how the software conforms to that design – quality of concerns.

It was a constant concern to use good software development practices, which would lead to a more readable and commented code, as well as obviously good performing code. Throughout the project, the concept that small sets of code should be produced and tested was always present, either by unit or functional tests.

Extensive research was made throughout the development process in order to analyse which tools were available for testing Android applications and how they could be used on the project. Despite that, on the first initial studies the solutions found presented very few options to what was possible to test and for that reason it was necessary to test tools outside the official SDK. It is worth mentioning that as new versions of the ADT were released new testing tools started to appear which provided more feedback and better results – they later replaced some of the ones being used in the project.

The present chapter is divided into five sections:

- Build environment.
- Test environment.
- Static program analysis.
- Functional tests.
- Quality tests.

6.1. Build environment

The project was initially developed with ADT (Android Development Tools) 15.0.0 and the Android SDK for the 2.1 platform. As more releases were launched, the software had to be upgraded so that it could be kept up to date at all times.

6.1.1. Continuous integration

Continuous integration is a software development practice that aims to improve the quality of the software being written, to increase productivity and to reduce risk. This process dictates that there should exist a single source repository, containing the last successful build source code, to where each developer should integrate their code frequently. On each upload a new build is made in order to detect possible integration errors as quickly as possible.

WIT Software uses Hudson for this purpose.

Hudson

[37]Hudson is a continuous integration engine that allows the continuously building and testing of software projects and monitors the execution of external-ran jobs – for example, when a build fails, the developer responsible for the last commit receives an email with the build log.

During the project development Hudson allowed to keep a stable build of the project available at all times which could easily be accessed for testing projects.

6.1.2. Building the application

The project was built, both locally and on Hudson server, via Ant in order to ensure that the application can be compiled on different environments and there is no associated error/warning from using the IDE's own build management process.

The following tools where used to optimize the Android application *.apk* file.

ProGuard

[45] ProGuard is a tool for shrinking, optimizing and obfuscating the project source code by removing unused code and rename the remaining classes, fields and methods using shorter names. It optimizes at byte code level and removes unused instructions.

The resulting process is a smaller *.apk* file with the code obfuscated which makes it more difficult to reverse engineer the application.

zipalign

[46] Zipalign allows reducing the amount of RAM consumed when running the application on an Android device. It is an archive alignment tool which optimizes the way that an *.apk* is packaged by specifying that all uncompressed data like images or raw files, must be aligned on 4-byte boundaries. This plays a particularly role in efficient handling of processes since Android is Linux-based[6] - if the *.apk* is aligned accordingly the system knows where to look for a specific resource instead of reading through the whole package – this leads to a smoother and faster performance.

The following data is the result of the execution of the command "adb shell dumpsys meminfo" on a local emulator with Android 2.1. In order to analyse the differences between an aligned *.apk* and one without two separate tests were run.

The following data is separated according to the following structure:

- Size, the total size in the address space (heap).
- Allocated, the number of actual allocations the heap has (in kb).
- Free, the remaining free space in the heap (in kb).
- Pss, kernel metric used to analyse the memory shared on a process.
- Shared dirty, amount of RAM for the current process being shared across different processes.
- Private dirty, amount of RAM for the current process that cannot be paged to disk.

	native	dalvik	other	Total
size	12388	6791	N/A	19179
allocated	11958	3678	N/A	15636
free	61	3113	N/A	3174
(Pss)	208373	1248	12631	15962
(shared dirty)	2080	1848	7404	11332
(priv dirty)	1972	204	9936	12112

Table 13 - Memory Information (meminfo) without zipalign

[6] On Linux man page [9]: "If necessary, data structures contain explicit padding to ensure 4-byte alignment for 4-byte objects, to force structure sizes to a multiple of 4, etc."

	native	dalvik	other	Total
size	11904	6727	N/A	18631
allocated	10999	3938	N/A	14937
free	72	2789	N/A	2861
(Pss)	1973	1190	11969	15132
(shared dirty)	2076	1848	7404	11328
(priv dirty)	1868	204	9384	11456

Table 14 - Memory Information (meminfo) with zipalign

From analysing the allocated memory from each one of the *.apk* files it is possible to confirm that the gains obtained from using zipalign are significant. The RAM usage on the device was reduced from 11958kb to 10999kb – less 959kb which is the equivalent of gains of 8,01%.

6.2. Static program analysis

Static program analysis does not involve running the program. Instead, it checks the project's sources for potential bugs, dead code and if it complies with a set of defined coding rules.

6.2.1. Checkstyle

To improve the developed code's readability, Java Checkstyle is used to confirm that it follows all rules defined by WIT Software. It is a static code analysis tool that allows checking if the source code of a Java project complies with a defined set of coding rules.

6.2.2. Javadoc

The code is documented with Javadoc compliant comments.

6.2.3. Android Lint

Lint is a static program analysis tool which scans the project's source code searching for potential bugs.

Test results

Tables 15 to 17 correspond respectively to the Lint runs for the Wise Energy project, the Charting API library and charts sample application.

Project:	Wise Energy
Errors	0
Warnings	42

Table 15 - Lint: Wise Energy test results

The number of warnings displayed on Table 15table 15 is due to a set of 40 images that are used as a background animation when the user is being authenticated and the system is still loading its resources (home subscriptions, energy data, etc.). Since these images are loaded by the *AnimationDrawable* class during run time, Lint is unaware that they are being used inside the application – there is no direct call to load them and therefore a "The resource <resource_name> appears to be unused" is displayed.

The remaining two warnings correspond to a suggestion given by Lint in order to save rendering time for when a view is being loaded – "This tag and its children can be replaced by one <TextView/> and a compound drawable", meaning that instead of having two separated components – a *TextView* and an *ImageView* in a single layout. Since the first one has the possibility to add an image the second can be

discarded. Nevertheless, since this layout corresponds to the content animation mentioned before it is necessary to have both elements separated in order to access and update them individually.

Project:	ChartingAPI
Errors	0
Warnings	0

Table 16 - Lint: Charting API test results

Project:	ChartingAPIExample
Errors	0
Warnings	0

Table 17 - Lint: ChartingAPIExample test results

6.3. Functional tests

Functional tests are used to validate and ensure the quality of the software.

6.3.1. Unit tests

Even though the development process did not follow a test driven approach, an extensive collection of unit tests were created in order to ensure that the most important methods were functioning correctly.

Before describing the unit tests written it is necessary to mention that since Wise Energy is not a deterministic application, it is particularly overwhelming to design tests with a high level of coverage. On other words, this means that the data received, processed and displayed is continuously being updated making it particularly difficult to know what is the correct value to expect during an *assert* operation. For this reason, unit tests were more focused on:

- Data availability and error handling: requesting, receiving, parsing and storing data.
- Navigation through the application: launching new activities, receiving and executing events, etc.
- Switching between online and offline modes.
- Services availability: if the manager was able to bind and the data was successfully received.
- Data operations: calculating a specific timeframe, validating data, etc.

These tests were later added to Hudson and ran after every successful build.

JUnit

[47] Android provides a testing API based on the JUnit which is being used for testing new features as soon as they become available.

Robotium

[48] Robotium is a test framework that allows creating black-box test cases for Android applications. It provides an abstraction layer in simulating user events and actions – clicks, touches, etc. allowing writing and evaluating new test cases faster.

Since one of the UI requirements states that if the device is on portrait mode the tab bar should always be visible, it was necessary to redesign the activities flow in order to navigate between different activities on the same tab (more details are available on the Architecture document). This requires that the *TabActivity* and the *ActivityGroup* context needs to be available on every sub activity in order to design and access the view components.

Furthermore, since the user events are received by the current running activity (which in this situation is the *ActivityManager – ActivityGroup* – from the selected tab) it is necessary to redirect them to the active sub-activity so they can be correctly processed.

Any modification, if it is not properly tested, can lead to several exception errors that, as the project increases complexity, can be difficult to track.

In order to detect errors earlier, a set of unit testing cases were defined to test the redesigned navigation and the application context:

- Key events.
- User events.
- Switching between different activities (tabs).
- Launching sub-activities (from the current *ActivityGroup)*.

MonkeyRunner and AndroidViewClient

Since Android deals with a large variety of devices apart from the previous unit tests mentioned and before focusing onto the application behaviour and performance it is important to confirm that the View components are being drawn according to the defined design requirements (mockups) on every device. Parameters like the screen size and density have a strong impact on how a certain View is draw.

It is almost impractical to test the application on every Android device - there are too many different hardware specifications and OS releases (official and manufacturer independent). Even if this is restrained to a smaller set of equipment's the process itself would be very time consuming and repetitive. Nevertheless, quality should not be disregard – as Steve Jobs once said *be a yardstick of quality*, so alternative solutions were studied and later developed.

The approach followed was to use MonkeyRunner as an automation tool responsible for simulating the user touch input data and AndroidViewClient to access the application Views. This last tool, as mentioned before, provides a high level interface that supports accessing View properties by its id, instead of entering hardcoded coordinates as it is required by MonkeyRunner. Nevertheless, it has still some limitations that needed to be overcome, especially when interacting with a *TabWidget* that on WiseEnergy represents the main navigation component. This feature was necessary to implement, and was developed for not being application-specific, meaning that it can later be used by other development teams inside WIT Software.

This test project was developed using Python. Given the importance of Wise Energy being integrated with Hudson, a new requirement was defined – it must be possible to run these tests on headless mode. Moreover, since MonkeyRunner has support for multiple device control (it is possible to launch several instances of the Android emulator and execute in each one of them the defined tests independently), an external interface for defining the emulator parameters was written. This way it is possible to define the screen dimension, android version, etc. and automatize the launching process.

As mentioned before, MonkeyRunner allows capturing the user interface and then comparing it with default set of images which correspond to the application expected behaviour. Nonetheless, it is necessary to take into account that there are several elements on the view that can change from different runs and devices (which in this case corresponds to the emulator with a specific set of parameters defined), for example: the status bar elements like the battery level, time, etc. After analysing several tests results these differences corresponds approximately to 3%. This is an acceptable value and was used as threshold for the automatic tests – if when comparing two images the difference is over 3% the test fails.
Moreover, it was analysed the possibility of removing the status bar both from the reference image and the actual one, nevertheless that involves several intermediary steps that will only add a degree of complexity, reduce performance and increase memory usage and running time.

Emma

[42] Emma is a code coverage tool developed for Java projects. It supports several types of coverage: class, method line and basic block and can detect when a single source code line is covered on partially.

Overall, approximately 230 unit tests were written using JUnit, covering about 44% of all the project classes. Since the application deals with a continuous flow of data, most of the 56% of classes left untested corresponded to exception classes, interfaces or configuration/default files.

	Packages	Exec. files	Classes	Methods	Exec. lines
Total	24	131	288	2348	11932

Table 18 - Emma: Overall stats summary

Name	Class, %	Method, %	Block, %	Line, %
All classes	44% (136/313)	23% (610/2709)	10% (7854/78254)	13% (1083/13701)

Table 19 - Emma: Overall coverage summary

Note: the previous results represent the tests made using the JUnit test framework. Due to limitations from Emma it was not possible to calculate the percentage of code coverage by the tests written on Robotium.

It is worth to mention that additional to those 230 tests another set of 190 were written using Robotium and were focused on testing the application context (state) while simulating the user navigation across different activities.

6.3.2. Regression testing

As new features were implemented on the application the above tests were used to evaluate the application stability.

6.3.3. Acceptance tests

The acceptance tests were conducted by WIT Software's quality team and follow a test specification sheet created for that purpose.

Results

The previous acceptance tests were run by WIT Software Quality Team on a Samsung Nexus S with Android 2.3.3. This device was selected due to its popularity on the market and since the installed OS version is the one used by more than 60% of all smartphones and tablets that daily access the Google Play store [49].

The SQA team performed the tests to the application using a black box approach: testing the entire system from a typical user's point of view.

When a bug was found, it was reported using Mantis Bug tracker. After several iterations of testing, bug reporting and correction the project passed on all of the defined acceptance tests. Table 20 resumes these results.

Status	Total	Percentage
Passed	120	100%
Failed	0	0%
Not implemented	0	0%

Table 20 - Wise Energy acceptance tests: result status resume

6.4. Quality tests

Quality tests focus on analysing the application behaviour. Contrary to the functional tests described on the previous section they do not related with functionality but instead they are designed to evaluate if the performance, security, usability, etc. of the requirements are met.

The tests made can be grouped into the following categories:

- Software performance
- Usability
- Security
- Internationalization
- Battery
- Installation

Additionally, there a last section where a test report tool used during the project development is analysed.

6.4.1. Software performance

In order to analyse what was possible to optimize on the application, to discover possible bottlenecks, to measure and evaluate the view drawing time, etc. several tools were used and different test ran.

Monkey

Monkey is a stress testing tool developed by the Android team which simulates the user continuous interaction with a particularly application. It generates a set of pseudo-random streams of events: clicks, touches, gestures, etc. on any part of the screen in order to analyse the application behaviour. When a crash, an unhandled exception or ANR (Activity Not Responding, it is an error dialog launched by the OS for when the UI thread is without responding for more than 5 seconds, it allows to kill the application) occurs, the program terminates reporting the problem.

The following table represents the number of events registered and the application behaviour.

Number of events:	Success	Failed	Severity
500	10/10	0/10	N/A
1000	10/10	0/10	N/A
10000	10/10	0/10	N/A
100000	10/10	0/10	N/A
1000000	09/10	1/10	Low
10000000	4/10	6/10	Low*
100000000	0/10	10/10	Low*

n/a –not applicable
* –most of them are not related with Wise Energy

Table 21 - Monkey: test results.

Note: Since we are dealing with random events, every set was executed 10 times.

When the number of events was increased to 100000000 the tests started to fail. On three of them the problem registered was due to an excessive memory usage which made the system terminate the application, the other ones were related to problems outside Wise Energy and were usually related with the Android launcher when the “back” key was pressed. A similar situation happened during the last test,

but in this case none of the errors were related with the application – in every one of them, the test was stopped by a system-related error.

Hierarchy Viewer

[50] Hierarchy Viewer is an Android debug tool which inspects the application layout and displays the measure, layout and draw times for a specific node. This corresponds to the duration (in ms) that the OS spent to calculate the view width and height, to find its position and to draw its content, respectively.

In order to analyse if any of the above metrics can be optimized, Android displays a set of performance indicators that identify the slower objects in a View. This tool had been widely used on the entire application – every component had been evaluated – which allowed to obtain significant improvements on the application user interface.

Pixel Perfect

[51] Pixel Perfect is a tool available with Hierarchy Viewer which focuses on providing a high detailed view of the application UI. It displays a magnified image of the screen that is currently visible on the emulator/device and allows examining the properties of individual pixels in the screen image.

6.4.2. Usability tests

In order to test the application usability, a set of tasks were written (for example, to edit a plug's name, etc.) and presented to a group of WIT employees that had no previous contact with the application nor the project.

These tests are available and can be consulted on the following document annexed to this report: WiseEnergy_-_Appendix_F_-_Usability_Tests.

Note: Although, the present section refers to usability testing for mobile applications, the same process can and should be expanded to other software products.

During the Wise Energy development it is possible to separate usability testing into two different phases:

1. Throwaway prototyping.
2. Working prototype.

Throwaway prototyping was accomplishing during the initial mockup and wireframe design, and was later revisited when new features were added to the application. During this phase, the requirements were mapped into mockups and submitted to the project manager. After his analysis, the wireframes were built along with the supported user interaction gestures (click, touch, drag, pinch, etc.) as well as the final aesthetics and sent to the PM for revision. Once approved, they were submitted to the product owner for further acceptance.

At the end of the development phase a working prototype was built and a usability study was conducted in order to evaluate the user's response towards the application.

The application design and wireframes can be accessed on the following document annexed to the present report: WiseEnergy_-_Appendix_C_-_Application_Mockups.

Subject Profiles

The subjects for the study were 8 employees from WIT Software with a degree in Informatics Engineering. They were between the age of 22 to 29 and only one of them was a female. They were asked to participate on a usability test and accepted it freely, even without any type of reward being promised.

Note: it is worth to mention that since Wise Energy is a confidential product it was mandatory that all of the test subjects were WIT Software employees.

Environment

In order to provide a more realistic approach every test was performed on a real device instead of an emulator on the computer. The tests were recorded with another device for further analysis.

Testing the application on every mobile phone is impossible; we do not either have the resources for that nor the number of users needed to test that range of equipment's. Therefore, two different devices were chosen according to a market study made on the most sold Android devices and their availability inside WIT Software:

- Samsung Nexus S with Android 2.3.3
- Samsung Galaxy Tab 10.1 with Android 3.1

Since Android is upward compatible (an application built for 2.1 should run on the 4.0.3 without any major problems) two separated versions were generated:

- Android 2.1
- Android 3.0

The first one for smartphones and the second optimized for tablets.

The tests were done on a controlled relaxed environment where the testers were free to discover the application.

Tasks

The study was planned for 30 minutes per subject (with an additional 5 minutes if the user had no previous interaction with an Android device) and a set of approximately 60 tasks were proposed for that time. Although, it might seem a large number these did were short direct tasks that intended to test if the requirements defined before were implemented properly.

It is also worth to mention that every task was written along with a small user story which reflected the reason behind that particularly action. They intended to provide a better feedback to the user by providing the reasons on why that task is relevant or what useful information it is possible to obtain.

Metrics

The metrics used to measure the outcome results were:

- **The duration of each task.**
  It was further analysed if this value corresponded to estimated time or not.
- **The result of each task**
  Each task was categorized in one of the following states: passed, found it difficult and failed, which reflected the user performance.
- **Number of actions**
  The testers were continuously monitored in order to analyse if they were able to complete the task within the expected number of actions.

The report contained another section in order to jot down any additional remark.

Results

Before starting the present analysis it is necessary to acknowledge that, although Nielson states that "it takes only five users to uncover 80 per cent of high-level usability problems", it is not possible to generalize these conclusions to the global user community once the test group was limited to 10 people.
Since every mobile OS has their own architecture and a specific user interface, in order to reduce the noise that could be caused by the tester unfamiliarity with the platform on the outcome results, an initial study was made reflecting the testers' device and it is represented on figure 18.

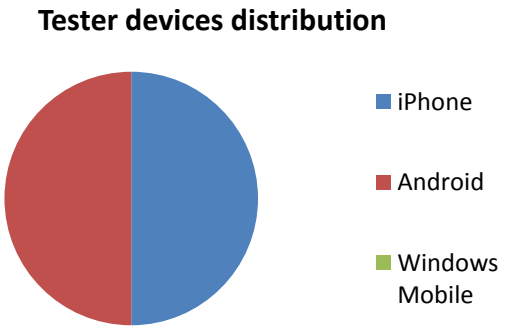
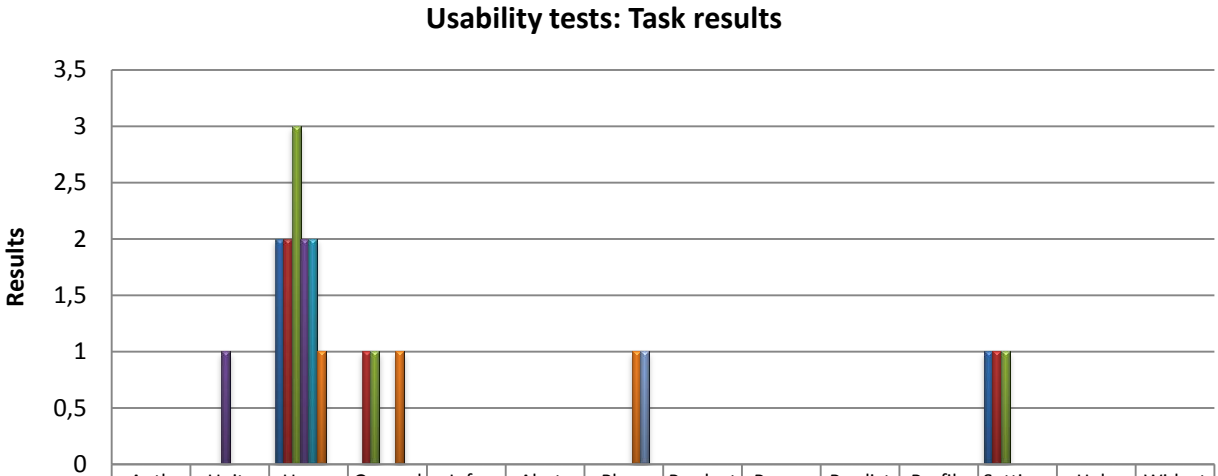


Figure 18 - Testers device distribution

To the 50% of subjects that did not have an Android the solution followed was to allow them to use the testing device for a 5 to 10 minute period before starting the test. This allowed them to get more familiarized with the user interface.

It was suggested that they should try to access the same features the have on their device's contacts and mail application since on both platforms they are easy and direct to use and provide the same basic functionalities.

It is worth to mention, that apart from the concept that Android has an "options menu" which provides a set of actions otherwise unavailable (iOS only has a home button which minimizes the running) there were no more difficulties registered.



	Auth	Units	Usage	General	Info	Alerts	Plugs	Product	Power	Predict	Profile	Settings	Help	Widget
Tester 1	0	0	2	0	0	0	0	0	0	0	0	1	0	0
Tester 2	0	0	2	1	0	0	0	0	0	0	0	1	0	0
Tester 3	0	0	3	1	0	0	0	0	0	0	0	1	0	0
Tester 4	0	1	2	0	0	0	0	0	0	0	0	0	0	0
Tester 5	0	0	2	0	0	0	0	0	0	0	0	0	0	0
Tester 6	0	0	1	1	0	0	1	0	0	0	0	0	0	0
Tester 7	0	0	0	0	0	0	1	0	0	0	0	0	0	0
Tester 8	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19 - Usability tests: Task results

Figure 19 represents the task results from the defined usability tests. In order to understand which areas would require a more detailed attention, these were grouped into a set of sections that reflect the application most important features.

Before analysing these results, it is important to mention that the testers completed all the proposed tasks. The values represented on Figure 19 correspond to the degree of difficulty measured by comparing the expected and duration time of each task *versus* the outcome result (passed or failed) and the number of actions needed to achieve it.

Since every task was completed, the metrics weight was upgraded in order to account only the time spent on each task and the associated number of actions. The yy-axis values correspond to the number of tasks that require attention. For example, for the units section the tester 4 had difficulties in changing the application unit type from € to kWh.

After compiling the above information, it is possible to analyse that there still room for improvement on some sections of the application, namely:

- When asked to access the energy consumption data for a specific module, most users were unable to associate the magnifying glass present the consumption view to the possibility to filter the energy data being displayed. For about 60% of the users the first action was to access the plugs management tab and all of its sub-views until they realize that they needed to go back to the consumption tab.
- The TOU rates are displayed on the settings tab due to inner requirements. According to the test this reveals that it is not a good design choice since most of the users associated this area to the user account settings or the application options.
- The general area corresponds to set of actions available for interacting with the chart data (available only landscape mode). 26% of the users did not realize that if they touched/dragged their fingers inside the graph area they could select a specific value or define a new visualization window.
- Finally, the difficulties found on the plugs area were more platform-specific. Users that had an Android device did not realize at first that they could organize the connected plugs by a simple touch and drag movement.

Although some minor tweaks were found during the usability tests, the overall results were good and should be taken into account along with the testers' feedback on future work.

Note: The defined tasks as well as the results are available on the following document: WiseEnergy_-_Appendix_F_-_Usability_Tests.

6.4.3. Security tests

Since the product deals with private information it is necessary to prevent system intrusions. If it is not properly secured outside sources can detect the user consumption patterns, inferring when someone is at home.

Wireshark

[52] Wireshark is a network protocol analyser and was used to determine if it was possible to access to the content from the application network traffic.

Results

The application content was encrypted and therefore it was not possible to access the user's home energy use.

Intent Fuzzer

[53] Intent Fuzzer is a tool which allows testing several security parameters of an application.

It was to test the Wise Energy Broadcast receivers, services and activities. Although, a single test can be used to analyse the application behaviour for broadcast receivers, and other one for services, the same is not possible for activities. Since this tool can only test one activity on each run it was necessary to create 28 additional tests.

Results

The application passed on every test made.

Test	Passed	Failed	Number of tests
Broadcast receivers	100%	0%	1
Services	100%	0%	1
Activities	100%	0%	28

Figure 20 - Intent Fuzzer test results

6.4.4. Internationalization tests

An internationalization test was made in order to analyse the application behaviour when the default language is changed.

This test was made by changing the device language from Portuguese into English and then manually evaluate each view in order to confirm that there were no fields left translating, the unit format had changed (from comma to decimal point), and that the application components were still aligned as they were before (since the text length has changed it is necessary to evaluate if the view fields are not defined to wrap its content).

6.4.5. Battery tests

Battery life and an application performance are two of the most important performance-based attributes.

An application must be optimized to have the lowest impact on the device resources. This does not mean that some feature should be discarded in order to increase battery life. Instead, the developer should analyse the user actions on a smartphone, what she expects when running an application and what functionalities the OS offers.

Being Wise Energy an energy application this gains even more focus. The entire project was designed to provide high responsiveness to user actions and to present data in near-real time (currently 5 seconds delay) with the lowest impact on the system resources and battery possible.

In order to establish a comparison between other applications, some of the most used ones were downloaded and analysed:

- Gmail, an email client by Google.
- Contacts application.

The devices used for these tests were:

- LG P-500 with Android 2.3.3
- Samsung Nexus S with Android 2.3.3

It was defined three different scenarios:

- On the first one, the application was launched and the devices were left on top of a table for an 8 hour period with the screens off. After the first two hours the keyboard was unlocked and the battery level registered; the same process was repeated every two hours.
- On the second scenario the screen was kept unlock all the time in order to analyse the services impact on the application (when it is off they are automatically stopped). The same process occurred during the same 8 hours – the battery level was analysed every two hours.
- The third scenario corresponds to the normal usage of the application and was done during the usability tests. The battery was initially measured, than a set of tasks were given to a tester and after she had accomplished them the new value was registered.

The widget was tested independently using the first and second scenario.

Note: since the first two scenarios can be done automatically a small application was written and every 2 hours the *AlarmManager* was launched and the battery level registered on a local log file along with the time it had measured.

In order to provide a coherent result every test was run for 5 times with the same applications open in every terminal. Nevertheless, it is necessary to keep in mind that there are a set of parameters that can affect the battery levels and are not possible to control: the network signal strength, the system services, etc. are some of these examples.

LG P-500	#1	#2	#3	#4	#5	Average
Threshold (8 hr):	-10%	-12%	-13%	-10%	-09%	-10.8%
Scenario 1 (8 hr):	-13%	-10%	-10%	-11%	-11%	-11.0%
Threshold (8 hr):	-39%	-47%	-46%	-37%	-41%	-42.0%
Scenario 2 (8 hr):	-53%	-52%	-51%	-51%	-54%	-52.2%
Threshold (30 mins):	-04%	-03%	-03%	-03%	-03%	-3.20%
Scenario 3 (30 mins):	-03%	-05%	-04%	-04%	-05%	-4.20%

Table 22 - Battery test: LG P-500

Nexus S	#1	#2	#3	#4	#5	Average
Threshold (8 hr):	-06%	-06%	-07%	-07%	-07%	-6.60%
Scenario 1 (8 hr):	-06%	-07%	-05%	-07%	-07%	-6.40%
Threshold (8 hr):	-17%	-13%	-15%	-14%	-13%	-14.4%
Scenario 2 (8 hr):	-21%	-19%	-19%	-18%	-19%	-19.2%
Threshold (30 mins):	-02%	-01%	-01%	-01%	-01%	-1.20%
Scenario 3 (30 mins):	-02%	-03%	-02%	-02%	-03%	-2.24%

Table 23 - Battery test: Samsung Nexus S

Note: The threshold values correspond to the battery tests made with the Gmail and contacts application running on background/foreground.

Tables 22 and 23 represent the results of the battery tests made. As it is possible to analyse the application behaviour does not stray too far from the threshold measured. It is worth to mention that for the second scenario the device battery lost almost 50% of its charge on the LG P-500 mainly due to the screen being on during the 8 hours timeframe. As it was expected, for the same scenario, keeping the application continuously requesting new data had a higher impact on battery life.

It is also worth to mention that on the first scenario on the Nexus the battery life was slightly lower than the defined threshold. This is due to the fact, that when the screen is off the application services are completely stopped, therefore not having any significant impact on the device's battery. The same does not apply on the Gmail application (threshold value) which from time to time checks if the user has received any new emails.

6.4.6. Installation tests

The installation tests focused on installing Wise Energy on different Android powered devices: smartphones and tablets and analyse the application behaviour across these platforms. The OS version installed was also taken into account.

These are short duration tests that involved installing the application on a clean device, authenticate the user and navigate across different views.

Device	Android release	Status
Samsung Nexus S	Android 2.3.3	Passed
Samsung Galaxy S II	Android 2.3.3	Passed
Samsung Galaxy S II	Android 4.0.1	Passed
Samsung Galaxy Nexus	Android 4.0.3	Passed
Samsung Galaxy Tab 10.1	Android 3.1	Passed
LG LP-500	Android 2.1	Passed
Sony Ericsson X10	Android 2.1	Passed

Table 24 - Installation test results

The application worked without problems in every device tested.

7. Conclusion

Things won are done, joy's soul lies in the doing.
– William Shakespear, poet and playwright

7.1. Accomplishments and work done

From a goal perspective, it is safe to say that the internship has met the goals defined for both semesters.

In terms of work done the following list presents the most important tasks, ordered chronologically:

- **State of the art analysis**
  Corresponds to the research made during the first semester about the smart grid concept and its technologies, the energy efficiency solutions available on market their features and limitations and the analysis of the value that these products can bring to the consumer and to utilities.

  Moreover, a study was made about future development, technologies and products available for energy efficiency.
- **HAN's and protocols used for wireless communication**
  An initial research about which protocols are used for wireless communication inside HAN's was made, giving a special focus on ZigBee®.

  Although, ZigBee® is defined as standard, some fields used for communication are manufacturer dependent – this means, that not every device is interoperable. A more deep analysis was made in order to understand and specify these differences.
- **Requirement elicitation and analysis**
  An initial requirement elicitation and analysis was made during the first semester and had later been revised in order to add new features to the Charting API.
- **Domain area knowledge**
  Since Wise Energy deals with information from other domain areas, like physics, it was fundamental to conduct a background research about several subjects, namely energy and power and how this two correlate
- **Mockup and wireframe design**
  After the requirements were defined the prototyping phase started. After the initial mockups had been approved the application wireframe was designed.
- **Application architecture**
  The application architecture follows the Android UI guidelines and design patterns.

  A document describing the entire application according to the IEEE 1471 standard is available on annex.
- **Charting API library and sample application**
  The Charting API library was designed to be as extensible and modular as possible facilitating the process of implementing new futures in the future.

  A document describing the entire application according to the IEEE 1471 standard is available on annex.
- **Unit and functional testing**
  During the development phase several unit tests were written in order to assure that the application behaviour was consistent. When it was fully developed new types of tests were ran.

  A document describing the tests made to the application is available on annex.
- **Acceptance testing**
  When it was fully developed and the Wise Energy application (*.apk* file) was sent to WIT Software Quality Team so they could run a list of acceptance test that mapped the defined requirements.

After a set of iterations which, included testing, report and error correction the application passed on all of the defined tests.

- **Documentation**
  The code is commented accordingly to the Javadoc standards defined by Oracle® for Java programs.

7.2. Contributions

Apart from the Wise Energy Android application, several other significant contributions were made in the context of the internship:

- **Android**
  During application development some limitations were found on the Android SDK Android Support Package. These were reported to the Android Development Team (some) along with a patch for a possible solution.
- **Charting API**
  The Charting API is an Android library initially developed for the Wise Energy application in order to add graphical support to data visualization. At this moment, it is independent from the initial project and is being used on other Android applications developed by WIT Software.

  New features were added and currently it provides full customization of the graph components and offers a wide range of charts. Additionally, a sample project was created in order to demonstrate its potential.
- **Components developed**
  The drag-and-drop interface developed for managing the application modules was released for internal used and had already being integrated with other Android projects.
- **Android experience exchange with external teams**
  Being integrated in a team and working on a project in production provided an earlier and useful insight into application testing. It was an enriching experience that made possible to discover and discuss several ideas about Android development, architecture design and UI optimizations. During that time, the main focus was to reduce battery usage, optimization on loading content to a view and test the application UI/UX on different Android powered devices.

7.3. Risk management reflection

During the project development some of the identified risks presented on the initial chapter of this report occurred. This section focuses on the analysis of these risks and the efficacy of the mitigation strategies earlier defined and used.

Since Wise Energy relies on an external server for retrieving the home's energy data it is necessary to consider that it is not possible to guarantee 100% uptime. When for any reason the application could not connect to the server it would retry until an x-number of attempts was registered. When this number was reached a *Dialog* warning the user that, at that moment, it was not possible to establish a remote connection was displayed and the current view mode was changed to offline, where the data instead of being requested to the server it was retrieved locally (database). The same process was used for when the device was not connected to the Internet.

Another risk related with the API requests was that the application must not assume that the responses received are error-free. This happened during the initial development and was reflected on sudden crashes due to parsing exceptions. The solution defined was to create a set of error detection methods – that were later integrated into handlers – that would parse the received data and analyze if every parameter corresponded to the defined specification. This had particular importance during the entire development phase, since the server-side was often updated leading to some inconsistent data that could otherwise reflect on unexpected application behavior.

Finally, the risk of running the application on different Android devices and releases was always kept present. The first approach followed – feature related – was to use the Android Support Package, an

external library which provides backwards support for older Android versions. Nevertheless, there were still some limitations on certain features, for example Android 2.0 is unable to detect multi-touch events, which had to be disabled in order to avoid an unexpected behavior. Moreover, other techniques were necessary to use in order to provide the same design and UI look across different platforms (this is discussed on more detail on the development chapter).

7.4. Future work

Since all of the defined goals were accomplished, there is not much left to do – feature wise – both on Wise Energy and on the Charting API.

On a short time frame, the usability results discussed on the Software Quality chapter should be further analyzed and a larger study should be made in order to create more viable conclusions. There still room for improvement on some sections of the Android application, namely the selection of a specific module in order to analyze its energy consumption. Most users took more than the expected time to understand how they could achieve this, and to many of them it was necessary to provide a hint. Supporting that good software does not require a user manual, it is necessary to re-think how this feature should be re-integrated within the application and repeat the usability tests.

The Android platform is continuously evolving, at the time this report was written a new release of the OS had just been launched – Android 4.1 codename Jelly Bean with a new set of features and tools that can be used to upgrade the application. The UI design has suffered a huge shift on Android 4.0 and although some initial mockups were designed which supported this new modifications, they are still not implemented on Wise Energy. This is due to two connected factors: some of the components used are now deprecated, particularly from the presentation layer of the application, new alternatives for the *ActivityGroup* and *TabActivity* were written and actually represent several gains of performance and usability. Nevertheless, redesigning the UI navigation would become a time consuming task with no direct major gains since the percentage of users that currently have a 4.x version installed represent less than 8% of the entire Android community. Moreover, most of the devices currently on market have no official upgrade nor it is predicted that most of them would receive one anytime soon.

The Charting API library is currently being integrated on other Android applications inside WIT Software each one with a particular specification and set of goals. Although, the defined requirements as well as other secondary features were implemented as the library is used on other projects, new functionalities will likely be necessary to write and add to the current version.

7.5. Lessons learned

As an exception to the rest of this report, once this section represents the author's view of the internship, it is written in the first person, in a slightly less formal language.

At the time of writing this report it is fair to say that the path followed on the last 10 months have resulted on creating a production-ready Android application – Wise Energy – and a chart library for the same platform which is currently being integrated on other internal projects. It is without doubt, one of the biggest challenges of someone that is learning to become a Software Engineer. Having the opportunity to think and design a solution from the group up, allowed to experience all of the development stages before a product goes into production. The experience of developing such project has proved to be an excellent learning opportunity with continuous challenges and goals, sometimes, hard to attain.

Since most of the project was written using the Android SDK it worth to mention some of the platform aspects and its community. Developing an Android application is not quite simple as one can initially think. Before starting, it is necessary to clarify that the word "simple" used to describe a project, means that it does not have much features, it is constituted by two or three activities at most and if it requires communicating with an external source in order to retrieve data this corresponds to single-isolated situation. I agree that developing such a "simple" project can be quite straightforward to accomplish on Android. Nevertheless, when it is necessary to build more complex applications, which require continuous communication with external services in order to present data in a near-real time environment, we are dealing with a large set of views (activities), it is necessary to implement components that are not available natively, and the most important the application must be supported by a huge variety of Android

powered devices, each one with its own hardware specifications and manufacturer OS release becomes a quite challenging task. Every one of these problems were addressed during the development and testing phase and for the last one additional methods were developed for guaranteeing that the application behaviour was consistent across different devices.

For the past 8 months, I have been following its development, the continuous launch of new features and tools which allow us, programmers, to write richer applications. Although, this was not an isolated situation, I find this one particularly worth to mention due to the inner satisfaction it provided and although it was on a very small scale the sense of contribution to the improvement of a product: when testing Wise Energy on past android releases I kept struggling with an error related with a *NullPointerException* that was returned from the *ViewPager* component when the device was on landscape. After several days of searching and attempting to understand the problem I have realized that it was an internal bug on the Android Support Package – I have submitted it to the Android Developers Team and created a small patch which I sent. Days later, they sent me a response thanking me from the report along with a new release of their support package.

As I write these final words, I cannot help to lookup back and analyse this last year's work. It was a toilful internship, full of enriching experiences. From day one I have been learning how to be a better software engineer, to create a final product by following the different phases of software development, and in the end to accomplishing those remaining 10% that are usually forgotten and "account for the other 90% of the development time".

8. References

[1] Google Developers, “Android,” Google, [Online]. Available: https://developers.google.com/android/. [Accessed 10 July 2012].

[2] Apache, “The Apache Ant Project,” [Online]. Available: http://ant.apache.org/. [Accessed 10 July 2012].

[3] Android Open Source Project, “Dalvik Technical Information,” [Online]. Available: http://source.android.com/tech/dalvik/index.html. [Accessed 10 July 2012].

[4] GE, “GE Appliances,” 2012. [Online]. Available: http://www.geappliances.com/home-energy-manager/. [Accessed 11 January 2012].

[5] Smarthome, "What is INSTEON," 2012. [Online]. Available: http://www.smarthome.com/INSTEON_basics.html. [Accessed 22 January 2012].

[6] Smarthome, "Getting Started," 2012. [Online]. Available: http://www.smarthome.com/about_x10.html. [Accessed 22 January 2012].

[7] Z-Wave, "FAQs," [Online]. Available: http://www.z-wave.com/frys/faqs.html. [Accessed 22 January 2012].

[8] Digi, “ZigBee,” 2012. [Online]. Available: http://www.digi.com/technology/rf-articles/wireless-zigbee. [Accessed 22 January 2012].

[9] WIT Software, "WIT Software," 2011. [Online]. Available: http://www.wit-software.com/. [Accessed 15 January 2012].

[10] Economist, "Wiser wires," 8 October 2009. [Online]. Available: http://www.economist.com/node/14586006. [Accessed 12 January 2011].

[11] Opower, "Opower," 2012. [Online]. Available: www.opower.com. [Accessed 7 January 2012].

[12] Google, "Google PowerMeter," 2011. [Online]. Available: http://www.google.com/powermeter/about/. [Accessed 7 January 2012].

[13] Microsoft Hohm, "Microsoft Hohm," 2011. [Online]. Available: http://www.microsoft-hohm.com/. [Accessed 20 January 2012].

[14] first:utility, "first:utility," [Online]. Available: http://www.first-utility.com. [Accessed 08 January 2012].

[15] ISA, "Solutions & Services. Energy Efficiency," 2010. [Online]. Available: http://www.isasensing.com/index.php?section=energy&action=details&id=9. [Accessed 20 January 2012].

[16] ONZO, "ONZO," [Online]. Available: www.onzo.com. [Accessed 8 January 2012].

[17] Energy Aware, "Energy Aware," 2011. [Online]. Available: http://www.energy-aware.com/. [Accessed 7 January 2012].

[18] Nest Labs, "Nest™," 2011. [Online]. Available: www.nest.com. [Accessed 8 January 2011].

[19] PassivSystems, "PassivSystems," 2012. [Online]. Available: http://www.passivsystems.com/. [Accessed 20 January 2012].

[20] Google, "Google I/O 2011 - Android@Home," Google, 11 May 2011. [Online]. Available: http://www.youtube.com/watch?v=3SNPFPKS4U4. [Accessed 9 January 2012].

[21] eragy, "eragy," 2012. [Online]. Available: http://www.eragy.com. [Accessed 11 January 2012].

[22] GE, "GE ecomagination," 2012. [Online]. Available: http://ge.ecomagination.com/smartgrid/#/landing_page. [Accessed 10 January 2012].

[23] EnergyHub, "EnergyHub," 2012. [Online]. Available: www.energyhub.com. [Accessed 11 January 2012].

[24] GridPoint, "GridPoint | Smart energy solutions for enterprise and utilities," 2010. [Online]. Available: http://www.gridpoint.com/home.aspx. [Accessed 11 January 2012].

[25] Serious Energy, "Energy Monitoring System as SaaS," 2011. [Online]. Available: http://www.seriousenergy.com/energy-management/platform/saas.html. [Accessed 18 January 2012].

[26] OGE, "OGE," 2012. [Online]. Available: http://www.oge.com/Pages/Home.aspx. [Accessed 19 January 2012].

[27] GEO - Green Energy Options, "GEO," 2010. [Online]. Available: http://www.greenenergyoptions.co.uk/. [Accessed 19 January 2012].

[28] Smarthome, "Smarthome - Home Automation Superstore," 2012. [Online]. Available: http://www.smarthome.com/_/index.aspx. [Accessed 19 January 2012].

[29] AlertMe, "AlertMe," [Online]. Available: http://www.alertme.com. [Accessed 14 January 2012].

[30] GreenWave Reality, 2012. [Online]. Available: http://www.greenwavereality.com/. [Accessed 10 January 2012].

[31] Motorola, "4HOME Software Solutions," [Online]. Available: http://www.motorola.com/Video-Solutions/US-EN/Solution-Sites/4HOME/Overview. [Accessed 8 January 2012].

[32] Techcrunch, "iOS Market Share Up From 26% In Q3 To 43% In Oct/Nov 2011," 9 January 2012. [Online]. Available: http://techcrunch.com/2012/01/09/ios-marketshare-up-from-26-in-q3-to-43-in-octnov-2011/. [Accessed 22 January 2012].

[33] android-ui-utils, "Android UI Utilities," [Online]. Available: http://code.google.com/p/android-ui-utils/. [Accessed 19 January 2012].

[34] SmartDraw, "SmartDraw Communicate Visually," 2012. [Online]. Available: http://www.smartdraw.com/. [Accessed 21 January 2012].

[35] Soyatec, "eUML2," 2011. [Online]. Available: http://www.soyatec.com/euml2/. [Accessed 21 January 2012].

[36] Sybase, "PowerDesigner," 2012. [Online]. Available: http://www.sybase.com/products/modelingdevelopment/powerdesigner. [Accessed 21 January 2012].

[37] Hudson, "Hudson - Extensible continuous integration server," [Online]. Available: http://hudson-ci.org/. [Accessed 09 July 2012].

[38] Python, "Python Programming Language - Official Website," 2012. [Online]. Available: http://www.python.org/. [Accessed 10 July 2012].

[39] Android Developers, "Platform Versions," [Online]. Available: http://developer.android.com/about/dashboards/index.html. [Accessed 09 July 2012].

[40] IEEE, "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems," IEEE, 2000.

[41] Lorenz, "How to Create QuickAction Dialog in Android," 12 July 2010. [Online]. Available: http://www.londatiga.net/it/how-to-create-quickaction-dialog-in-android/. [Accessed 13 January 2012].

[42] EMMA, "EMMA: a free Java code coverage tool," [Online]. Available: http://emma.sourceforge.net/. [Accessed 10 July 2012].

[43] P. B. Crosby, Quality Is Free: The Art of Making Quality Certain, Mentor, 1979.

[44] J. M. Juran and F. M. Gryna, Quality planning analysis: From product development through usage, McGraw-Hill, 1970.

[45] Android Developers, "ProGuard," Google, [Online]. Available: http://developer.android.com/tools/help/proguard.html. [Accessed 09 July 2012].

[46] Android Developers, "zipalign," Google, [Online]. Available: http://developer.android.com/tools/help/zipalign.html. [Accessed 09 July 2012].

[47] JUnit, "JUnit.org Resources for Test Driven Development," [Online]. Available: http://www.junit.org/. [Accessed 09 July 2012].

[48] robotium, "robotium - It's like Selenium, but for Android™," [Online]. Available: http://code.google.com/p/robotium/. [Accessed 09 July 2012].

[49] Android Developers, "Platform Versions," [Online]. Available: http://developer.android.com/about/dashboards/index.html. [Accessed 09 July 2012].

[50] Android Developers, "Hierarchy Viewer," [Online]. Available: http://developer.android.com/tools/help/hierarchy-viewer.html. [Accessed 10 July 2012].

[51] Android Developres, "Optimizing Your UI," Google, [Online]. Available: http://developer.android.com/tools/debugging/debugging-ui.html. [Accessed 10 July 2012].

[52] Wireshark, "Wireshark," [Online]. Available: http://www.wireshark.org/. [Accessed 10 July 2012].

[53] ISEC Partners, "Mobile Security," [Online]. Available: https://www.isecpartners.com/mobile-

security-tools/. [Accessed 11 July 2012].

[54] G. O'Regan, A Practical Approach to Software Quality, Springer, 2002.

[55] S. H. Kan, Metrics and Models in Software Quality Engineering, Addison Wesley, 2002.

[56] M. Fowler, "Continuous Integration," 01 May 2006. [Online]. Available: http://www.martinfowler.com/articles/continuousIntegration.html. [Accessed 09 July 2012].

[57] dtmilano, "dtmilano/AndroidViewClient," [Online]. Available: https://github.com/dtmilano/AndroidViewClient. [Accessed 09 July 2012].

[58] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord and J. Stafford, Documenting Software Architectures, Addison-Wesley, 2011.

[59] P. Clements, "Comparing the SEI's Views and Beyond Approach for Documenting Software Architectures with ANSI-IEEE 1471-2000," CMU/SEI, 2005.

[60] E. Chin, A. P. Felt, K. Greenwood and D. Wagner, "Analyzing Inter-Application Communication in Android".

[61] addictivetips, "What Is Zipalign In Android And How To Make Apps Ziapaligned [Complete Guide]," 09 November 2010. [Online]. Available: http://www.addictivetips.com/mobile/what-is-zipalign-in-android-and-how-it-works-complete-guide/. [Accessed 09 July 2012].

[62] Android Developers, "Traceview," Google, [Online]. Available: http://developer.android.com/tools/help/traceview.html. [Accessed 12 July 2012].

[63] Android Developers, "Publishing Checklist for Google Play," Google, [Online]. Available: http://developer.android.com/distribute/googleplay/publish/preparing.html. [Accessed 10 July 2012].

[64] Android Developers, "monkeyrunner," Google, [Online]. Available: http://developer.android.com/tools/help/monkeyrunner_concepts.html. [Accessed 09 July 2012].

[65] threeminds, "Mobile Application Testing: Process, Tools & Techniques," 02 May 2011. [Online]. Available: http://threeminds.organic.com/2011/05/mobile-application-testing-process-tools-techniques.html. [Accessed 09 July 2012].

[66] Oracle, "How to Write Doc Comments for the Javadoc Tool," Oracle, [Online]. Available: http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html. [Accessed 09 July 2012].

[67] die.net, "elf(5) - Linux man page," [Online]. Available: http://linux.die.net/man/5/elf. [Accessed 09 July 2012].

[68] TechTerms.com, "Android," 04 January 2010. [Online]. Available: http://www.techterms.com/definition/android. [Accessed 10 July 2012].

[69] Android Developers, "Activity," Google, [Online]. Available: http://developer.android.com/reference/android/app/Activity.html. [Accessed 10 July 2012].