Domotics On-The-Go 2.0

Master's Thesis, Spring 2011

Mario Muñoz Sanz

Department of Computer Science
Aalborg University
Selma Largerløfs Vej 300
9220 Aalborg
Phone 96 35 80 80
Fax 96 35 97 98
http://www.cs.aau.dk

Master's Thesis Report

Title:
Domotics On-The-Go 2.0

Period:
SSE4, Spring semester 2011

Author:
Mario Muñoz Sanz

Supervisor:
Arne Skow

Circulation: 4

Number of pages: 75

Number of Appendixes: 3

Finished on: 14th of June, 2011

Abstract:

Now days, the rapid progress of technology are becoming it increasingly affordable and they are now present in many places of our diary life. One example of that is known as *Home-automation* (also called *Domotics*). The computational power of home appliances is constantly increasing, and the wireless networks are already present in many homes. All this allows home sensors and devices to work in an autonomously and intelligent way, in order to make the home more comfortable, secure and able to save the maximum energy than ever was possible.
The HomePort project is a collaboration between Centre for Embedded Software Systems (CISS) at Aalborg university and Centre for Software Innovation (CSI) at Aarhus university. It is an on-going research project that focus on establishing interoperability between existing domotic devices from different vendors.
In this report I develop a mobile application which is able to manage a building equipped with a HomePort server. This version can interact remotely with the home devices and make the house behaves in the way the user wish according to the events that were happen on itself.

Preface

This report has been written by Mario Muñoz Sanz, guest student in the 4th semester of Software System Engineering in Aalborg University. This report is addressed to other students, supervisors and anyone else who might be interested in the subject. To read and understand the report correctly,. It is necessary to have basic knowledge about computer related terms.

The whole report is written in English. Abbreviations and acronyms will at first appearance be written in parenthesis, to avoid breaking the reading stream. References to sources are marked by [#], where # refers to the related literature in the bibliography at the end of the report.

It has been written in OpenOffice.org Writer and consists of six chapters and 3 appendixes which can be found in the last chapter of the report. The Eclipse IDE has been used to develop the source code of all the project. A web address is provided as support material including the source code of the whole Eclipse project:

http://dl.dropbox.com/u/3298901/Domotics%20On-The-Go%202.0.rar

Mario Muñoz Sanz

Contents

1 Introduction

Now a days the technology improves and evolves faster than ever. The numbers of transistors that can fit a square inch, or the amount of information capable of being stored by an electronic memory increases each year and hence electronic devices get smaller and smaller. This development is of great importance for sensor nodes since their actual size can determine if they can be applied in a specific context.

An example could be a natural habitat, where the miniaturization of the sensor nodes allow us to collect data in a non intrusive way. In this scenario tiny, almost invisible sensors become essential in terms of getting accurate data without interfering with the normal life of the animals or polluting the area.

Both sensor networks wired or wireless, are increasingly present in our homes under the name of Home Automation (also called domotics). The computational power of home appliances is increasing, and the adoption of wireless networks is widespread in many homes. This trend is growing and opens a potential to have otherwise isolated home appliances communicate and collaborate in a semi-intelligent way, hereby adding comfort, safety and resource optimization to our homes.

In 1966 Jim Sutherland, an engineer working for Westinghouse Electric, developed a home automation system called "ECHO IV"; this was a private project and never commercialized. With the invention of the micro-controller, the cost of electronic control fell rapidly. Remote and intelligent control technologies were adopted by the building services industry and appliance manufacturers worldwide.

The first convincing example of a home automation system was created in 1995 by Bill Gates - the founder of Microsoft. In 1995, this was a very expensive installation limited to very few people. A lot has changed since 1995 in terms of computing power, and size of computer devices. The technical evolution in the electronic industry together with an increasing demand from the consumer marked has lead to faster, smaller and cheaper computers. Today, we find many small computers seamlessly embedded into many of our common household appliances. Their programmable micro-controllers has typically replaced their mechanical counterpart as control devices, hereby providing more advanced, flflexible and user-friendly control systems.

The fact of be able to have lots of cheap computers embedded inside of many devices that we use during our daily life in our homes, suppose a perfect base for the home automation. However, due to this recently easy access to domotic, multiple communication standards used inside private homes have emerged. Six different communication systems are currently popular on the market, namely X10, C-bus, ZigBee, Z-wave, IO Home Control or KNX (this last one is approved as the only open standard for Home and Building Control, internationally, in Europe and China) [18]. It should also be noted, that different systems supporting these different communication standards are not able to collaborate under normal circumstances. The interoperability between the different protocols prevents optimal utilization of different vendor specific systems.

Various vendors have made different home automation systems which focus on one of the communication protocol. The firm Homesystem [13] provides a house-controller packet which is a complete system with a controller, a weather station and a terminal. Another provider is TELETASK domotics [24] which also sells a packet solution. Indigo [15] provides an application for ”any Mac-based X10 setup”. Unfortunately, neither of them supports multiple communication protocols.

A complete solution will include registration of electricity and heat consumed and will be able to control access, doors, windows, indoor climate and entertainment devices as well as administrating alarms reported.

The HomePort project is a collaboration between Center for Embedded Software Systems (CISS) at Aalborg university, Center for Software Innovation (CSI) and Aarhus university. Among the industry partners are Servodan, SeluxIT and Develco [6] [17]. The purpose of this project is to create solutions that can ease the administration of home automation equipment and ensure interoperability between different protocols and as a consequence achieve important energy and environmental advantages. [10]

1.1 Motivation

The way we perceive the concept of computing is constantly changing over time. In the past, computing was only associated with big data processing centers, and computers were enormous and extremely expensive machines which were only used by armies and a few investigation centers.

Today, this has changed in such a way that computers are part of our daily life. Almost everybody uses a computer during their work day and is very common to find one or more desktop or laptop computer in each house in every developed countries. Now we can find ourselves sitting in front of our personal desktop computers solving e.i work related tasks, gaming or with the entry of the internet communicating seamlessly with friends and family, making the shopping of the month or booking a holiday trip.

A special kind of computer, product of this evolution, are the smart-phones. A smart-phone is a mobile phone that offers more advanced computing ability and connectivity than a contemporary feature phone. Smart-phones and feature phones may be thought of as hand-held computers integrated with a mobile telephone, but while most feature phones are able to run applications based on platforms such as Java ME, a smart-phone allows the user to run and preemptively multitask applications that are native to the underlying hardware.

The first smart-phone was the IBM Simon. It was designed in 1992 and shown as a concept product. Besides being a mobile phone, it also contained other features such a calendar, an address book or e-mail. It had no physical buttons to dial with. Instead customers used a touchscreen to select telephone numbers with a finger [28]. Four years later, in 1996, the the Nokia Communicator line was lunched with the first Nokia smart-phone Nokia 9000. This distinctive palmtop computer style smart-phone was the result of a collaborative effort of an early successful and costly personal digital assistant (PDA) by Hewlett-Packard combined with Nokia's bestselling phone around that time, and early prototype models had the two devices fixed via a hinge. Then other Nokia devices were lunched with new features like an open operating system (Nokia 9210), color screen, Wifi connection or GPS antenna. And finally, in 1997 Ericsson released the concept phone GS88, the first device labeled as 'smartphone'. [31]

Four years later, Palm, Inc. introduced the first smart-phone to be deployed in widespread use in the United States, and Microsoft announced its Windows CE Pocket PC OS that would be offered as "Microsoft Windows Powered Smartphone 2002". RIM also released the first Blackberry one year after, which was the first smart-phone optimized for wireless email use and had achieved a total customer base of 32 million subscribers by December 2009. [8]

The market of the smart-phones was increasing in an extraordinary way until the launch of the first Apple smart-phone: the iPhone. This release revolutionized the the smart-phone market, doing this kind of devices really interesting for the non-professional users. It was the first mobile phone to use a multi-touch interface, and it featured a web browser that *Ars Technica* then described as "far superior" to anything offered by that of its competitors. [7]

Only one year after the iPhone release, which later will be its biggest competitor was released: the Android operating system. Android is an open source platform backed by Google, along with major hardware and software developers (such as Intel, HTC, ARM, Motorola and Samsung, to name a few), that form the Open Handset fotrAlliance [23]. The first phone to use Android was the HTC Dream. The software suite included on the phone consists of integration with Google's proprietary applications, such as Maps, Calendar, and Gmail, and a full HTML web browser.

Now a days, the popularity of mobile phones (1.596.802,4 thousands of units sold during 2010) [12] and overall, the advent of the smart-phones (sales grew 72 percent in 2010) [12] introduces a paradigm shift that again will change how we perceive computing. The smart-phone brings three fundamental changes that sets it apart from how we have performed computation in the past. The smart-phone is always turned on, it is always connected and it is always with you wherever you go [39]. In addition to this, the smart-phone is "smarter", that is: trough GPS, accelerometers and compass it is aware of its location, and how the device is being held.

Due to all these advantages that smart-phones are contributing to our daily life, added to the aforementioned trend of integrating low cost microcomputers in home appliances to facilitate domotics to enter common house holdings, make of these kind of devices the perfect allies to manage our home in a comfortable and easy way. There are several advantages gained by utilizing the smart-phones in extension to a domotic system.

Comfort & Mobility: The greatest user value is generated trough the comfort of mobility. The user experience is no longer limited to operate the domotic system trough fixed terminals or workstations. The domotic system can be monitored and controlled by a smart-phone from any location where internet connectivity is possible e.g. from an internet cafè in China (Wifi) or while riding the bus (3G).

Energy savings: Energy savings is a well known argument in favor for domotic systems but it is not a direct consequence of introducing the smart-phones as control devices. The smart-phone does however improve the utilization of resources within the home e.g. in domotic systems that control heating, ventilating, and air conditioning (HVAC). HVAC systems are rule based, using sensory data as input to its decision making. GPS information from the smart-phone will allow the domotic system to make more qualified decision making in terms of resource utilization. As the smart-phone is "conscious" of its geographical location trough GPS, the smart-phone can act as a sensor itself and supply the domotic system with information about the location of its user. This will allow the domotic system to lower temperatures when it can sense that no users are inside the house.

Security: By using the smart-phone as a control and monitoring device, we can be presented various information about our home while we are on-the-go. Events such as intrusion detection, accompanied with live video feeds will ensure that it will not come as a surprise to user if the home is compromised. When accessing the home the smart-phone can improve security when replacing the traditional key. Where access to the traditional home only requires a key, access using the smart-phone can be configured such that access will require both the physical device but also a code combination.

This makes the smart-phones a very interesting device for controlling domotics. By extending the HomePort system with a remote control and monitoring application, we can enhance the usability and provide additional comfort to the system. But this could increase the utility of the system even more than the remote increased the pleasure of using the television, because we can use the smart-phone like a part of the home sensors net, or to set up different profiles, which would make of the mobile phone an extension of the home brain that we can carry in our pocket. Suitable devices for such controller could contain a touch screen implying keyboard and mouse can be omitted. Tablet computers and smart-phones comes into mind. Both are widely used, and surveys show that 70% of mobile phones sold today are smart-phones [32]. This trend is positive as it entails potential owners of HomePort will have a smart-phone and thus be interested in using it as a remote.

In this report, I focus on enhance the existing smart-phone application Domotics On-The-Go, which acts as a basic remote monitor of the HomePort system, becoming it in an extension of the main home management system. I mean, currently Domotics On-The-Go is an application able to connect to a HomePort server using the secure protocols required, represent all the different networks managed by the server, including their locations and the actuators placed at each location. Moreover it can show the current state of every device (actuators). My work in this issue is focused on transform this basic application in one able to change the state of each actuators as a function of the measure and state of the sensors connected to the server, aswell other variables provided by the mobile phone. Besides those functions responsible for changing the state of the home devices, will be presented to the user as profiles that may create and configure to liking, so their setup will be an easy task.

1.2 Problem statement

HomePort is an ongoing research project that attempts to solve the non trivial problem of interoperability among heterogeneous components within domotic systems. In addition, the Domotics On-The-Go application (DOTG from now) try to add value to the HomePort project by extending it in an industrial/commercial application context. Moreover, DTOG attempts adapt existing technologies and embrace current market trends in order to provide a portable control interface which will contribute to future market adaptability and enhance the end-user experience by adding ease of use and comfort. Because of all these reasons I will try to provide greater functionality to DTOG to allow a simple and friendly way to manage an automatic home, and that will make of HomePort a more interesting project for the home-automation industry and also for the end-users.

With the purpose of achieve this goals I will examine the current market of mobile platforms to determine the platform with most market potential. Then, I will use this platform to develop an improvement of DTOG that will become it an extension of the control system of a domotic home equipped with a HomePort service. This application must conform to the application requirements set by the HomePort developers at CSI. This requirements are described in the Section 3.1

1.3 Report structure

In the following chapter I will talk about the background of this project. That is, an explication of HomePort and the different kinds of communication protocols used and needed. Also I will show what were the successes achieved by the last DOTG version. Below, in the chapter 3 – Analysis – I explain my analysis of various perspectives needed to design my application as well as any decision about which features develop. The forth chapter elaborates on the design choices of my application, followed by a chapter regarding my implementation and these troubles encountered. For ending, a conclusion will complete the report together with thoughts on future work.

2 Background

This chapter contains pre required information necessary for this project. In the first part I make a description of the most commonly used protocols within domotic in order to show the differences existing between each of them. Subsequently a detailed explanation of HomePort will be given, showing also a little example of the event flow through the HomePort system.

2.1 Communication protocols

This section describes the six protocols most often used within domotic systems. In the following, each of the protocols will be described individually based upon the following sources [36][37][38][25][16][18].

2.1.1 X-10

The X10 protocol utilizes existing power lines to send its messages. Because this protocol exploits the power lines, this protocol is cheap to apply, since no additional wiring is needed. However, the power lines also become a downside since the protocol are dependent on them, compared to wireless protocols like ZigBee and Z-wave. It should be noted that X10 bridge units exist so a message can be changed and transmitted wirelessly. Another downsides to the X10 protocol is the nature of the communication. In this protocol, only one command can be transmitted at the time. This means that first, a message to the intended receiver is transmitted. Secondly, the action is transmitted, where after the receiving device performs it. These signals can interleave or collide leading to commands that either cannot be interpreted, or in worst case trigger incorrect actions. Finally, the standard X10 power lines lack support for encryption.

Typical X10 appliances consist of either a X10 switch or some kind of device, which relays on switches or triggered events. In other words - devices with an interaction which does not requires a lot of message passing. Examples on such devices could be lamps, light dimmers, sensors and surveillance cameras.

A X10 message contains three parts; a house code, an unit code and an operation code. Since the house code part is 4 bit long, it has have a total of 16 unique house codes. They are denoted by letters A through P. The same applies for the unit code, although instead of letters, it is denoted by numbers 1 through 16. Lastly, a 4 bit operation message which denotes which of the 16 different commands the device should do. These messages are transmitted over the power line in 120 kHz bursts to separate them from the normal signal.

An example of a message that the protocol may send is “select code A3” followed by “turn on”, which says to unit “A3” to turn on its device. Also is possible that multiple units are addressed before send the command code, which means that the given command will be executed by each units called previously. The following figure shows the appearance that a message of this type could have:

A3	A15	K4	0010

Figure 1: A sequence of X10 messages broadcasted over the power line.

An example on X10 communication can be seen in Figure 1. In this message chain, the A3 device is demanded to pay attention, followed by the same request for the devices A15 and K4. At the end of the chain the operation command 0010 is send (On)[1]. When all these 3 devices will receive this command both of them will change their status to “on”.

2.1.2 C-Bus

C-Bus is a microprocessor-based control and management system for homes and buildings. This mean that each C-Bus device has its own in-built microprocessor and “intelligence”, allowing units to be individually programmed. This makes C-BUS communication very reliable and robust compared to X10. Also, as C-Bus uses point to multi-point communication, every device on a C-Bus Network issues and responds to commands directly from the Network, rather than requiring a central computer or controller. Each device is allocated a specific frame of time and each of them are initialized individually. Then the devices broadcast its status, synchronized by a self-generated system clock pulse. In this way, a large amount of data is allowed to be transmitted in a very small time frame, effectively and reliably on the network, leading to low processing overheads and low bandwidth requirements.

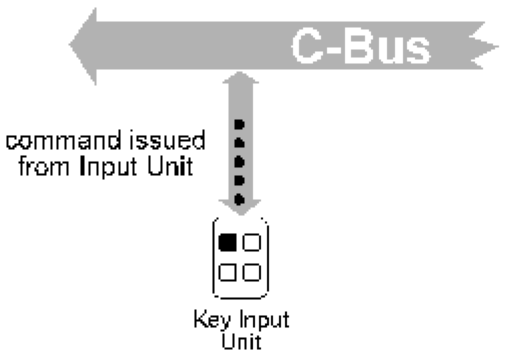
On the other hand, C-BUS is more expensive than X-10 due to is needed a communication wiring for the system. This wiring consist of an unshielded twisted pair cable, which is the C-BUS Network Bus. The C-Bus Network is electrically isolated from the mains power, and operates at safe extra low voltage levels (36 V dc). All input or output devices can be connected to any point of the C-BUS Network by a twisted pair cable, through which all the communication flows over. Anyway, a C-BUS gateway exist so both wire and wireless communication are available.

The topology of the network is free, which gives flexibility, since new devices can be added to a subsystem at any time without the need to reconfigure anything. During commissioning, the system is programmed so that specific commands trigger specific responses in one (or more) devices on the Network. At any time the commands can be re-programmed, and C-Bus units can also be added, removed or moved.

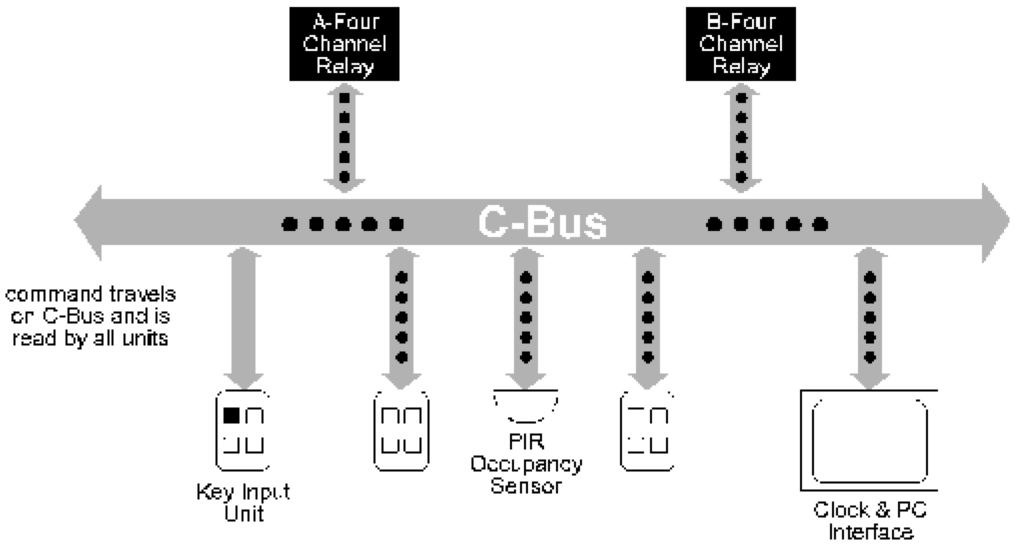
1 Table of operation commands can be seen on http://en.wikipedia.org/wiki/X10_(industry_standard)

The size of the network is limited to the number of IP addresses. Usually the network is divided in sub-networks of 100 C-BUS Units, which divide the system into manageable sections, simplifying the topology design, limiting possible propagation faults and aiding in troubleshooting.

Below, a simple example of how the C-BUS protocol works will be given[38]:



In the boardroom there is a C-BUS input switch that is programed with the address group name "Boardroom main lights". When this input switch is pressed, it send an ON command through the C-BUS to the "Boardroom main light" address group.



This signal will be received by every devices connected to the C-BUS Network, but only devices that are also programed as belonging to the "Boardroom main light" will interpret this command. The other devices that are not in this group will ignore this command.

2.1.3 Z-Wave

Z-wave is a wireless communication protocol developed by the danish firm Zensys [24]. To create a Z-wave network, a single controllable device with appertaining controller is needed. A simple network can then easily be extended by adding more devices / controllers to it. This is done by a process called pairing. By pressing a specific sequence of buttons on the controller, the device will be added and always recognized by the controller. This straight up approach requires little to no computer experience and thus allows inexperienced computer users to create a home automation system without dealing with the heavy technical part.

Within the domain of home automation Z-wave becomes a great alternative to X10 for users which either need the mobility of wireless communication or live in older houses lacking the neutral wire. Also, compared to the C-Bus and X10, the Z-wave protocol is created for wireless purposes thus have build-in security at its core. This comes in hand, if the user needs to utilize remote control / monitoring of the house.

Z-wave is a low-powered wireless protocol, which utilizes the 900 MHz ISM band, making it less susceptible to interferences otherwise generated by the more common and crowded 2.4 GHz band. The average reliable communication range is 100 feet in open areas, while the range will be reduced inside depending on various conditions such as building materials, placement etc.

The protocol is defined to transmit small messages, thus a low frequency radio is perfect with a bandwidth of either 9.6 or 40 kbits/s. In Europe, the 900 ISM band has a 1% duty cycle limitation. This mean that a Z-wave device can only transmit 1% of the time, whereafter the device resides in standby the rest of the time which reduce its power consumption significantly. The short radio range inside buildings does not hinder the effectiveness of the protocol, because the network is constructed using source routed mesh networking. This approach enables devices to relay messages through intermediate nodes, hereby extending the range of the network.

The Z-Wave protocol has 2 basic kinds of devices; controlling devices and slave nodes. A controller is a Z-Wave device that has a full routing table and is therefore able to communicate with all nodes in the Z-Wave network. The functionality available in a controller depends on when it entered the Z-Wave network. In case the controller is used to create a new Z-Wave network it automatically become the primary controller. The primary controller is the "master" controller in the Z-Wave network and there can only be one in each network. Only primary controllers have the capability to include/exclude nodes in the network and therefore always have the latest network topology. The rest of controllers added to the network using the primary controller (secondary controllers) don't have allowed to add or remove devices from the network.

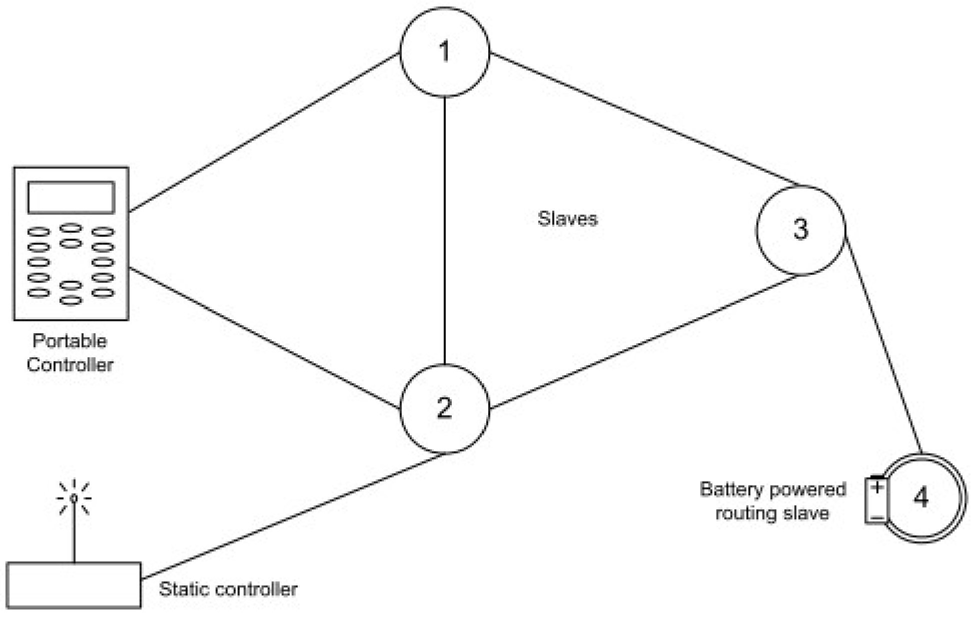


Figure 2: simple example of a Z-Wave network

The other kind of devices in a Z-Wave network are the slaves. A slave nodes are nodes that receives commands and performs an action based on the command. Slave nodes are unable to send information directly to other slaves or controllers unless they are requested to do so in a command. A light dimmer could be an example of an slave device.

2.1.4 ZigBee

ZigBee is the name of a specification of a set of high-level protocols for wireless communication to be used with low-powered radio broadcasting, based in the IEE 802.15.4 standard of wireless personal area networks (WPAN). It is aimed at applications that require secure communication with low data transmission rate and maximization of the life of their batteries, being home-automation a perfect example of this kind of applications.

As with Z-wave, ZigBee is applied within a vast of different domains ranging from industrial control, security and smoke detection to medical data collection and home automation.

ZigBee allow three network topologies: start topology, where the coordinator is placed in the center, tree topology, where the coordinator will be the root of the tree, and finally mesh topology, in which at least one of the nodes will have more than two links. This last topology is the one that make this technology more interesting. The mesh topology able manage a possible damage of a node from the way keeping the communication between the remaining nodes without any interruption, being this a coordinator task.

Three different ZegBee devices can be defined according to their role in the network:

- *ZigBee Coordinator*, ZC: This is the most complete device. There must be one into each network. Its task is to control the network and the path to be followed by devices to connect to each.
- *ZigBee Router*, ZR: They interconnect devices separated in the mesh topology, moreover that offer an application level for running user source.
- *ZigBee End Device*, ZED: It has the functionality needed to communicate with its father node (the coordinator or the router), but is not able to transmit information allocated to other devices. In this way, this type of nodes can be sleeping almost all the time.

The ZigBee networks have been designed to save the power of the "slave" nodes. According to this strategy, a "slave" node spends a lot of time in "sleep" mode, in such a way that only is "woken" for a fraction of second to prove that is "alive" into the device network in which is member. Because of that reason is why this kind of networks are able to save a great amount of energy.

2.1.5 IO Homecontrol

Is also a wireless protocol which handles radio communications at a frequency between 868 and 870 MHz. This enables each product at all times to select the frequency for transmitting the command being sent. Turn this enables each product to stay connected independently of radio interference. The protocol complies with the EN 300-220 standard for low-power radio applications.

As well as ZigBee, a IO Homecontrol system is upgradable. This means that it immediately recognizes new products and integrates them into the control unit automatically.

Regarding safety, the communication between the control unit and devices is secured with a unique, random 128-bit key encrypted message used to authenticate the origin of each command, as well as do banks and cash machines.

2.1.6 KNX

KNX is a standardized, OSI-based network communications protocol for intelligent buildings. KNX is the successor to, and convergence of, three previous standards: the European Home Systems Protocol (EHS), BatiBUS, and the European

Installation Bus (EIB or Instabus).

In this case, KNX is able to use several communication medias:

- Twisted pair wiring
- Power line networking (similar to X10)
- Radio
- Infrared
- Ethernet

Besides the great variety of medias that can be used (twisted pair medium is the most common used), KNX is designed to be independent of any particular hardware platform. This means that a KNX Device Network can be controlled by anything from an 8-bit micro-controller to a PC, according to the needs of a particular implementation.

2.2 HomePort

In the last section I have just done a description of six of the protocols most used within home-automation in order to illustrate the diversity of implementation and interoperability among them. Once been described the different communication types, I continue explaining the structure of the HomePort system. All the information in this section is based upon [17].

The core development of HomePort takes place at the Center for Software Innovation (CSI) in Soenderborg, Denmark where a HomePort prototype system called "Living Lab" is configured. The goal of the HomePort system is to create interoperability between multiple vendor specific communication protocols. This is done by creating a generic framework for domotics. To achieve common understanding between the different communication protocols, a common service layer is created to provide access to device functionality. This common service layer is controlled by service composites that define the interaction between devices, thus enabling a ZigBee switch to turn on a Z-wave lamp.

The HomePort system architecture illustrated in Figure 3, is divided into four different parts: *Composition Layer, Service Layer, Bridge Layer* and *Device Layer.*

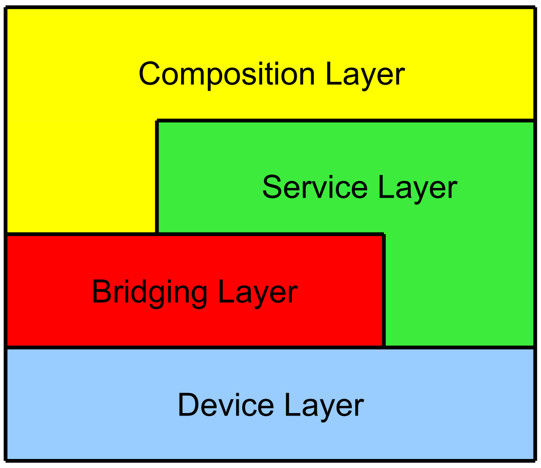


Figure 3: HomePort layered structure

Then I pass to describe each layer separately and at the end I summarize with an example.

2.2.1 Device Layer

The device layer consists of the physical home automation devices, e.g. lamps, switches, motion sensors etc. The end devices are grouped into subsystems each characterized by a vendor specific communication media and protocol. The communication within the subsystem is controlled by the specific hardware vendor, and the HomePort system makes no assumptions of the communication within a subsystem. Multiple subsystems can co-exist within a home. To take full advantage of the potential of home automation, these subsystem must be able to interact in a flflexible and intelligent manner [17].

Two device subsystems A and B are illustrated as a part of Figure 4.

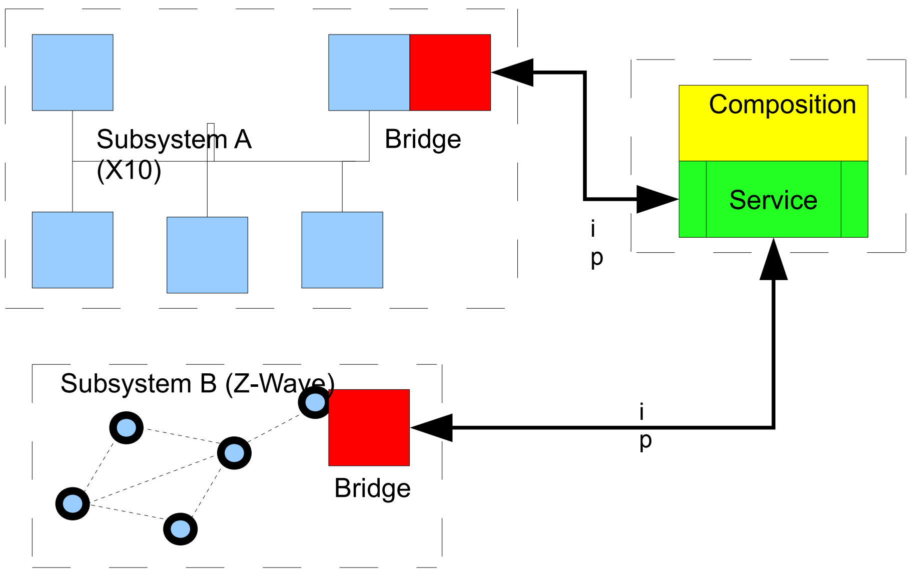


Figure 4: HomePort Illustration

2.2.2 Bridge layer

The Bridge layer acts as an abstraction on top of the subsystem, making the communication media and protocol of a specific subsystem transparent to the upper layers of the HomePort system. For the subsystems to be accessible to each other, at least one of the devices within a subsystem has to be able to communicate through some other protocol than the subsystem-protocol. The Bridge consists of two parts; a subsystem dependent part that ”speaks” the language of the subsystem, and an IP bridging part. The two parts communicate using a common network adapter interface (CNAI). By having this separation in the bridge, arbitrary subsystem protocols can be integrated in HomePort through a well defined interface, while sustaining the business model of the subsystem hardware vendors [17]. The vendors can expose some or all functionality of the sub-net trough CNAI, without opening their business domain to the competition. On behalf of the vendors the subsystem dependent part must be provided as a basic module to the bridge.

2.2.3 Service Layer

This layer presents the device functionality to the composition layer. Communication between Service and Composition layer is in a common, subsystem independent language. The Service gateway is connected to a number of bridges, one bridge per subsystem [17]. Each bridge registers the devices functionality of the subsystem to the Service gateway.

The accumulated services of the subsystems are stored in the Service gateway registry. The functionality of the subsystems are exposed through the HTTP based service protocol [26], using Representational State Transfer (REST) architecture [29].

This means that a lamp in a ZigBee subsystem is accessed in the same manner as a lamp in Z-wave subsystem. To access the state of a device in a subsystem, a HTTP GET is invoked and to alter the state of a device, a HTTP PUT is used instead.

Devices are described in XML. Listing 1 shows an XML descriptor for a light sensor. As displayed in line 1, each device has a name, id, ip, port, location, type and uid attribute to define it. As of now, the location attribute is optional, but the rest of the attributes are mandatory. The IP address and port number are especially important. Without those, HomePort cannot find the devices thus not control them. In line 3 a description of the device can be added, but is not mandatory. Finally, the services of the device is described. Each service has a name and id attribute which is needed to call the specific service on a device. In Listing 1 the light sensor only have one service, called light-sensed. The light sensor has an value url which is needed to access the data from the sensor. The information has the type int which is measured in the unit lux.

```
      < device   name ="Light sensor" id="lux - sensor" ip="192.168.1.225" port ="10002"
        location ="bedroom"
        type ="7A77042001" uid ="04">
      < deviceinfo />
      < service
            value_url =" http://www.cs.au.dk/dithus/services/light-sensor/"
             type ="int"
             id="sl23"
             unit ="lux"
             name ="light-sensed"/>
      </ device >
```

Listing 1: XML Device descriptor

2.2.4 Composition Layer

The composition layer contains the logic that controls how devices interact. Device interaction can be specified in either traditional programming languages or through the HomePort Control Logic Language (HCLL). HCLL is expressed through XML that describes the control logic. The main control structure of HCLL is Finite-State machines (FSM) [17]. The finite-state machine is a model of behavior consisting of a finite set of states (e.i On,Off), a finite set of actions (e.i. Press, dimm, etc.) and a transition function that describes the behavior of the system for all states, and actions [20]. A small example of a simple finite-state machine can be seen in Figure 5. In this example a lamp is in one of the states on or off. When the switch is pushed it changes to the other state. This process repeats each time the switch is pushed.

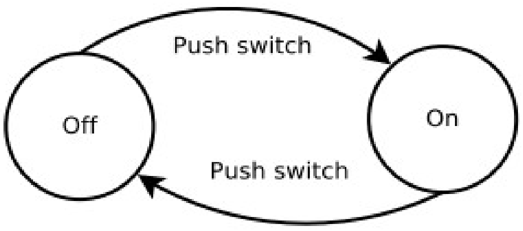


Figure 5: Example of small finite-state machine.

2.2.5 Example

To get a better overview of the functionality of the different layers, an example is presented in Figure 6. In the example, a HomePort system has two subsystems running in a home.

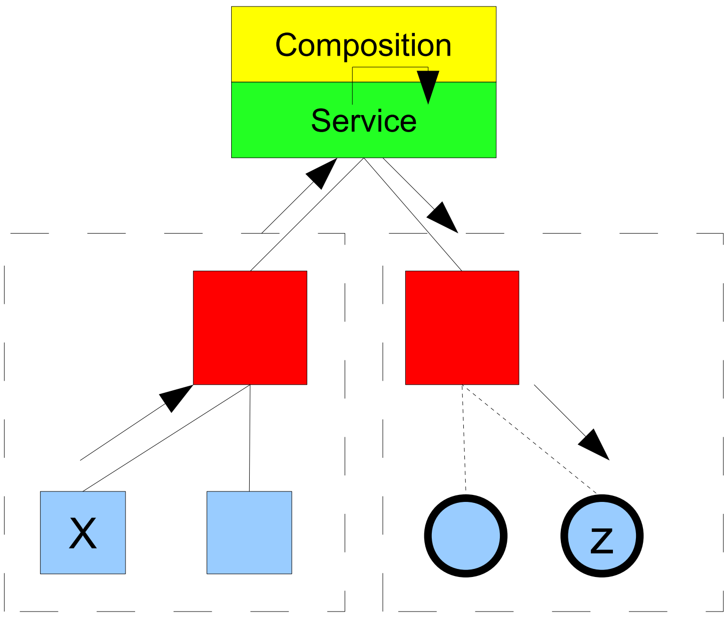


Figure 6: Event Flow

Subsystem A is using the X10 protocol and contains switch X and subsystem B is using Z-wave and contains lamp Z. When switch X is toggled an event is generated. The event travels within the subsystem X10 network, and is translated via CNAI to an IP broadcast within the bridge layer of system A. The service layer registers the event and identifies the source of the event by the source IP- and port address. Predefined composition layer logic associates the event of switch X, to toggle the light state of lamp Z. The service layer has the location URL of all devices in its registry, and to modify the state of lamp Z it invokes a HTTP PUT request to the URL of lamp Z containing the new state value. The HTTP request is handled by the bridge in subsystem B where it identifies the target device within the subsystem, and through CNAI forwards the request into the vendor specific subsystem protocol Z-wave, which subsequently toggles the state of the lamp.

3 Analysis

In this chapter I give more details on the core components needed to create my application. In the first place I describe the application requirements. Then, I continue by investigating the most suitable platform wherein implement my application, including a detailed description about the chosen platform. Below, in the section called Persistence, I explain how the data required by the application is stored and managed in the smartphone. After that, follow a section about the REST architecture which show how devices are manipulated in HomePort. Almost at the end, a section regarding issues with wireless control is given and finally a part where I explain what house management scenarios should be considered to implement and why.

3.1 Application requirements

Much of the requirements are originated from HomePort developments crew. Some of them are specific describe while other were more flexible. This allow me choose what approach I want give to my application. Taking all this in account, the purpose of the application is to connect to the HomePort server to manipulate the devices available in it. From this approach I have identify the following requirements:

Requirements

1. **Mobile platform.** The application should run in a mobile device but this is not specified by the HomePort developers so it is a decision that I can make, always keeping in main that old version of DOTG runs in Android platforms, which could be helpful.

2. **Security through SSL.** Since the application should connect wirelessly to the HomePort server security is needed. The HomePort developers required that I used secure socket layer (SSL) when communicating with their server. This would require certificates which they would provide.

3. **Visual data representation.** The devices available in the HomePort server must be represented in a graphic and intuitive way. That means that:
   (a) Get the devices by reading the network XML file allocated into the HomePort server.
   (b) Show the information in clear and orderly manner.
   (c) The navigation have to be intuitive, which means that devices should be easy to found.

4. **Device interaction.** The application must be able to interact with home

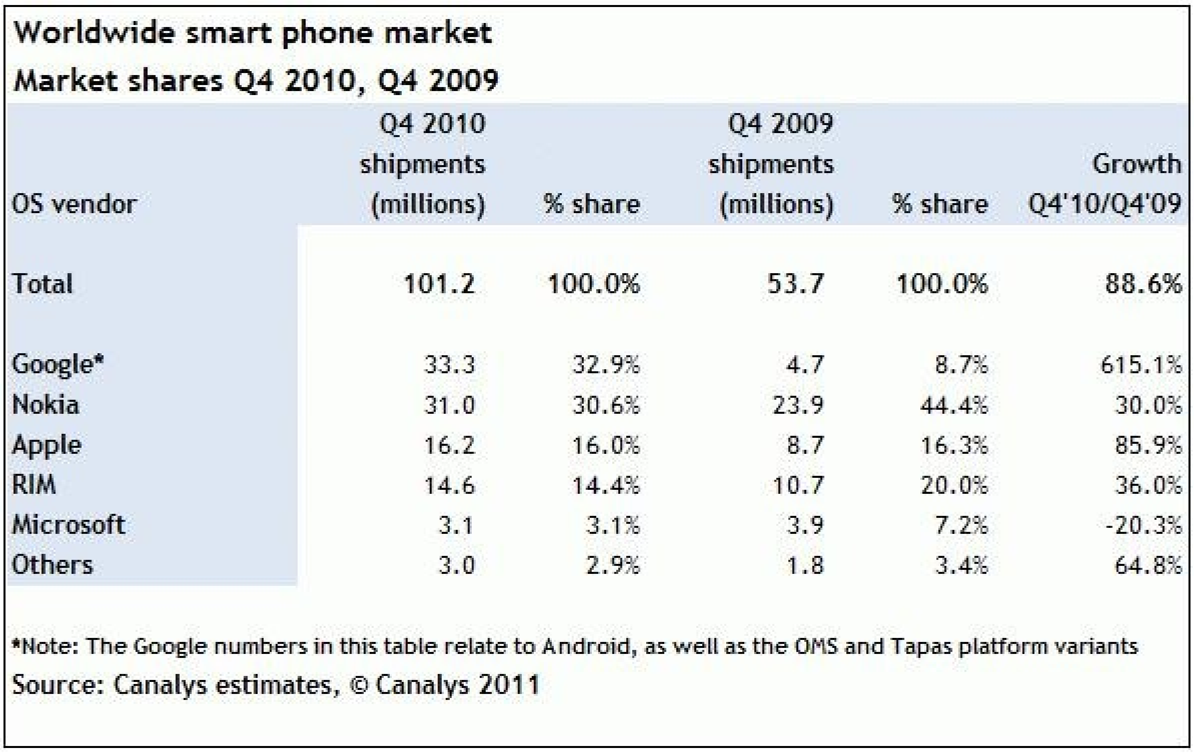
devices. That means being able to change the device status or read the sensor status, like turn off a lamp or know if a room is occupied.

5. To interact with the devices on the HomePort server, the application must use the REST architecture and use the HTTP GET and PUT message types.

3.2 Platform

In order to choose the most suitable platform it is necessary to take a look over the current most used types of operating systems for smartphones. The most important criteria that should be considered are number of users and ease of development. Because of that in this section I look to market shares as an indicator of number or potential users, and to technical specification to know which of them provides more facilities to developers.

As you can see in Figure 7, where is compared the sales of smartphone during the last quarter of years 2009 and 2010, order by operating system, Google's Android has become the leading platform. Shipments of Android-based smartphones reached 32.9 million, while devices running Nokia’s Symbian platform trailed slightly at 31.0 million worldwide. If we take in account that not all devices running Symbian can be considered as Smartphones, the following most popular platforms are Apple's iOS, despite having lost 0.3% of share, and RIM's OS (Blackberry OS), followed from far by Microsoft Windows Phone [33].



Worldwide smart phone market
Market shares Q4 2010, Q4 2009

OS vendor	Q4 2010 shipments (millions)	% share	Q4 2009 shipments (millions)	% share	Growth Q4'10/Q4'09
Total	101.2	100.0%	53.7	100.0%	88.6%
Google*	33.3	32.9%	4.7	8.7%	615.1%
Nokia	31.0	30.6%	23.9	44.4%	30.0%
Apple	16.2	16.0%	8.7	16.3%	85.9%
RIM	14.6	14.4%	10.7	20.0%	36.0%
Microsoft	3.1	3.1%	3.9	7.2%	-20.3%
Others	3.0	2.9%	1.8	3.4%	64.8%

*Note: The Google numbers in this table relate to Android, as well as the OMS and Tapas platform variants
Source: Canalys estimates, © Canalys 2011

Figure 7: Worldwide smartphone market

Then, I can conclude that the iPhone OS and Android are the two dominating OS's for smartphones worldwide. We therefore choose to focus on the strengths and weakness of these two platforms. One of the main reasons for the Android success is the relatively low price of the handset and the wide range of Android based handsets available. There are currently more than 111[2] smartphones based on Android, compared to iOS, that only just reached two handsets this including the recent addition of the iPhone 4 [21].

3.2.1 iOS versus Android.

One of the biggest advantages that Android has over the iPhone is the ease of development. Apple is a closed, proprietary environment, thus none of the inner workings of the iPhone are exposed for the developers [21]. Furthermore, in order to download the iPhone software development kit (SDK), one must first register as an Apple Developer Connection Subscribe [5]. Then agree on some strict license agreement in order to start the download of the SDK. It should also be noted that the iPhone SDK only run on Mac OS, which is determent for developers. As of October, 2010, Mac OS represented only 10% of the PC market share [14][11], eliminating 90% of potential developers. Contrary to the iPhone SDK, Google released Android's SDK for free in an environment that will run on any PC regardless of the OS. Also, they created an Android plug-in for Eclipse (free Java IDE). With Eclipse and the Android SDK developers are ready to write applications right away.

Once considered all this issues, I select the Android platform for my application because of the following reasons:

- Android is the fastest growing platform for smartphones.
- The platform is open source.
- Android application development is written entirely in Java, a well established platform independent language.
- The old version of DOTG is running in Android platforms therefore it will be helpful and easy to reuse some parts.

3.2.2 Android

The Android OS was a product of a company called Android Inc. which in 2005 was purchased by Google Inc. to make from this operating system their main bet for mobile devices market.

Android applications are usually developed in the Java language using the Android Software Development Kit, but other development tools are available, including a Native Development Kit for applications or extensions in C or C++, and

2 http://www.andro-phones.com/2011-android-phones.php

Google App Inventor, a visual environment for novice programmers. That is because there is no java byte code or Java virtual machine running on the Android device. Java is compiled into Dalvik Executable, and is interpreted by the Dalvik Virtual Machine. Dalvik is targeted at the mobile devices, and is optimized for devices with limited memory and CPU. The virtual machine allows multiple instances of itself to run at once and takes advantage of the underlying operating system (Linux) for process isolation and security [27].

3.2.2.1 Architecture

The Android architecture consist of four different layers which is illustrated in Figure 8[3]. In the bottom layer a Linux kernel version 2.6 runs the core system services [27]. This entails process and memory management, networking, driver models and security. This layer also serves as an abstraction layer between the underlying hardware and the rest of the software stack [9].

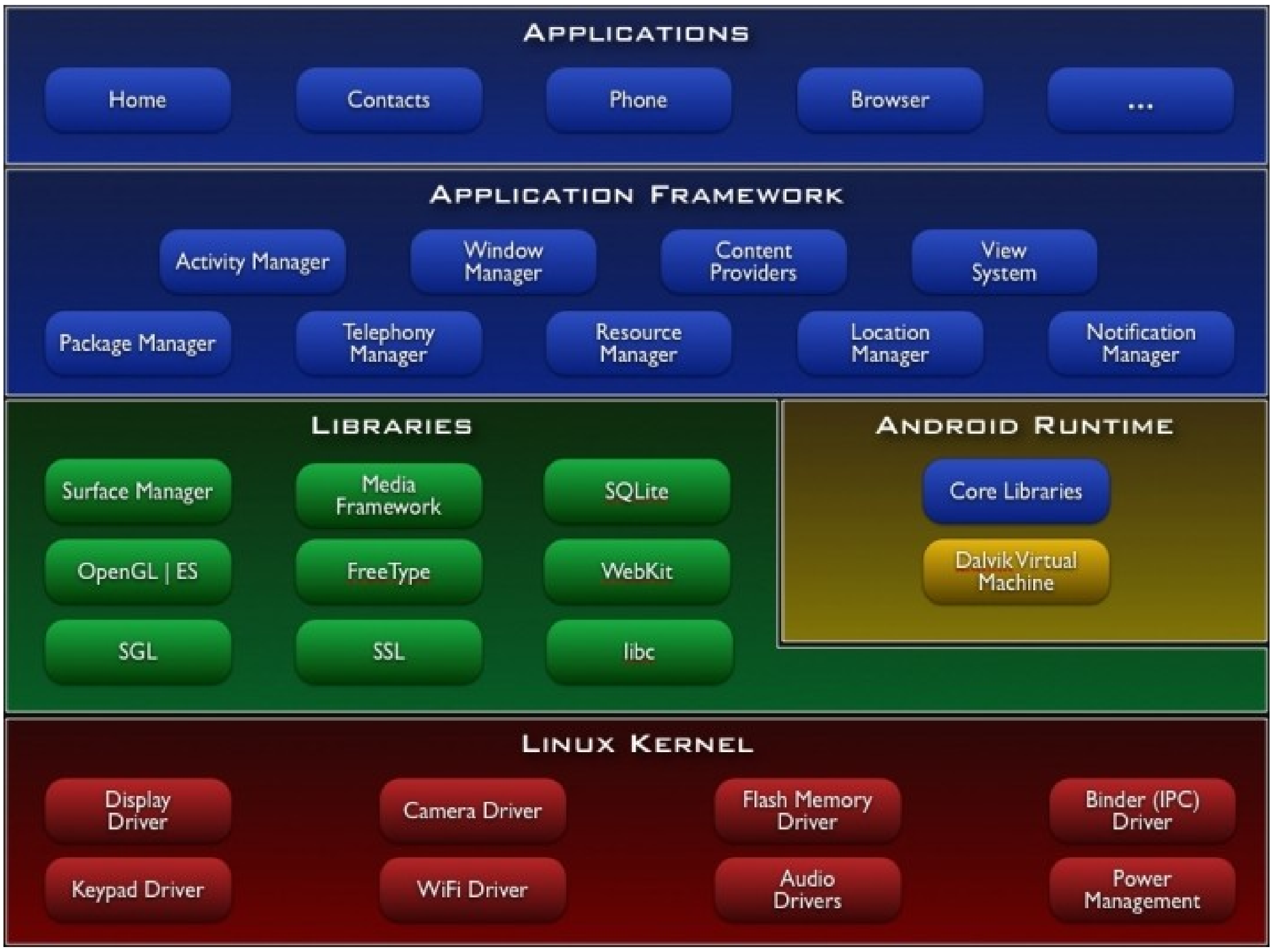


Figure 8: The major components of the Android operating system

3 Picture from http://developer.android.com/guide/basics/what-is-android.html

On top of the Linux core reside the native libraries. All libraries are written in C or C++ and compiled for the particular hardware architecture used by the phone and pre installed by the phone vendor. All of these libraries are written with focus on fast and simple execution, so low energy usage is assured.

Above the libraries is the application framework. This layer provides high-end building blocks to the developer. It should be noted that the Activity Manager, which plays a key role in Android system resides in this level [9]. The importance of the Activity Manager is discussed in the following subsection. Finally, the highest level of the Android architecture is the applications. Most Android users will only see this level.

3.2.2.2 Activity life cycle

An instance of the Activity class is an object that represent a single object or action that a user can do e.g. choose menu item, enter settings or read ”about” dialogue. In android Activities are often associated with a graphical View that allows user interaction. One Activity is actively running at a time, and multiple Activities are organized in an activity stack structure. Only the top Activity element of its stack is running actively [9][27]. Activities can be in different states according to their visibility and focus. The life-cycle of an Activity is illustrated in Figure 9[4].

In Figure 9 the states of the activities are represented as a kind of colored ovals, and the gray rectangles indicate the name of the methods that are invoked upon a state change. In the white rectangles are shown what user or OS action causes the invocation of these methods.

Because of the scare resources available on a mobile phone, and the continuous urge to keep battery usage as low as possible, management of concurrently running application is needed.

When an application is started Android brings it to the foreground. From that application, the user might invoke another application and so forth. All these screens are stored in the application stack by the systems Activity Manager [27]. With this stack, the user can backtrack the different screens - like the history in a web browser. In Android, each screen is represented by an Activity class and has its own life cycle.

This means that Linux encapsulate each application as one or more activities. However, the life cycle of the activities are not connected to the encapsulating process, which can be seen as a disposable container for activities. When a new application is invoked and presented in the foreground, the Linux process that was running the previous Activity is killed. The states of the previous Activity are stored and the new application is opened. On a later time, the user might return to the stored activity, where it will be restarted and the states will be loaded. This behavior

4 Picture from http://developer.android.com/images/activity_lifecycle.png

forces a special design choice which will be discussed in the following subsection.

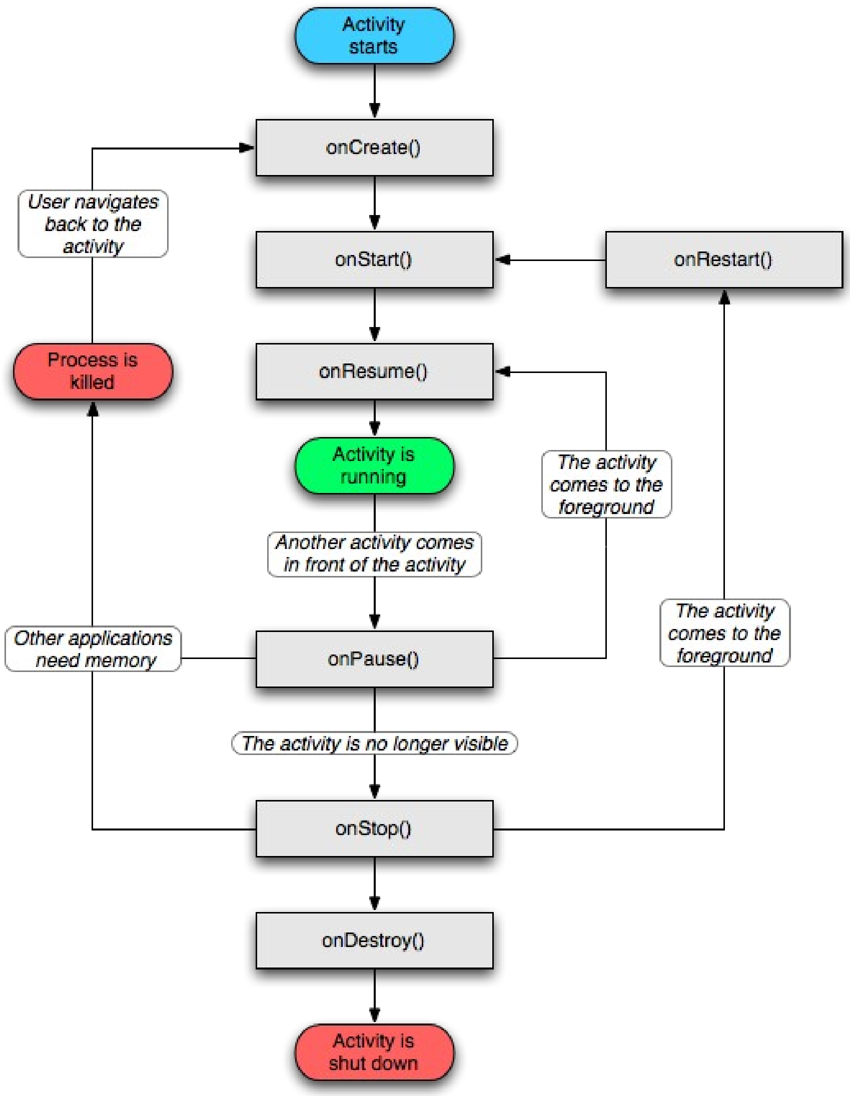

Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
The activity comes to the foreground
Other applications need memory
onPause()
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

Figure 9: Android Activity life-cycle

3.2.2.3 Services

A *service* is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity. Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it.

Like activities, services have a cycle life, which will depend on how it is used. It can be started and allowed to run until someone stops it or it stops itself, or it can be operated programmatically using an interface that it defines and exports. Clients establish a connection to the Service object and use that connection to call into the service.

The diagram in Figure 10[5] illustrates the callback methods for a service. Although it separates services that are created by the first method from those created by the second one, keep in mind that any service, no matter how it's started, can potentially allow clients to bind to it, so any service may receive onBind() and onUnbind() calls.

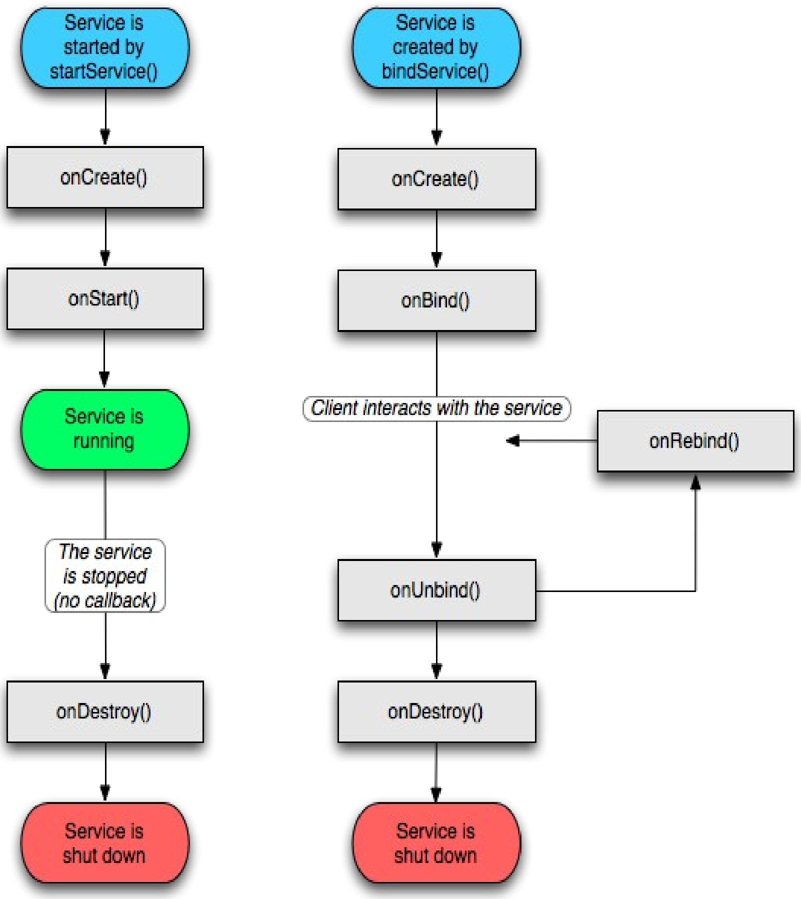

Service is started by startService()
onCreate()
onStart()
Service is running
The service is stopped (no callback)
onDestroy()
Service is shut down
Service is created by bindService()
onCreate()
onBind()
Client interacts with the service
onRebind()
onUnbind()
onDestroy()
Service is shut down

Figure 10: Cycle life of services in Android

3.2.2.4 Android design influences

The way that we develop and design the Android applications is affected by how Android handles activities. In traditional Java programing we often use the Singleton design pattern when some information or functionality is accessed across multiple classes. However, we can not use the Singleton design pattern when we are developing on Android platform. Objects does not persist by default. In fact, the system may choose an applications that appear being running to kill its underlying process and later restart it again. This leads to the data persistence is required, so it

5 Picture from http://www.cinterviews.com/2009/11/service-lifecycle-in-android.html

gets store as a state. In Android we can implement the data as a global service to do it. The data must be stored as a state because the service will be available to all applications. The other option is to store the data using an SQLite database or some other way of data storage.

3.2.2.5 Persistence

The Android platform provides two main tools to storage and retrieval of structured data:

- SQLite Data Bases
- Content Providers

SQLite covers every task related to own data storage of the application. The second tool, the Content Providers, ease the task of make visible these data to other applications and also allow our application retrieve data published by a third application [3].

The SQLite is a very popular database engine now a days because of provide features so interesting as its little size, not to need a server, require little configuration, to be transactional and, of course, to be open source.

Unlike the client-server database management systems, the SQLite engine is not an independent process with which the main program communicates. Instead, the SQLite library is linked with the program becoming an integrated part thereof. The main program uses the functionality of SQLite through simple calls to subroutines and functions. That reduces the latency in the access to the database, due to function calls are more efficient than communication between process. The entire database (definitions, tables, indexes and the data itself), are stored as a single standard file in the host machine. This simple design is achieved by locking the database whole file at the beginning of every transaction [4].

3.3 REST architecture

The Representational State Transfer (REST) is a style of software architecture for distributed hypermedia [19]. According to this architecture, everything can be perceived as resources. Those resources are requested by the clients by transmitting the a GET message to some server. An important characteristic of this architecture is that any state is stored during the communication, so, all the context information needed to respond to the request send by the client with the right resource, must be implicit in the request itself.

According to HomePort specifications, a gate device exposes its services through the link which connect it to the bridge devices that has each available subsystem.

Every device are modeled as a resource, so they all can be invoked using the HTTP methods GET and PUT [19]. Having a REST based service oriented architecture ensures that device functionalities can be composed across subsystem and ownership domains. The interaction with a device at the "service-layer" is transparent, regardless of the wireless network to it is connected [17]. The implementation can be found in almost all platforms because the HTTP is a light-weight protocol. Moreover, it is possible to use any standard browser to address the resources, which means any third program is not needed.

In the following section I address issues of security originated by the fact of extend the HomePort system with a smartphone application.

3.4 Security

The fact of allow every devices and sensors of a house be able of being handled from a remote system like a mobile phone or a web browser implies a lot of risks. A security breach may involve high-cost consequences, so, in order to not compromise domotic system some security measures must be taken. Of course these security measures are required in all layers of the HomePort architecture, but I will address issues related to connecting the system from a mobile device.

There are three security requirements which are needed consider when connecting a remote device: *user authentication, message confidentiality* and *message integrity* .

- **User Authentication:** is required to ensure that we know in every moment unequivocally the parts engaged in the communication process.
- **Message Confidentiality:** warrant that if the sender "A" sends a message for the receiver "B", any third listener “C” hided in any midpoint of path, is able to read these message and know some information about the house state. Without confidentiality, foreign agents may gather vital information such as know if the alarm system is enabled or if some exit door is unlocked, clearly endangering the security of the house and its inhabitants.
- **Message Integrity:** as well as can not be allow anyone alien access to a message information, also can not allow any third agent modifies the message content. Furthermore, if this happens, the correct receiver must be able to detect that the message has been altered.

The solution to these three issues that HomePort crew proposes is an encryption protocol called Secure Socket Layer (SSL).

Secure Socket Layer

This protocol is very well known within the bank and online payment world, where the security is paramount. In 1996, Netscape published the SSL version 3.0, which was developed by them, and later it was accepted as an standard; in fact, Visa, MasterCard, American Express and many others of the main financial institutions have approved SSL to the internet commerce [34][22]. The SSL connection provides the elements of authentication, message confidentiality and message integrity.

To access to the HomePort Living Lab is necessary use a secure socket layer connection. In the following example I will illustrate the principles behind SSL communication.

In Figure 11 the client is going to establish an SSL connection to the server. First of all, both parts must agree, at the beginning of the connection establishment, on the version of SSL protocol and the cipher encryption they want to use [30].

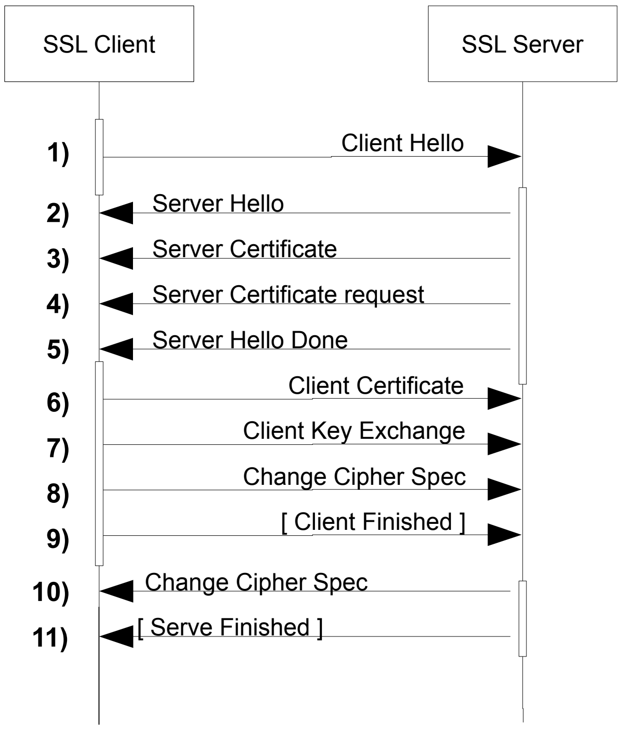


Figure 11: SSL session handshake

1. The client sends to the server a hello message to start the communication. This message contains the supported *versions* of SSL, the *cipher* and a random number which will be used step number 7.
2. The response returned by the server will be a “Server Hello” which will contain the SSL version chosen and the cipher for this SSL session.
3. The server sends to the client its certificate along with the public key.
4. The server needs a certificate from the client to authenticate it so, it will send a “Client Certificate Request” to the client saying what certificate types are

sported, and name names of acceptable Certification Authorities (Cas).

5. Then, the server sends a “Server Hello done” which means that all request are finished.
6. Below, the required certificate is send by the client to the server.
7. Using the random value received from the server, and the clients own random value, itself will compute a pre-master secret. After that, the client sends a “ClientKeyExchange” message to the server in which the pre-master secret is encrypted using the server's public key. The client and server both compute a master secret from the pre-master secret locally. The session key used for symmetric encryption is derived from this master key. We will have ensured that the server is the only one which can decrypt messages from the client if this server is able to decrypt the “ClientKeyExchange” message and proceed with the protocol negotiation.
8. The client confirm the following messages will be encrypted by sending a “Change Cipher Spec” message.
9. The ”Client Finished” message is encrypted with the session key, containing also a hash of the entire negotiation.
10. As well as the client did, the server sends a “Server Cipher Spec” message which indicate the following messages will be encrypted.
11. Finally, the server sends a “Server Finished” message encrypted and containing a hash of the negotiation until this point. If the client can decrypt this message and validate the hash, the SSL handshake was successful.

3.5 Device access

I must make some design choice between the existing alternatives about how are the home devices going to be accessed. In the composition logic layer the automation home device interaction is defined trough HCLL. An automated scenario may be that at a preselected room temperature and with the heaters turned of, the windows could be opened, if it is not raining outside, in order to refrigerate the room. Or perhaps, a different "automated scenario" could be to close all exit doors and the ground floor windows, when nobody is detected inside the house by the presence sensors.

In the last two sample scenarios could appear some collateral effects because of direct devices access. If a remote application is able to change a device state, it could alter the correct behavior of the defined composition logic within the home. That is, if the application unlocks an exit door, the house could be exposed to robberies if no-one is in the house. Furthermore, some house furniture may be damaged if the application opens a door when it is raining.

A different approach could be not to allow the remote application to access directly to any actuator, but only allow it to interact with non-automated sensor (e.i regular switches and dimmers) trough generating system events. Doing this, the application remains always isolated in a secure zone where it won't be able to cause any undesired behavior unhanded by the composition layer defined in the connected HomePort system.

Generate a list that included devices which devices can be accessed and handled by the remote application could be a third approach. Nevertheless, HomePort does not implement any access policy although it is considered as a future work [17]. Moreover, the composition of this list would be a tedious work for almost every end users and it could lead to errors

3.6 Scenarios considered

As I have already say, the number of devices and sensor capable of being integrated in a smart-house is very big, and it will continue increasing with the improvement of the technology and the lower prices. Because of this I have selected a bunch of the most common devices used within this kind of houses, along with the most important and needed sensors, in order to describe the possibilities that there are to interact with them.

Consequently, in this section I explain all the different scenarios that I have considered to implement in this application and the different reasons to do it. Anyway it will depend on household equipment and sensors available.

The following table shows in a graphical way the relationship between the magnitude that are enabled to be measured by the home sensors and all the type of devices which the system can interact with.

Magnitude/Actuators	*Dimmers*	*Blinds*	*Switches*	*Outdoor Switches*	*Doors*	*Windows*	*Heaters*
Wind					X	X	
Rain					X	X	
Presence	X		X				X
Outdoor Light				X			
Indoor Light	X	X	X				
Temperature						X	X

Table 1: Scenarios

3.6.1 When it is windy

In order to avoid little accident inside the house, like door slam or falling objects due to the air currents, the application should allow to close all windows and exit doors in the house when the day is windy. Moreover, the wind speed which it should close windows and doors should be chosen by the user.

To achieve this it will be necessary at least one outdoor wind sensor. Due to this isn't very common it is possible that this scenario may not be in every building. Of course it will necessary operable doors and windows, but I will assume it from now on.

3.6.2 When it is rainy

As in the previous scenario, it will be necessary that the application close all exit doors and windows when it is raining, to prevent the water from entering the house and can damage the floor, walls, etc.

In this case also will be necessary to have a rain sensor which indicate when the this scenario is happening.

3.6.3 Depending on the presence

Now a days movement and presence sensors are some of the most common and cheapest in any building. To exploit this fact, this scenario takes in account the possibility of use its signal in order to turn off all the lights of the house, and turn down the heaters temperature when nobody is detected inside any room of the house. The main goal of thought in this function is to save energy, that as I have already say is one of the main purpose of the home automation.

An other behavior, supported by the phone location system, could be to block the exit doors and windows, and turn on the alarm system when the last person is out of the house and going away. But this competency will have to be considered to future works.

3.6.4 Depending on the outdoor light

A very common skill of many houses is to turn on the outdoor street lights when the night arrives and turn it off when sunrise happens. The easiest way to develop this behavior is basing on the hour in the system. However, in this scenario, I have considered the possibility of manage the outdoor lights using any light sensor located outside of the house, which is also very usual to find in many buildings. Obviously the minimum light level needed to switch on the street lights should be selected by the user.

3.6.5 Indoor light level management

In this scenario I have taken account, in addition of the purpose of save energy mentioned in the scenario 3.6.3, other of the main goals of the domotic which is the comfort of the inhabitants. The aim is to maintain the light level inside the house combining the sun light coming through the windows and the light from the lamps in each room.

To make available this function it will be necessary at least to have light dimmers in all rooms, because without them there will not be any way to control the light level. Also, there can be operable blinds in some rooms that allow to control the amount of light that is coming in through the windows.

The idea is to combine the use of the light dimmers and the blinds to get inside the house the light level wanted by the user, by using as little as possible the electric energy, and keeping this level without affecting the time of day and changes in the outside light level.

3.6.6 Temperature management

The purpose of this scenario is simple. The intention is to make the application able to control the indoor temperature. It could look too simple using only the thermostat of the house, but the idea also to use the windows to manage the temperature, opening they when we want to refrigerate the house and lower the house. Of course it will be necessary a temperature sensor outside the house to indicate if the temperature outside is lower and it is useful opening the windows.

4 Design

In this chapter I describe the main different design decisions made to accommodate those requirements and other analysis issues addressed in the last chapter. As we can see in Figure 12, the software structure is organized in three layers. The first one represent the GUI whose operation is detailed in section 4.3. Among other things, we can find the profile settings interface, which is one of the main new features. Moreover, in the logic layer we can see the Profile manager and the automation service, both of them new features in this version too, and explained detailed in sections 4.6 and 4.7 respectively. Finally, we have the data layer in which is included the data base, hosted on the mobile phone and where the profile settings will be saved. In this layer is also included the remote HomePort server that is the interface to which the application has to interact.

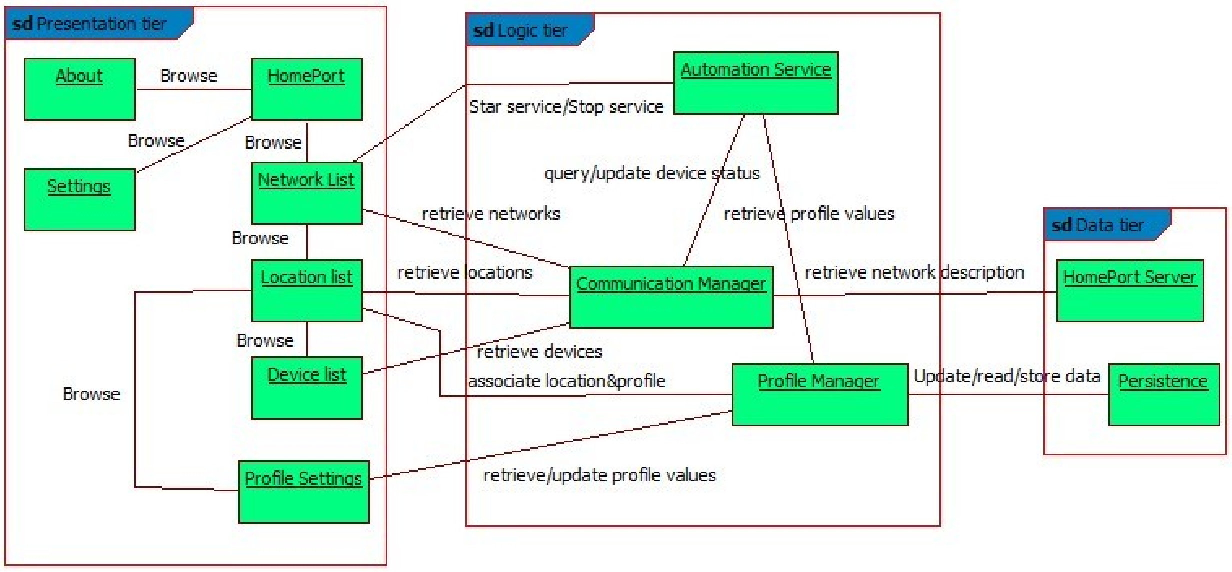


Figure 12: Software structure

Furthermore, a short description of the relationships between each component is given in the Figure 12. These relationships are also addressed in the following sections.

4.1 Device list parsing

All the information needed by the application, which will be the base for all the functionality is obtained from the network XML file located in the HomePort server. This file will say to the application what networks are available in that server and how are distributed the devices inside each network. The network XML also provide all the information about how to access to each device, what kind of device it is, etc. An example of this file can be seen in the Appendix A: Network XML description

Now a day, in almost every common development platforms there are several ways to read and write data in XML format. The most widespread are SAX (*Simple API for XML*) and DOM (*Document Object Model*). Subsequently, other ways have been appearing, among which stand out StAX (S*treaming API for XML*). Obviously, Android also include this three main models for the XML handling, being in the last case an analog version (*XmlPull*) [35]. Of course, all of them allow to do the same things although not in the same way.

The SAX parser is only a simple API which works as an stream parser with an event-driven API. The advantage of this approach is that use the memory in an efficient way. It will not be required the maximum depth of the XML document in memory in almost all cases. Following almost the same working method, *XmlPull* also is based on defining actions to be done for each events generated during the sequenced reading of the document. While with SAX we don't have the control the reading once started, with *XmlPull* we can demand the reading of the next XML element, answering with the appropriate action.

In the other hand, DOM parser stores the whole XML file in the memory, while it builds the tree structure. Even though this approach uses much more memory resources, it provides an implicit device object representation and elegant data traversal, which may be a big advantage during the implementation task.

In the last version of DOTG was used the DOM method because the benefits were more than the drawbacks of the additional memory. It should be noticed that the memory size is becoming less of a problem due to the increase of the capacities, and also even if we work with a huge network full of devices, the storage of the network XML document would be much less than a simple video game. Because of all this, and in order to be able to reuse some code I also used this parse method.

Once the network XML document is parsed, each device will be classified according to its network-tag, which indicates what house it belongs (we may manage our office and our house with the same HomePort server). Then, all devices will be grouped inside each network according to its location-tag. That means, all devices located in the kitchen will be found under the category Kitchen. Nevertheless, we have to bear in mind that the location-tag is not mandatory, therefore all devices without this tag will be grouped as Unknown.

4.2 Device interaction

When the XML document was parsed and it was made the list of the devices available in every location in each network, the next work is to determinate how to interact with this devices. In the previous version of DOTG we only could interact with a single device at a one time. This time, we will be able to access to several devices at the same time to get an automatic behavior, and to allow to change the state of some devices depending on the state of some others. Of course it is also possible access to an individual device an the functionality will depend on the type of device accessed.

To get all this, a couple of issues must be considered. First, how to set up the automatic behaviors, and then, how to display this configurations and the individual devices in an intuitive and friendly way.

4.2.1 Device access

As a main need, the application have to be able to interact with the different available devices. That is possible thanks to the attribute that each device has in its declaration in the network XML, called *value_url* (See Appendix A). Those URLs are used by HomePort to communicate with the different devices. That means we need the following things if we want to know the status of a concrete device:

1. Find the *value_url* into the description of the selected device.
2. Parse the value of the attribute to HomePort.

Having this value, we can send a HTTP GET-request to the HomePort server, and it will reply with the status of the device in question. Below I show a generic example of a *value_url:*

```
Http://<ip-address>:<port>/services/<id>/<name>
```

In the last version, each time a list of devices was created HomePort was asked about the status of each device, so the list was updated. This means that it may be inconsistencies between the status recorded in the application and the real state of a device. For example, if the user of the smartphone create the list and gets the status of a given device, and later a resident interacts physically with the device. In this case the device status known by the smartphone user would be wrong.

To fix this problem, in this version the automatic behavior will be the only one who need to know frequently the device status. This will be a task of an android service responsible for the automation which will check the device status before request any change (See section 4.7).

4.3 User interface

All the interactions with the house devices are done through PUT and GET request, but this must be something transparent for the application user. Moreover, it should be able to do it in an intuitive way. To get this purposes, I have devised some sketches and I have reused some others from the older version of DOTG. I will immediately turn to describe it:

4.3.1 Main screen

In the main screen there should be at least the two indispensable buttons **exit** and **about**. As they don't have the most important functionality, they will be placed on the bottom of the screen. Besides these functions, the main screen have to allow to connect to the HomePort server, and also have to give us way to the place were the user can introduce all the information needed to establish that connection. Because of being the most important functionality, the connect button is placed in the top of the screen. Then, just below, it is located the settings button. In the following section will be given a detailed description of the screen shown after click this two buttons. In the Figure 13 you can see a sketch of the main screen.

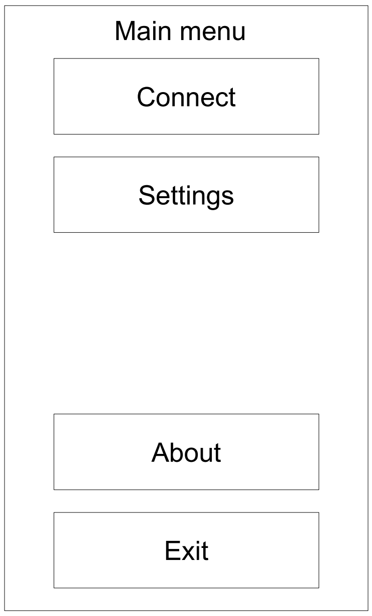


Figure 13: A sketch of the main screen

4.3.2 Setting screen

To connect to the HomePort server the application require some information. In this screen the user will introduce all the needed data. In first place, on the top, there would be a text field which the user will fill with the server **URL**. Also a **username** may be just below. This user name will be used to distinguish the type of user that is connected to the house. Having this information the application could use different settings for a parent or a teenager. I mean, if the user is arriving at home, the application may turn on the coffee machine, if he was logged as Dad. However, if the user was logged as a child, the application could start to warm his room instead of turn on the coffee machine. If there is a username field, a **password** field shouldn't miss. This is because the application has to be as secure as possible. And finally, if I wish to support several **languages**, there should be a drop down menu where the user will be able to choice his preferred. In the Figure 14 you can see a sketch of the settings screen.

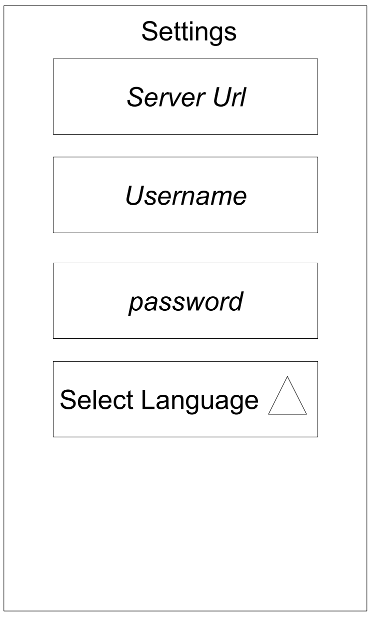


Figure 14: A sketch of the settings screen

4.3.3 Connect screen

Once the connection to the HomePort server was established, the application must get the description of the networks and parse it. After that, the application should show present in a list all the networks available (See Figure 15). Then, the user can select a network. In that moment, the application will list all the locations in this network like a list, in the same way as it did with the networks. Other option could be to show in a drop list under the network representation. Despite that, I decided to take the first option because gives the user the feeling of really getting into the network. At this point the user have two different actions possible. The first one is to click on a location and the application will give him a list of the devices available in this location. This list will show the user a representative icon of the type of device, the name of the device in question and a short description (See Figure 15).

The second possible action is to select an option from the available menu. This menu gives the user the option to apply a configuration profile to the previously selected network, or the option to set up an existing or new profile (See Figure 16).

When the user is on the screen of devices, he can click on any of them to see their status and interact with them. The application shows a pop-up box with a bar which will represent the state of the device. In the Figure 17 I give two example of this pop-up boxes. The first one is for scalable devices (Thermostat, dimmers, etc), and the second one is for two-state devices (Switches, TV, etc).

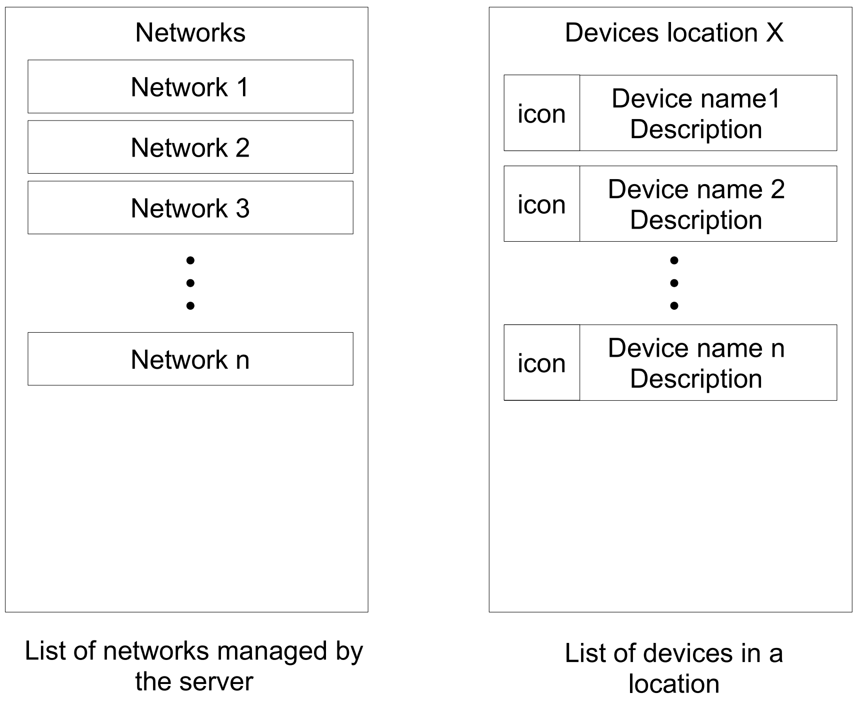

Networks
Network 1
Network 2
Network 3
Network n
Devices location X
icon
Device name1 Description
icon
Device name 2 Description
icon
Device name n Description
List of networks managed by the server
List of devices in a location

Figure 15: Sketch of the networks list, and the available devices list in a location

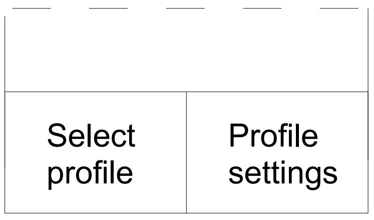


Figure 16: Sketch of the context menu

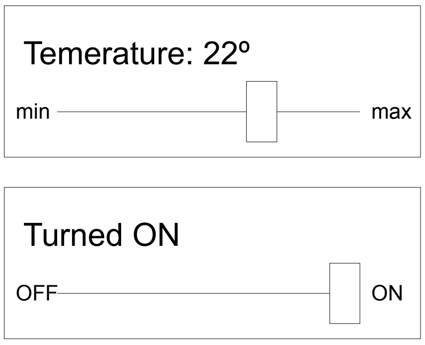


Figure 17: Pop-up boxes with the device status and controller

4.3.4 Profile screens

In order to make of DOTG a real smart application, it is necessary to allow the user to set up his own configuration profiles. The application will use these profiles to manage the house, keeping always the desired measurements by the user, and responding to the possible events according to the user wishes. Bearing this in mind, we need two screens related to the profile handling. The first one is a screen where the user can choose a profile to be applied in the selected network. But instead of a screen, I propose a pop-up menu where the user will be able to watch what profile is being currently applied. In this menu it is also possible choose a different profile between the existing. In the Figure 18 I give a draft of this pup-up menu.

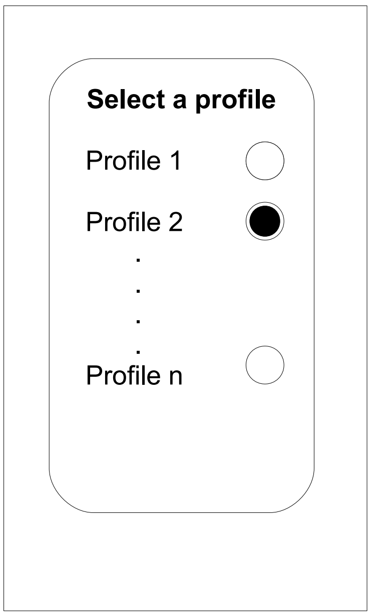


Figure 18: pop-up menu for profile selection

The second screen required is one where the user is able to set up each profile, or make some new. Firstly, the screen must have a drop-down list where the user can select the profile that wants to set up. The state of the rest of controllers in the screen will vary depending on what profile is selected in this list. If the user select "new" in this list, a text field will appear just under the **profiles list**, which the user can fill with the **new profile name**. Bellow, we need a slide bar which represent the desired **light level** inside the house measured in luxes. Also we need other slide bar to represent the desired **temperature**. Furthermore, we need to say to the application when it should **close the windows**. To do this, there are a list with the four different options available. The user will have to choose one between them: *When rain, When wind, Always* and *Never.* Finally, on the bottom of the screen we need a slide bar which represent the **outside light level** required to switch on the outside lights, followed by a **save** button to save all the changes done. All the information collected by this controllers is required in order to allow the application automate the possible scenarios presented in section 3.6. A sketch of the profile settings screen is given in Figure 19.

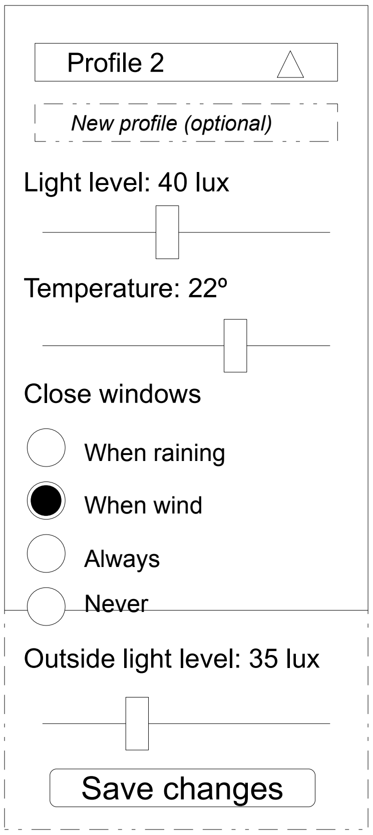


Figure 19: Sketch of the profile settings screen

Should be noted that the initial state of each controller when a profile is selected, corresponds to the value saved the last time the profile was set. In this way, the user can see the current configuration of each profile only selecting it in the drop list.

4.4 Security

As I said in Chapter 3, when using a remote applications, security becomes in one of the most important elements. In our case, HomePort also require some secure methods to access to it. In particular they only accept a SSL communication to be accessed. Furthermore, they request a signed certificate.

In order to allow the connection to the server, the HomePort development crew provided a server certificate and public key for the first version of DOTG, which I will use for this version. This both files were encoded in a different format (PEM) to that used by java (DER). This problem can be fixed by using the tool *openssl* which can convert both files from one format to the other.

4.5 Communication manager

The communication manager is the java class which will unit all the functionality related with the communication with the HomePort server. This class is called *CommMan* and among its task are as follow:

- Establish and finish the communication.
- Handle all possibles communication exceptions.
- Get the whole networks description allocated in the server.
- Interact with devices.

Moreover this class prevents changes in requirements for access to HomePort affecting the application functionality in a serious way. In case of this happens, it would only be necessary to change this class by another, or simply change it. Besides if in some moment we want to adapt DOTG to other type of domotic server, it would be also possible changing only the communication manager.

4.6 Persistence manager

As well as the "communication manager" groups all the functionality related to the communication with the server, in this case, the persistence manager is the responsible of all tasks related to storing and reading data regarding the application. That meas this class called *ProfMan* will do the following things:

- Store new profiles in the application database.
- Read profile configuration.
- Update changes in the profile during the configuration process.

Once more, as in CommMan case, this class prevents changes in the database, such as more detailed profiles, affecting the whole application functionality. If that happens, it is only necessary changes in this class.

4.7 Automation service

Previously, in section 4.2.1, I talked about the automatic behavior of the application which will be the responsible of check the status of the available devices. This "automatic behavior" is managed actually by an Android service which will be running in background once connection with the HomePort server is established. At the beginning, the service checks what networks it have to handle and retrieves from the database the profile that it have to apply to each of them. After that, the service checks the status of every devices and makes in them the needed changes according to the profile selected for the network where the device is located.

Since the service must be aware of changes that can occur in both sensors and actuators, all the process of check and adjust the status of the devices have to be repeated periodically. The length of this period of time should be able to be chosen by the user, but due to time constraints this option is left out for future work. Instead, the length of this interval is one minute. This is the length that allows the application be aware of some change in the house status in a relatively small space of time, and make a reasonable use of the bandwidth.

Moreover, if any physical user make any change in an actuator, the service detects that this device is being manipulated by someone in the house, giving priority to this fact, and excluding this device of being manipulated from the application.

4.8 Summary

In this Chapter I have explained how the list of devices have to be parsed from the XML network document which resides in the HomePort server, taking in account all tools that Android provides. Then I have addressed issues regarding manipulation of the devices and all troubles that involved. Furthermore I have sketched the GUI of the application, explaining thoroughly the most functional screens. After that I have had a look over security through SSL, followed by a description of the Communication Manager and the Persistence Manager and their functions. Finally, I have shown how works the background service responsible for make from DOTG a real automatic remote controller of the house.

5 Implementation

In this chapter I explain the main issues regarding the implementation, described in more detail the classes that I consider more interesting and novelty in this version of DOTG.

5.1 Main Activity

In this application the main class is “HomePort” which extends the Activity class. Besides being the main class, this class will serve as example to show some basic "Android" issues needed for the other classes. In the following Listing 2 I will show an extract of the “HomePort” class.

```
...
public class HomePort extends Activity implements OnClickListener {

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.main);

		// Setup ClickListener
		View connectButton = findViewById(R.id.connect_button);
		connectButton.setOnClickListener(this);
		...
	}

	public void onClick(View v) {
		switch (v.getId()) {
		case R.id.connect_button:
			Intent i = new Intent(this, ListLVL1.class);
			startActivity(i);
			break;
		...
		}
	}
}
```

Listing 2: "HomePort" class

When the activity is going to be created, the method *OnCreate()* is invoked with the Bundle *savedInstanceState* as parameter. A Bundle is the object that store the last state of an Activity it was previously stopped. Later, the *setContentView()* is called. This method associates the activity with a GUI representation which was previously defined in a XML file called main.xml (Listing 3). This view is statically referenced in the auto-generated file R which has the reference to every Android resources.

In this view there are some buttons that allow to interact with the activity. These buttons are also interpreted as a view and we can refer they by using the method *findViewById()* which receives as a parameter the the ID number of the button, and replies with the own button. This button needs a handler which reply to the events generated when clicking the button. That is a task of the class itself that will catch all click events generated by the button, which we have to indicate who is going to be its "listener". This is done through the method *<button>.setOnClickListener(this)* and giving as a parameter the own class.

The events generated by all buttons in the view are caught by the class method *onClick()* which implements a switch-case structure to differentiate the button pressed, given as a parameter. In this case, once the "Connect" button is clicked, a new Intent is created with the current and next activity and then it is passed to the *startActivity()* method.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/my_background">
    <TextView
      android:id="@+id/main_title"
      android:layout_height="wrap_content"
      android:text="@string_/main_title"
      android:layout_gravity="center_horizontal"
      android:layout_width="wrap_content"
      android:textSize="22px"
      android:layout_margin="20px">
    </TextView>
    <Button
        android:id="@+id/connect_button"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/connect_label" />
    ...
</LinearLayout>
```

Listing 3: Extract of the main screen layout XML.

The Listing 3 is part of the XML file which contents the definition of the main screen. At the first line we can see the tag *LinearLayout*. That means that inside this layout every elements are going to be placed in a row. Also there is an attribute in the first line which indicating the android XML schema that describes the correct structure of the XML file. The rest of the attributes just below this last, indicate appearance-related properties that I will explain in the following section. Finally, we can see two tags of two different components of a view as an example of how they are included inside the layout.

5.2 GUI

As I said in the previous section, the whole GUI is made in XML. In order to explain how looks the representation of a XML document I will itemize the different properties used and also give the final screen achieved.

The second line of the Listing 3 describes the orientation property of the layout which determine if the elements are going to be situated next to each other or one below the other. The two next attributes say if the size is going to match the content or the container, and the last one regards the picture, stored on the drawable resource folder, used to the background. After that, a new text field is created to contain the main title. The first attribute of this element prefixed by "@id" indicates the name through which the field is referenced in the auto-generated resource file **R**. Then further different settings are defined, such as the content of the field and the format. After that, four buttons are also defined, even only one has been included in the example. The graphic result of this XML document is given in Figure 20., as well as the "Settings" and "About" screens (Figure 21 and 22), both created in a similar way.

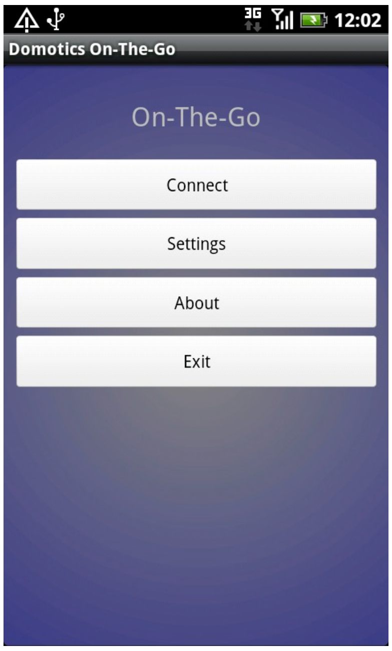


Figure 20: Main screen

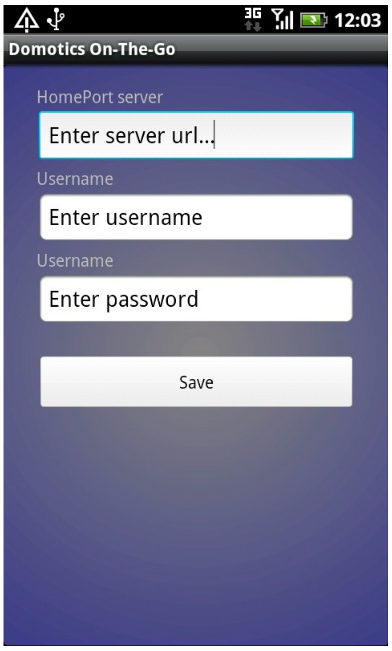
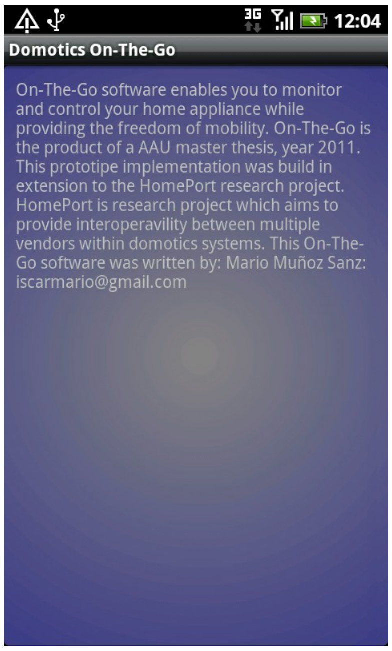


Figure 21: Settings screen



On-The-Go software enables you to monitor and control your home appliance while providing the freedom of mobility. On-The-Go is the product of a AAU master thesis, year 2011. This prototipe implementation was build in extension to the HomePort research project. HomePort is research project which aims to provide interoperavility between multiple vendors within domotics systems. This On-The-Go software was written by: Mario Muñoz Sanz: iscarmario@gmail.com

Figure 22: About screen

When the "Connect" button is clicked, the event is caught by the main activity which invokes the method *onClick()*. Then, the switch-case structure recognizes the id of the button and creates a new Intent with the ListLV1.class to later start that activity (see Listing 3). This new activity has three main tasks. First of all, it is the responsible of show to the user the available networks in the server so it will start the parse of the network XML by creating a CommMan object. The activity will call the method *getNetworks()* and will show the result like a list of networks, as shown in Figure 23. We can see how is it done in the first half of Listing 4.

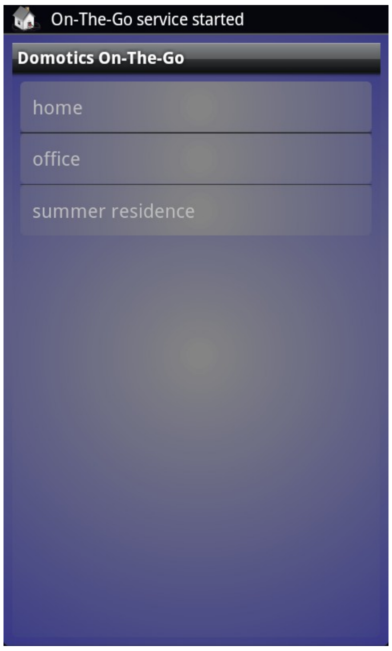


Figure 23: Networks screen

Moreover, this class as well as the "HomePort" class does, implements the "listener" for the elements of the list of networks. When a network is selected, the method *onItemClick()* identifies the pressed network, and create the new Intent for the new activity with the ListLV2.class.

Thirdly, this activity is the responsible of start the background service which will control every changes in the house sensors, and will manipulate all the devices according to the profiles settings. In the second half of Listing 4 we can see how the service is started within a try-catch structure. The way of do it is very similar to the way of start a new activity. Firstly, the beginning of the service is recorded on the log file. After that, it creates a new Intent with the ControlService.class, which implements all the functionality of the service. Finally, the service is started by calling the method *startService()* and giving the created Intent as a parameter. When this happens, a notification appears in the android notification bar (see Figure 23). How this notification is launched and the whole functionality of the background service is discussed in section 5.5

```
@Override
    public void onCreate(Bundle icicle)
    {
      super.onCreate(icicle);
        setContentView(R.layout.device_list);
        lv1=(ListView)findViewById(R.id.MyList01);
        getWindow().setBackgroundDrawableResource(R.drawable.my_background);


        cm = new CommMan();  //  Fetch Data from URL
        myList = cm.getNetworks();
        lv1.setAdapter( new ArrayAdapter(this, R.layout.list_item, myList) );
        lv1.setOnItemClickListener(this);

        // Start of the control service
        try{
            Log.i(getClass().getSimpleName(),
                  "Starting service...");
            Intent intent = new Intent(this, ControlService.class);
            startService(intent);

        }catch(Exception e){
            Toast toast = Toast.makeText(this, e.getMessage(),
                 Toast.LENGTH_SHORT);
            toast.show();
        }

    }
```

Listing 4: Code fragment of the ListLV1 class.

As I said before, when the user press a network, the application starts the activity ListLV2 which show the available locations in the selected network. This class will not be discussed in detail as it resembles listLVL1. The only different thing is that in this activity is available a menu which has two buttons. A picture of this locations screen is shown in Figure 24. The first one shows a pop-up dialog that allow the user change the profile that is being applied to the selected network. The second button will launch the profile settings activity. To create this menu is necessary to override two additional methods. A code fragment with the implementation of this two methods is given in Listing 5. The method *onCreateMenu()* is called when the user presses the physical menu button. First, we get a reference to the inflater through the method *getMenuInflater()* and then we generate the menu structure by calling its method inflate(). This method receives as a parameter the id of a menu defined in a XML file. After all we return true to confirm that the menu has to be shown.

The second method override is *onOptionsItemSelected()* that is invoked when the user clicks an option of the menu. It receives by parameter the id of the option selected and decides what instructions has to execute through a switch-case structure. In in this case, if the user presses the first option an alert dialog is shown to choose a profile. Instead, if the chosen option is the second, a new Intent is created with the Profile_settings.class, and then the new activity is started. This last class is discussed few paragraphs later.

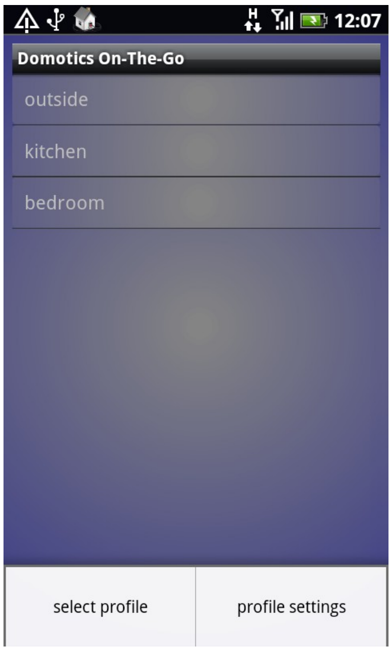
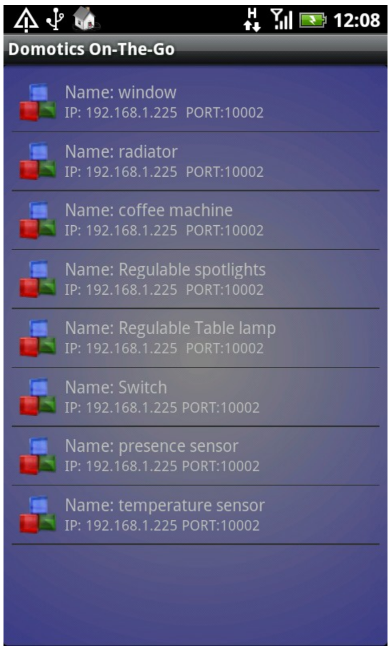


Figure 24: Location screen

```
 @Override
    public boolean onCreateOptionsMenu(Menu menu){
      MenuInflater inflater = getMenuInflater();
      inflater.inflate(R.menu.menu_house_selected, menu);
      return true;
    }
@Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle item selection
            int Id = item.getItemId();
            case R.id.boton1:
                 AlertDialog.Builder builder = new AlertDialog.Builder(this);
                 ...
                 AlertDialog alert = builder.create();
                 alert.show();
                 }
                 return true;
            case R.id.boton2:
                 Intent i = new Intent(this, Profile_settings.class);
                 startActivity(i);
                 return true;
            default:
                  ...
            return super.onOptionsItemSelected(item);
      }
    }
```

Listing 5: context menu methods

The final list class - listLVL3 - displays the devices within a specific location. This class uses the same approach as in the first two list classes, but instead of creates a new list class when an item is pressed, displays a message with the state of the device in question. An example of this device screen is show in Figure 25.


12:08

Domotics On-The-Go

Name: window
IP: 192.168.1.225 PORT:10002

Name: radiator
IP: 192.168.1.225 PORT:10002

Name: coffee machine
IP: 192.168.1.225 PORT:10002

Name: Regulable spotlights
IP: 192.168.1.225 PORT:10002

Name: Regulable Table lamp
IP: 192.168.1.225 PORT:10002

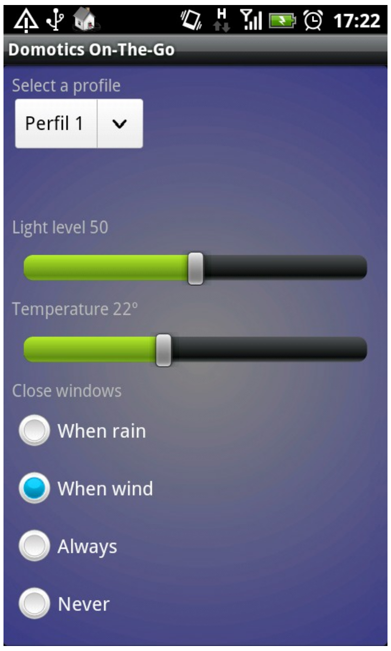
Name: Switch
IP: 192.168.1.225 PORT:10002

Name: presence sensor
IP: 192.168.1.225 PORT:10002

Name: temperature sensor
IP: 192.168.1.225 PORT:10002

Figure 25: Device list screen

Finally, to conclude this section, I will present the *Profile_settings.class* which is the most innovative in this version of DOTG. This is the activity responsible of allow the user to set profiles that can be applied to the different networks available within the server. As I said previously, this activity is started from the location screen, when the user press the second option of the menu. Figure 26 shows an example of the profile settings screen, created following the sketch given in section 4.3.4, Figure 19.


17:22

Domotics On-The-Go

Select a profile

Perfil 1

Light level 50

Temperature 22°

Close windows

When rain [✓ When wind] Always Never

Figure 26: Fragment of the profile settings screen

Then I explain in detail the most important parts of the Profile_settings.class. As we can see in Listing 6, at the beginning we associate the activity with its GUI by using the method *setContentView()*. Then all the controls of the screen must be also associated with their view in the GUI. In this listing I only show one control of each type by the whole list is obviously longer.

```
@Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.profile_settings);

        seekbar_light = (SeekBar) findViewById(R.id.seekBar_light_level);
        seekbar_light.setOnSeekBarChangeListener(this);
        light_level = (TextView) findViewById(R.id.text_light_level);
        radioGroup_close_w = (RadioGroup)findViewById(R.
                                    id.radioGroup_close_w);
        checkBox_outside = (CheckBox)findViewById(R.id.checkBox_outside);
        s1 = (Spinner)findViewById(R.id.profile_list);
             ...
```

Listing 6: code fragment of Profile_settings.class: controls declaration

As we can see in Figure 26, the first control that that we found is the drop-down list which allow to select the profile that the user want to edit. This kind of list is called Spinner in Android. First of all we have to declare an *ArrayAdapter* which will be use to support the values of our list (see Listing 7). Then we create a Profile Manager object (functions of this class are detailed in section 5.3) and ask it for the list of profiles available, which is returned as a Cursor. After that, we have to connect the spinner with the adapter as it is shown in the three following lines. This spinner must be filled with all the names of the profiles. To do it we move the cursor from the beginning to the end of the results and we add each name to the list. At the end we put a new string which indicates the user wants configure a new profile. Finally we say the spinner what it have to do when an item is selected by setting its *onItemselectedListener*. This listener calls the method *showProfileSetting()* that will adjust all controls according to the measurements of the profiles selected.

Listing 7: code fragment of Profile_settings.class: dropdown list of profiles

```
        ArrayAdapter<String> adapterForSpinner;

        ProfMan man = new ProfMan(getApplicationContext());
        Cursor c = man.getNames();

        adapterForSpinner = new ArrayAdapter<String>(this,
             android.R.layout.simple_spinner_item);
        adapterForSpinner.setDropDownViewResource(android.R.layout.
                simple_spinner_dropdown_item);
        s1.setAdapter(adapterForSpinner);
        if (c.moveToFirst()) {
               do {
                      adapterForSpinner.add(c.getString(0));
```

```
            } while (c.moveToNext());
            adapterForSpinner.add("New");
      }
      s1.setOnItemSelectedListener(new OnItemSelectedListener() {
            public void onItemSelected(AdapterView<?> parent, View view,
                        int position, long id) {
            ...
            }
            public void onNothingSelected(AdapterView<?> parent) {
            ...
            }
      });
      ...
```

Two more elements of the screen must be manipulated within the method *onCreate()* of this class. The way in it is done is shown in Listing 8. The first one is the check box that allow to say to the application that never has to switch on the outside light. This check box has to disable the seek-bar that indicate the outside light level required. To do it we have to call the method *setOnCheckedChanged()* by giving it as parameter a new listener for this event. This listener will disable the *seekbar_outside* when the box is checked, and enable it if otherwise.

In Listing 8 we also can see the implementation of the functionality of the save button. As well as every button we have to implement a new *onClickListener*. This listener creates a *ContentValue* object where is stored every value that have to be updated in the data base. Then we put the progress of each seek-bar in the container and the corresponding value of the selected radio button by using a switch-case structure. In the concrete case of the *seekbar_outisde* we also have to check if the bar is disable. If this happens, the value stored is zero. Once we have the container filled, we create a new profile manager and invoke its method *setSettings()* giving the container as parameter. To end the process a message is displayed to say that the new settings were saved.

Listing 8: code fragment of Profile_settings.class: check box and save button

```
checkBox_outside.setOnCheckedChangeListener(new CheckBox.
      OnCheckedChangeListener() {
      public void onCheckedChanged(CompoundButton buttonView,
                  boolean isChecked) {
            if (isChecked) {
                  seekbar_outside.setEnabled(false);
            } else {
                  seekbar_outside.setEnabled(true);
            }
      }
});
```

```
boton_guardar_profile.setOnClickListener(new OnClickListener() {

      public void onClick(View v) {

            ContentValues valores = new ContentValues();
            valores.put("lightLevel", seekbar_light.getProgress());
            valores.put("temperature", seekbar_temperature.getProgress());
            switch (radioGroup_close_w.getCheckedRadioButtonId()) {
            case R.id.radio_close_w0:
                  valores.put("close_windows", 0);
                  break;
            ...
            }
            if (checkBox_outside.isChecked()) {
                  valores.put("outside", 0);
            } else {
                  valores.put("outside", seekbar_outside.getProgress());
            }
            ProfMan man = new ProfMan(getApplicationContext());
            man.setSettings(valores,
                        String.valueOf(s1.getSelectedItemPosition() + 1));
            showToast("New setting saved");
      }
});
```

Besides *onCrete(),* this class has also a very important method called *showProfileSettings().* This method is shown in Listing 9. First of all, a profile manager is created. This manager is used to get the profile values by calling its *getSettings()* method and giving it the profile code, which was received as a parameter. When we have the values, we have to invoke the *setProgres()* of every seek-bar with the value stored in the cursor as parameter. After that, through a switch-case structure, we check the appropriate radio button according to the value in the cursor. And finally, if the progress of the last seek-bar is zero, the outside light check box is checked and the bar is disabled.

Listing 9: code fragment of Profile_settings.class: showProfileSettings()

```
void showProfileSetting(String codigo) {
      ProfMan man = new ProfMan(getApplicationContext());
      Cursor c = man.getSettings(codigo);

      if (c.moveToFirst()) {
            do {
                  seekbar_light.setProgress(c.getInt(1));
                  seekbar_temperature.setProgress(c.getInt(2));
                  seekbar_outside.setProgress(c.getInt(4));
                  switch (c.getInt(3)) {
                  case 0:
                        radioGroup_close_w.clearCheck();
                        radioGroup_close_w.check(R.id.radio_close_w0);
                        break;
                  ...
                  }
                  if (c.getInt(4) == 0) {
                        checkBox_outside.setChecked(true);
                        seekbar_outside.setEnabled(false);
                  } else {
                        checkBox_outside.setChecked(false);
                        seekbar_outside.setEnabled(true);
                  }
            } while (c.moveToNext());
      }
}
```

5.3 ProfMan

In order to encapsulate the functionality regarding management of the profiles stored in the SqLite data base, I create the class "ProfMan", following the style of "CommMan" from the previous version of DOTG. This class is the responsible of access to the data base to retrieve data of the profile settings and of the associated profile to each network. The class is also responsible of save any change in the profile settings and association between networks and profiles. This class has four public methods:

- getNames(): Cursor
- getSettings(String profile): Cursor
- setSettings(ContentValues values, String profile): void
- getNetworkSettings(): Cursor

Listing 10 shows an example each kind of method implemented in this class. But before this, the creation method is developed. As we can see, the method create a *SqlProfile* object, which is an extension of the class *SQLiteOpenHelper*. This last class is the responsible of give access to the data base and create it with the predefined profiles if it is open for the first time (see Appendix B: SqlProfiles.java to watch the source code of this class). Then, a writable database is created by using this last object. After the creation, we can see the *getSettings()* method. First of all two arrays of strings are created to content the name of the columns desired and the value of the "where" clause. Below, we make the query to the data base through the method *query()*. This method needs seven arguments to be invoked. The first one is the name of the involved table, followed by the array of columns, the "where" clauses and the array with the values of this clauses. The last three parameters represent the "groupBy", "having" and "orderBy" clauses respectively.

Listing 10: code fragment of ProfMan.class

```
public class ProfMan {
	SqlProfiles profDBH;
	SQLiteDatabase db;
	public ProfMan(Context cont){
		profDBH = new SqlProfiles(cont, "DBProfiles", null, 1);
		db = profDBH.getWritableDatabase();
	}
	...
	public Cursor getSettings(String profile){
		String[] campos = new String[] { "nombre", "lightLevel",
				"temperature", "close_windows", "outside" };
		String[] args = new String[] { profile };
		Cursor c = db.query("Profiles", campos, "codigo=?",
				args, null, null, null);
		return c;
	}
	public void setSettings(ContentValues values, String profile){
		String[] argsUp = new String[] { profile };
		db.update("Profiles", values, "codigo=?", argsUp);
```

```
        }
        ...
}
```

As example of a method to write in the data base we also have the *setSettings()* in the Listing 10. This method receive as parameter the values that has to store, and the code of the profile in question. The we only have to put the profile code in an array of strings and call the *update()* method of the data base object, which requires four parameters: the name of the table, the values, the "where" clause and the value to this clause.

5.4 HttpClient

The http client class is the complementary class of the Communication Manager developed in the previous version of DOTG. The responsibility of this class is to send to the HomePort server the PUT and GET query which alter or read the status of the sensors and the actuators within the controlled houses. This class is used by the control service every time that it need to manipulate an actuator or it need to know the current state of the house. As well as its complementary class "CommMan" encapsulates the functionality regarding the network structure, this class also does the same with the access to devices. If because some reason the way to manipulate is changed in the future, thanks to the "HttpClient" class the application would be affected slightly.

This class implements three main methods. The first of them is se*tState()*, which need as parameters the id of the devices to be manipulated, and value which represent the desired state. The second method is *getState().* In this case, it receives the id of the device and return an string with the current state of the device. This two methods use the third of them which is called *executeHttpGet().* As its name says, it is the responsible of execute the request given as a parameter by the the other two methods. Then, by using a buffered reader, it store the server response and return it as a string. (See appendix C to watch the source code of this class).

Moreover, this class should also be the responsible of the security during the communication process using the SSL method required by the HomePort development team. In the previous version of DOTG couldn't develop this kind of security due to some incompatibilities between the format used by the java truststore, where the certificate and private key must be stored, and the format of the private key provided by the HomePort crew. Unfortunately this problem have not been able to be fixed in the current version due to time constraints. However a simulation of a HomePort server, without any requirement about the SSL security, was developed by Peter Finderup, who is one of the two developers of the last version. This simulation can be checked out in www.homeport.dk. In this web site we also can see all the behavioral of this application in a graphical and intuitive way.

5.5 Control service

One of the main goals of this project was to develop a really automatic application which be able of control the state of the domotic home, and handle the devices according to the user preferences. To achieve this, android provides the services, which allow execute in background some task that does not require the user interaction. In our case, I have implement a the control service class that is an extension of the service class provided by Android. The main operation of this service was detailed previously in section 4.7. In this section I focus over the main methods implemented in this class and their interaction.

Listing 11: code fragment of ControlService.class: mandatory methods

```
@Override
public void onCreate() {
      NetDev = getNetworkDevices();
}
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
      timer = new Timer();
      timer.schedule(new TimerTask() {
            @Override
            public void run() {
                  checkAndAdjust();
            }
      }, new Date(), 60000);
      Notification notification = new Notification(R.drawable.house,
                  "On-The-Go service started", System.currentTimeMillis());
      Intent notificationIntent = new Intent(this, stop_service.class);
      PendingIntent pendingIntent = PendingIntent.getActivity(this, 0,
                  notificationIntent, 0);
      notification.setLatestEventInfo(this, "On-The-Go", "The service is
            currently running", pendingIntent);
      startForeground(Notification.FLAG_ONGOING_EVENT, notification);

      return START_STICKY;
}
@Override
      public void onDestroy() {
            timer.cancel();
            Toast.makeText(this, "Service finished. Connect again to restart",
                  Toast.LENGTH_LONG).show();
      }
```

In Listing 11 we can see how it is necessary to override three methods that are not defined in the predecessor class. The first of this method is *onCreate()* which only store in a list of lists, called *NetDev*, the networks and the devices available within them. This list is obtained as result of invoke the method *getNetworkDevices().* Subsequently the method *onStartCommand()* is override, which is invoked when the method *startService()* is called. Firstly, we create a timer and then we assign it a task and a time period of 1 minute to wait until repeat this task. As we can see, the responsibility of this timer is to execute the method which will check all measurements of the sensors on each network and will manipulate the devices as appropriate. After the timer, a status bar notification is created and displayed to warn the user that the control service is running in background. At the end, we return a constant called *START_STICKY* which indicate that the service will be explicitly started and stopped to run for arbitrary periods of time. Finally we override the method *onDestroy()* that stops the timer by calling its method *cancel()*, and displays a message warning the user about the end of the service.

Previously I have talked about a method responsible of coordinate the manipulation of the house devices according to the sensors devices and the profile settings. This method is *checkAndAdjust().* The first thing that the method do is to create a cursor with all the settings of every network through a profile manager. This cursor includes the id of each network with the settings of the profile it has associated. Then, at the same time that the cursor moves forward, we invoke a set of methods. Each of them is specialized in a different type of devices and receives as parameters the data regarding this type. Below we can see in Listing 12 the source code of this method.

Listing 12: code fragment of ControlService.class: checkAndAdjust()

```
private void checkAndAdjust() {

      ProfMan man = new ProfMan(getApplicationContext());
      Cursor c = man.getNetworkSettings();

      int network = 0;
      if (c.moveToFirst()) {
            do {
                  manageLigths(c.getInt(2), network);
                  manageTemperature(c.getInt(3) + 14, network);
                  manageWindows(c.getInt(4), network);
                  manageOutdoorLights(c.getInt(5), network);
                  network++;
            } while (c.moveToNext());
      }
}
```

Since the code of these methods are tedious but not too convoluted, I will explain how works each of them but without showing any code fragment. The first that appears in Listing 12 is *manageLights()* and as we can see it receive two parameters. The first one is the light level desired by the user inside the house, and the second one, which is common to all this methods, is the network id. This method checks the presence sensors to know if someone is in the house. If that is true, it turns on switches and adjusts dimmers to achieve the required light level. In case of no one is in the house, it switches off every lights to save energy.

The secondly method called is *mangeTemperature()*. This method requires the desired temperature level as parameter besides the id of the network. Once the desired temperature is known, it is compared with the real temperature inside the house measured by the thermostats. Then all the radiators and heaters are turned up or down according to this comparison. Moreover, if the temperature inside the house is more than 3º higher than the desired, the windows are opened by 15% in order to refrigerate the house. This last, is only done if the windows are not already open and it is not raining outside. This method also consider the presence sensors, so if nobody is in the house, it will turn every radiator and heater to 16º to save energy.

Then it is invoked the *manageWindows()* method, giving it as parameter the value which says when the windows must be closed. The functionality of this method is quite simple. For example: if the user selected the option "when rain", it will check the rain sensor and will close the windows if it is raining in that moment. If that is not true, it won't do anything. This behavior is the same if the option selected was "When wind" with the difference that the sensor checked is the wind sensor. Furthermore, if the user chose third option, "Always", the method will do the same than before, but checking both sensor and closing all windows if one of them is detecting something. Finally, when the option "never" was chosen, the method does not check any sensor and never closes any window.

The last method invoked is *manageOutdoorLights()*. It receives as parameters the limit light level needed outside to maintain switched off the outdoor lights and lampposts. When this method, by checking the outside light sensor, detect that it is darkest than desired, it turns on all the outdoor lights.

5.6 Implementation tests

Both during of the implementation process of the application, as for the testing and remediation time, it was absolutely necessary the use of some tool where be able of see the results obtained. To perform this task the Android SDK provides the device emulator, which is a virtual mobile device that runs on your computer. This tool, together with the eclipse development environment, allow the developer to debug the application, and check how it works stopping in the most troubled parts of the code execution and showing the constant and variable values while they are being manipulated. Besides, the android emulator allow to run every existing versions of the OS and many display configurations, among other features, in order to test incompatibilities with our applications. Once known all the emulator features, the OS version chosen to test DOTG was Android 2.2 (Froyo) with a HVGA display (320x480) which is the most extended configuration in the market [1][2].

However, although the Android Emulator provides almost all the features of a real device, I consider that the application should be also be tested in a physical platform to rule out unexpected errors. The chosen device in this case is an HTC Desire (known as HTC Bravo in USA) with the 2.2 version of Android.

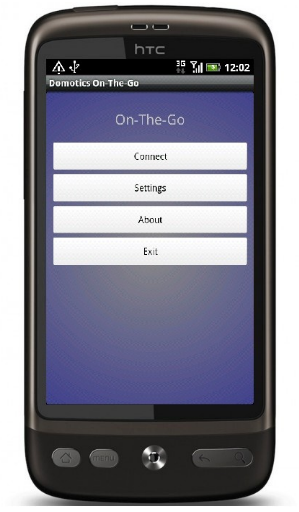


Figure 27: HTC Desire running Android Froyo

In addition to the device emulator, the Android SDK also provides some other tools which were very useful during the implementation process. On the of them is the Dalvik Debug Monitor Server (DDMS). This tool provides a file browser for the device/emulator memory content, port-forwarding services, screen capture on the device, thread and heap information on the device, location data spoofing, among other features.

Other tool very useful provided by the SDK is the Android Debug Bridge (ADB). This is a command line tool that lets you communicate with an emulator instance or connected Android-powered device. It is a client-server program that includes three components: a client which runs on your development machine, a server that runs as a background process on the development machine and a daemon, which also runs as a background process on each emulator or device instance. ADB allow you to issue a few different commands from a command line on the development computer, being the command *shell* the most used during the development of DOTG. This command Starts a remote shell in the target emulator/device instance giving you access as if it was a common linux machine. Thanks to this shell and the sqlite3 command, we are able to query any information from the application data base as is done in any other SQL database.

Finally, as I already remarked in section 5.4, some thing essential to test the implementation was the web site www.homeport.dk also developed by an Aalborg University student. This website simulates a HomePort server allowing to be handled by Http GET and PUT queries. Furthermore it pretend to be as controlling a real house with different kind of typical locations within a house and with devices of different types installed in these locations. In Listing 13 I show an example of a http query used to manipulate a radiator installed in the kitchen.

```
Http://www.homeport.dk/onthego.php?id=kitcrr1&state=24
```

Listing 13: Http query to the HomPort simulation website

The first part of the URI address regards the website. Then we can see that we call to a file called onthego.php which is the responsible of respond to our request as if it was a HomePort server. Then we give as parameters the id of the device in question, and the new value for its status. The effect of this query can be appreciated later if we browse in the web site to the kitchen devices page. In case we only give it the first parameter, the server will reply us with a simple string indicating the state of the device.

6 Conclusions

After a study comparing the sales in the smartphone market I identified the Android OS as the mobile platform with more growth expectations, and as which contains a greater number of potential users of Domotics On-The-Go.

I have developed a mobile application by following the requirement exposed in the analysis section and achieving almost all of them. This applications, based on a previous simple version, is able to connect to a remote server and interact with it by using the http request. Moreover it is able to retrieve all the information about how are distributed the home devices by parsing its description file xml, and represent this structure in a friendly and useful way. Furthermore, Domotics On-The-Go also allow the user to create and manage configuration profiles in order to change the device status and the house behavior quickly and easily.

Unfortunately, due to time constraints, the requirement regarding the security connection using the SSL protocol could not be reached. Nevertheless the implementation could be tested without this feature thanks to the existence of a server simulation, also developed as a AAU student project.

Moreover, this last requirement will be easily addressed in a future work thanks to the developed class *HttpClient*, which encapsulates all the responsibility of send and receive all the request to the remote server. This class can be substituted with no major changes to the rest of the architecture. Because of that, the application can be also easily adapted to any other system requiring any other kind of communication protocol.

As well as the last class, the ProfMan encapsulates the functionality to manage the profiles. Thanks to this, the application could be improved in the future adding more details to the profiles or saving it in the cloud, without seriously affecting to whole application.

Finally I succeeded to implement all the control scenarios proposed during the analysis chapter by creating the Control Service. This service runs in background and takes care of every devices in the house are set according to the profile selected and the house measurements provided by the available sensors.

6.1 Future work

The options to future work are abundant due to the fast development of new features in the smartphone technology. Moreover there is the possibility of develop more control scenarios involving more kind of devices and sensors and obtaining a more efficient behavior of the managed house. Below I propose some example of future work taking into account these issues:

Voice recognition

More and more mobile applications are integrating this feature which allow a free hand interaction with the mobile phone. In the case of Domotics On-The-Go, this feature would give the user the possibility of interact with their house talking to the phone. For example, the user could change the current profile, or activate the security system while he is driving using the free hand device.

Location commands

As well as the voice recognition, this feature is present in ever more smartphone applications. Thanks to this, the application may close the doors and windows when the user is going away from the house or start to heat the house when the application detects that he is going back.

Variety of platforms

Although Android platform was recognized as being which has more potential clients, other platform like Apple iOS or Blackberry OS also have a very important presence in the smartphone market. Because of that, is necessary always consider the possibility of develop other versions of this application to be execute in other platforms.

Appendix A: Network XML description

```
<?xml version="1.0" encoding="utf-8"?>
<devicelist name="Homeport Register" id="devices">
  <mynetworks>
    <network name="home">
 <device uid="E399999D00BC1500" ip="192.168.1.225" id="outsws1" location="outside"
       type="7A621A210401090001080000030004000500060002070A000900010604" port="10002"
      desc="wind sensor ">
       <service value_url="http://roombrige5.:8080/services/C0A80198C0A801E12712/some-id4" id="some-id4"
       desc="wind sensor">
         <parameters>
           <parameter  min="0" max="100" step="1" type="int" id="1" unit="boolean"/>
         </parameters>
       </service>
     </device>
     <device uid="E399999D00BC1500" ip="192.168.1.225 " id="outsrs1" location="outside"
       type="7A621A210401090001080000030004000500060002070A000900010604" port="10002"
      desc="rain sensor">
       <service value_url="http://roombrige5.:8080/services/C0A80198C0A801E12712/some-id4" id="some-id4"
       desc="rain sensor">
         <parameters>
           <parameter  min="0" max="100" step="1" type="int" id="1" unit="boolean"/>
         </parameters>
       </service>
     </device>
     <device uid="E400000D00BC1500" ip="192.168.1.225 " id="kitcwd1" location="kitchen"
      type="7A621A210401090001080000030004000500060002070A000900010604" port="10002"
      desc="window ">
       <service value_url="http://roombrige5.:8080/services/C0A80198C0A801E12712/some-id4" id="some-id4"
       desc="Regulable window">
         <parameters>
           <parameter  min="0" max="100" step="1" type="int" id="1" unit="percentage opened"/>
         </parameters>
       </service>
     </device>
     <device uid="E400000D00BC1500" ip="192.168.1.225 " id="kitcrr1" location="kitchen"
      type="7A621A210401090001080000030004000500060002070A000900010604" port="10002"
      desc="radiator">
       <service value_url="http://roombrige5.:8080/services/C0A80198C0A801E12712/some-id5" id="some-id5"
       desc="Adjustable radiator">
         <parameters>
           <parameter  min="14" max="36" step="1" type="int" id="1" unit="degrees"/>
      </parameters>
    </service>
  </device>
  <device uid="E400000D00BC1500" ip="192.168.1.225 " id="kitccm" location="kitchen"
   type="7A621A210401090001080000030004000500060002070A000900010604" port="10002"
   desc="coffee machine">
    <service value_url="http://roombrige5.:8080/services/C0A80198C0A801E12712/some-id4" id="some-id4"
    desc="Regulable window">
      <parameters>
        <parameter  min="0" max="100" step="1" type="int" id="1" unit="percentage opened"/>
      </parameters>
    </service>
  </device>
  <device uid="8717010000BC1500" ip="192.168.1.225" id="kitcts1" location="kitchen"
  type="7A621022040107010103000003000604010A00" port="10002" desc="temperature sensor">
    <info/>
    <service value_url="http://roombrige5.:8080/services/C0A80198C0A801E12713/someid6" id="someid6"
    desc="temperature sensor">
      <parameters>
        <parameter type="string" values="On, Off" id="1"/>
      </parameters>
    </service>
  </device>
  <device uid="8717010000BC1500" ip="192.168.1.225" id="bedrlp1" location="bedroom"
  type="7A621022040107010103000003000604010A00" port="10002" desc="lamp">
    <info/>
    <service value_url="http://roombrige5.:8080/services/C0A80198C0A801E12713/someid6" id="someid6"
    desc="lamp">
      <parameters>
        <parameter type="string" values="On, Off" id="1"/>
      </parameters>
    </service>
  </device>
  <device uid="8717010000BC1500" ip="192.168.1.225" id="bedrwd1" location="bedroom"
  type="7A621022040107010103000003000604010A00" port="10002" desc="window">
    <info/>
    <service value_url="http://roombrige5.:8080/services/C0A80198C0A801E12713/someid6" id="someid6"
    desc="light switch">
      <parameters>
        <parameter type="string" values="On, Off" id="1"/>
      </parameters>
    </service>
  </device>
</network>
```

```
    <network name="office">
        </network>
    <network name="summer residence">
         <device uid="E400000D00BC1500" ip="192.168.1.225" id="mariooooooo" location="bedroom"
      type="7A621A210401090001080000030004000500060002070A000900010604" port="10002"
      desc="Develco Zigbee light switch device ">
        <service value_url="http://roombrige5.:8080/services/C0A80198C0A801E12712/some-id1" id="some-id1"
        desc="On/Off switch">
          <parameters>
            <parameter type="string" values="On, Off" id="1"/>
          </parameters>
        </service>
        <service value_url="http://roombrige5.:8080/services/C0A80198C0A801E12712/some-id3" id="some-id3"
        desc="electrical Energy counter">
          <parameters>
            <parameter min="0" max="281474976710655" step="1" type="int" id="1" unit="watthour"/>
          </parameters>
        </service>
      </device>
    </network>
  </mynetworks>
</devicelist>
```

Appendix B: SqlProfiles.java

```java
public class SqlProfiles extends SQLiteOpenHelper {
    String sqlCreate = "CREATE TABLE Profiles (codigo INTEGER, nombre TEXT,
    lightLevel INTEGER, temperature INTEGER, close_windows INTEGER, outside
    INTEGER)";
    String sqlCreate2 = "CREATE TABLE Network (name STRING, currentProfile
    INTEGER)";
    public SqlProfiles(Context contexto, String nombre,
                           CursorFactory factory, int version) {
        super(contexto, nombre, factory, version);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(sqlCreate);

        ContentValues nuevoRegistro = new ContentValues();
        nuevoRegistro.put("codigo", 1);
        nuevoRegistro.put("nombre","Perfil 1");
        nuevoRegistro.put("lightLevel", 50);
        nuevoRegistro.put("temperature", 18);
        nuevoRegistro.put("close_windows", 1);
        nuevoRegistro.put("outside", 100);

        db.insert("Profiles", null, nuevoRegistro);

        nuevoRegistro.clear();

        nuevoRegistro.put("codigo", 2);
        nuevoRegistro.put("nombre","Perfil 2");
        nuevoRegistro.put("lightLevel", 20);
        nuevoRegistro.put("temperature", 20);
        nuevoRegistro.put("close_windows", 2);
        nuevoRegistro.put("outside", 10);

        db.insert("Profiles", null, nuevoRegistro);

        db.execSQL(sqlCreate2);
        nuevoRegistro.clear();
        nuevoRegistro.put("name", "home");
        nuevoRegistro.put("currentProfile", 1);
        db.insert("Network", null, nuevoRegistro);
        nuevoRegistro.clear();
        nuevoRegistro.put("name", "office");
        nuevoRegistro.put("currentProfile", 2);
        db.insert("Network", null, nuevoRegistro);
        nuevoRegistro.clear();
        nuevoRegistro.put("name", "summer residence");
        nuevoRegistro.put("currentProfile", 2);
        db.insert("Network", null, nuevoRegistro);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int versionAnterior, int
    versionNueva) {
        db.execSQL("DROP TABLE IF EXISTS Profiles");
        db.execSQL(sqlCreate);
    }
}
```

Appendix C: MyHttpClient.java

```
public class MyHTTPClient {
      private Context context;
      public MyHTTPClient(Context cntxt) {
            context = cntxt;
      }
      public String getCurrentState(String device) {
            SqlDevices devdbh = new SqlDevices(context, "DBDevices", null, 1);
            SQLiteDatabase db = devdbh.getReadableDatabase();
            String[] campos = new String[] { "id", "name", "status", "location",
                        "type" };
            String[] args = new String[] { device };
            Cursor c = db.query("Devices", campos, "id=?", args, null, null,
                        null);
            int nameColumnIndex = c.getColumnIndexOrThrow("status");
            String status = "";
            if (c.moveToFirst()) {
                  do {
                        status = c.getString(nameColumnIndex);
                  } while (c.moveToNext());
                  if (db != null) {
                        db.close();
                  }
            }
            return status;
      }
      public String setState(String device, int value) {
             String temp = "";
              try {
                  temp = executeHttpGet(device,
                     "http://www.homeport.dk/onthego.php?id="+ device
                     +"&state="+ value );
              }
              catch( Exception e ){}
              return temp;
      }
      public String setSensor(String device, int value) {
            SqlProfiles usdbh2 = new SqlProfiles(context, "DBProfiles",null, 1);
            SQLiteDatabase db = usdbh2.getWritableDatabase()
            ContentValues values = new ContentValues();
            values.put("status", value);
            String arg[] = {device};
            db.update("Devices", values, "id=?", arg);
            return device;
      }
      public String getState( String device ){
             String temp = "";
              try {
                  temp = executeHttpGet(device,
                     "http://www.homeport.dk/onthego.php?id=" + device);
                  temp = temp.replace("\n", "");
              }
              catch( Exception e ){}
              return temp;
         }
      public String executeHttpGet(String deviceID, String queryString) throws
Exception {
        BufferedReader in = null;
        String page = "";
        try {
            HttpClient client = new DefaultHttpClient();
            HttpGet request = new HttpGet();
            request.setURI(new URI( queryString ));
            HttpResponse response = client.execute(request);
            in = new BufferedReader(new
                InputStreamReader(response.getEntity().getContent()));
            StringBuffer sb = new StringBuffer("");
            String line = "";
            String NL = System.getProperty("line.separator");
            while ((line = in.readLine()) != null) {
                sb.append(line + NL);
            }
            in.close();
            page = sb.toString();
            } finally {
            if (in != null) {
                try {
                    in.close();
                    } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            return page;
        }
    }
}
```

Bibliography

1: Adroid Developers, Platform Versions, 2011, http://developer.android.com/resources/dashboard/platform-versions.html
2: Android Developers, Screen Sizes and Densites, 2011, http://developer.android.com/resources/dashboard/screens.html
3: Android.com, Data Storage, , http://developer.android.com/guide/topics/data/data-storage.html
4: Android.com, package android.database.sqlite, , packageandroid.database.sqlite
5: Apple, Developer ios dev center, , http://developer.apple.com/iphone/
6: Arne Skou, Homeport home page, , http://www.energybox.dk/en/partners/
7: Ars Technica, iPhone in depth: the Ars review, , http://arstechnica.com/apple/reviews/2007/07/iphone-review.ars/6
8: CRN, BlackBerry Users Call For RIM To Rethink Service, , http://www.crn.com/news/client-devices/222002587/blackberry-users-call-for-rim-to-rethink-service.htm
9: Ed Burnette, Hello, Android,
10: Energybox.dk, Homeport, , http://www.energybox.dk/en/home.htm
11: Gartner, Gartner says worldwide pc shipments grew 7.6 percent in third quarter of 2010, , http://www.gartner.com/it/page.jsp?id=1451742
12: Gartner, Inc., Worldwide Mobile Device Sales to End Users Reached, , http://www.gartner.com/it/page.jsp?id=1543014
13: IDC, Global pc market maintains double-digit growth in third quarter despite weak results in somesegments, according to idc., , http://www.idc.com/about/viewpressrelease.jsp?containerId=prUS22531110&sectionId=null&elementId=null&pageType=SYNOPSIS
14: Indigo, Perceptive Automation. Automate your home with Indigo, , http://www.perceptiveautomation.com/indigo/index.html
15: IO homecontrol, How does it work, , http://www.io-homecontrol.com/en/pros-area/how-does-it-work.html
16: Jeppe Brønsted, Per Printz Madsen, Arne Skou, Rune Torbesen, The HomePort System,
17: KNX Association, What is KNX?, , http://www.knx.org/knx/what-is-knx/
18: Leonard Richardson, Sam Ruby, RESTful Web Services, 2005
19: M. Sipser, Introduction to the Theory of Computation, 1996
20: Neil McAllister, SDK shoot-out: Android vs. iPhone - Apple and Google differ along familiar lines with their smartphone development kits, 2008, http://www.infoworld.com/d/developer-world/sdk-shoot-out-android-vs-iphone-074
21: Netscape Comunications, The SSL Protocol Version 3.0, , http://www.mozilla.org/projects/security/pki/nss/ssl/draft302.txt
22: Open Handset Aliance, Aliance Members, , http://www.openhandsetalliance.com/oha_members.html
23: Perceptive Automation, domotics - control at your fingertips, , http://www.domotics.uk.com/
24: Pico Electronics, X10 (industry standard), , http://en.wikipedia.org/wiki/X10_(industry_standard)
25: R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, Hypertext transfer protocol, 1999, http:www.ietf.org/rfc/rfc2616.txt
26: Reto Meier, Professional Android 2 Application Development, 2010
27: retrocom.com, BellSouth - Simon, , http://www.retrocom.com/bellsouth_ibm_simon.htm
28: Roy Thomas Fielding, Architectural styles and the design of network-based software architectures, 2000
29: S.A. Thomas, SSL & TLS essentials, 2000

30: Stockhomsmartphone, History, , http://www.stockholmsmartphone.org/history/
31: Telia, Salg af smartphones i danmark., , http://www.mobilsiden.dk/nyheder/boom-i-salget-af-smartphones-i-danmark,lid.12139/
32: The Mobile Beat, Google’s Android becomes the world’s leading smart phone platform, 2011, http://www.themobilebeat.com/archives/4384
33: Verisign, Centro de información sobre SSL y credibilidad en línea, , http://www.verisign.es/ssl/ssl-information-center/index.html
34: xmlpull.org, Quick Introduction to XmlPull, , http://www.xmlpull.org/v1/download/unpacked/doc/quick_intro.html
35: Zensys, Z-wave specification,
36: ZigBee Alliance, Zigbee specifications, , http://www.zigbee.org/Markets/ZigBeeSmartEnergy/Specification.aspx
37: CLIPSAL integrated systems, C-bus specifications, , http://www.cbus-enabled.com
38: Simon Silvester, Mobile Mania: a manual for the second internet revolution., 2010, http://pubs.wunderman.com/mobilemania/