

SafeHouse

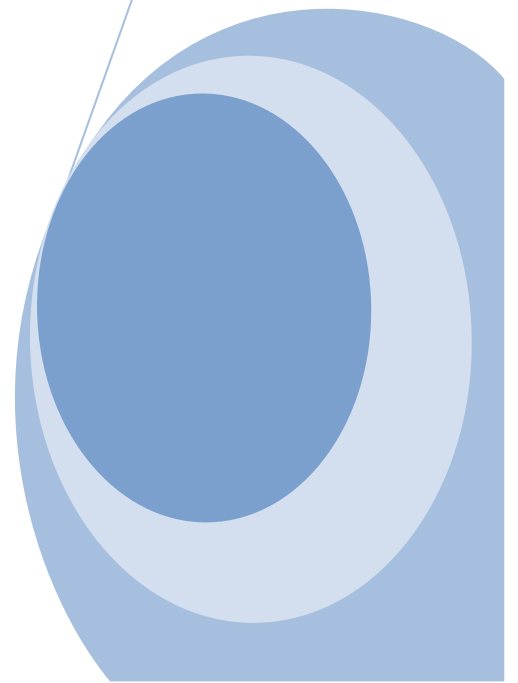
Wireless platform for controlling sensors and actuators in multiple rooms.

Sérgio Pedro

**MSc Eng. Vision, Graphics and Interactive Systems
Master Thesis Project - 10th Semester 2009/2010**

**Department of Electronic Systems
Niels Jernes Vej 12, 9220 Aalborg
Aalborg University**

3rd June 2010



DEPARTMENT OF ELECTRONIC SYSTEMS

VISION, GRAPHICS AND INTERACTIVE SYSTEMS

TITLE: SafeHouse

THEME: Control and Monitor of sensors/actuators in multiple rooms

PROJECT PERIOD: February 1st 2010 to June 3rd 2010

PROJECT GROUP: 10grp1027

GROUP MEMBERS:

Sérgio Pedro

SUPERVISOR: Lars Bo Larsen

NUMBER OF COPIES: 2

REPORT PAGES: 109

APPENDIX PAGES: 12

TOTAL PAGES: 136

ABSTRACT:

This report documents the development of a system to control and monitor sensors and actuators in a house or other indoor environment. The system developed allows a user to get and change status of components quickly and wherever he is through his cell phone. Moreover, the user can have more detailed information at home through a desktop application. The system was developed using Service Oriented architecture:

- A desktop application providing the services for the user and serving as mean of communication between the cell phone (user) and the micro controllers (sensors and actuators).
- A mobile application where the user can in a glance, get the status or change the status of some sensor/actuator.
- A micro controller application, responsible to receive requests from the desktop application, to act according to those requests and to send an answer back.

The desktop and mobile phone applications have been developed in Java programming language, with J2SE and J2ME, respectively. The micro controller application has been developed using C language.

Moreover, the desktop application allows the user to manage and configure the system as he wants. The user is able to create new users, to automate the house and to configure notifications, as well as to check system logs and visualize charts.

This report presents the system development life cycle, from the analysis to the tests.

Preface

This report and all of its contents are part of the 10th semester in Vision, Graphics and Interactive Systems at Aalborg University. It has been written by the project group VGIS 1027 during the spring semester of 2010.

The purpose of this master thesis is to develop a system to control and monitor sensors and actuators, either in houses or in other indoor environment (hotel, office). The theme of this project is: Wireless platform for controlling sensors and actuators in multiple rooms.

Each chapter and the corresponded sub chapters are marked by one or more consecutives numbers. As more consecutives numbers has the index, more deeply is the information presented in there. Regarding the figures, they are simply order in sequence of appearance in the report.

1. Report outline

The structure of this report is divided into 8 parts:

1. Introduction: it is introduced the target of this system and explained the scenarios idealized for the realization of this project.
2. Analyses: here it is analyzed all the different technologies and techniques that it was thought useful for this project. For instance, it was analyzed SOA, existing systems of smart homes, sensors and actuators, communication protocols and mobile phones.
3. Design: here it is explained how it was designed all of the components of this project, either in a global view or in a more specific view.
4. Implementation: here it is explained which problems were faced during the implementation and deploying of the system. It is also explained the choices made to avoid those problems and which problems remained without a possible solution.
5. Testing: here it is shown the final tests made to the final program and which results were obtained.
6. Conclusion: in this part is given a brief conclusion of the entire project, comparing the early ideas of the project and the final solution.
7. Appendixes: in this section it is included some tables, graphics and example files that are not so relevant to include in the previous parts of the report.
8. Bibliography: this section integrates a list of references used to develop this project and to write this report.

2. Authors

The master thesis project *SafeHouse* was developed by the group members Sérgio Pedro and Luben Ivanchev of the group VGIS 1020 till the date of 14th of May of 2010. From that date, the group became two different groups: VGIS 1020 and VGIS 1027.

Thus, the following sections of this report are common to both members, even though there were further minor changes to them:

- Introduction.
- Analysis.
- Appendix A.
- Bibliography.

Moreover, the embedded application of the *SafeHouse* system was designed and implemented by both members of the group VGIS 1020.

Therefore, the remaining parts were designed and implemented individually by Sérgio Pedro, member of the group VGIS 1027.

Acknowledgments

I would like to thank all the people who helped me during the accomplishment of our project, and particularly:

- My supervisor **Lars Bo Larsen**, semester coordinator and project supervisor for his help, management and advice.
- My family and friends, particularly **Emese Timár** for her patience and personal support all over this semester.
- Mr. **Ben Krøyer**, from the E-lab, for his help and supporting regarding the micro controllers and external sensory.
- Ms. **Mette Billeskov**, semester secretary, for her support regarding the administrative issues.
- Mr. **Per Mejdal Rasmussen**, from the IT workshop, for his help and support concerning the desktop application of this project.
- Mr. **Jørgen Schiønning**, CEO of PDM Technology, for his advice and support for the master thesis regarding Service Oriented Architectures. Also, for this permission of using software libraries of PDM Technology in this project.
- Professor **Frank Fizek** and Mr. **Morten Pedersen**, from the mobile devices lab, for lending the necessary phones for this project.

Table of Contents

Preface	3
1. Report outline	3
2. Authors	4
Acknowledgments	5
Table of Contents	6
List of Figures	10
I. Introduction	13
1. Scenarios	14
II. Analysis	16
1. Overall System Architecture	17
2. Service Oriented Architecture	19
2.1. Key elements for a Service Oriented Architecture	19
3. Existing Household Systems	22
2.1. Conclusion	26
3. Embedded Devices	27
3.1. Distance Sensor	28
3.2. Temperature Sensor	30
3.3. Other Sensors	31
4. Mobile Devices	32
4.1. Conclusion	34
5. Communication protocols	35
5.1. X10 Industry Standard (Wired communication)	35
5.2. Radio Frequency Identification – RFID Standard (Wireless communication)	36
5.3. Wireless Personal Area Networks (WPANs)	36
5.4. 3G Mobile Communications (Wireless Communication)	38
5.5. Conclusion	39
6. Problem formulation	40
III. Design	41
1. Architectural Choice	42

1.1.	Remote-Server Architecture	42
1.2.	Home-Server Architecture	45
1.2.1.	Embedded Layer.....	47
1.2.2.	Application Layer.....	48
1.2.3.	Presentation Layer	50
2.	Communication Protocol	51
2.1.	XML as the base of communication	51
2.1.1.	XML Structure.....	51
2.1.2.	Layer Structure	53
2.2.	Application and Embedded Layers Communication	54
2.3.	Application and Presentation Layers Communication	57
3.	Structures of the system	59
3.1.	Room/Component.....	60
3.1.1.	Room	60
3.1.2.	Component.....	60
3.1.3.	Types of Data.....	61
3.2.	Users.....	62
3.2.1.	Definition of user and super-user	62
3.2.2.	Creation and Deletion of an user	63
3.3.	Rules.....	64
3.3.1.	Definition of Rule	64
3.3.2.	Creation of a rule.....	65
3.4.	Notifications	67
3.4.1.	Definition of Notification.....	67
3.5.	Statistics	69
3.5.1.	Average of results in the Last Day	69
3.5.2.	Average of results in the Last Month	70
3.6.	Conclusion	70
4.	Home Server.....	71
4.1.	Discovery of Embedded Devices	72
4.2.	Mobile Server	73
4.3.	Embedded Server	76
4.3.1.	Embedded Looping Block.....	76
4.3.2.	Rules Trigger Block	76

4.3.3.	Notifications Trigger Block	77
4.4.	Statistics Thread	79
5.	Desktop Application	80
5.1.	Loading Screen	81
5.2.	Login Screen	82
5.3.	Main Menu	84
5.3.1.	Overview Screen.....	84
5.3.2.	Notification Screen.....	85
5.3.3.	Rule Screen.....	88
5.3.4.	User Screen	90
5.3.5.	Charts Screen.....	92
6.	Mobile Application	93
6.1.	Elements of the Presentation Layer	93
6.1.1.	Title and Ticker	94
6.1.2.	Choice Group.....	94
6.1.3.	String Item	95
6.1.4.	Text Field	95
6.1.5.	Button.....	96
6.2.	Screens Sequence.....	96
6.2.1.	Login Screen	97
6.2.2.	House Screen.....	98
6.2.3.	Room Screen	99
7.	Conclusion	100
IV.	Implementation.....	101
1.	Problems Faced	102
1.1.	Limitations on the Embedded Devices.....	102
1.2.	File Structure of the Home Server.....	104
1.3.	SMS notifications.....	107
2.	Deployment of the System.....	108
3.	Integrated Development Environments (IDEs)	109
3.1.	Netbeans IDE.....	109
3.2.	MPLAB IDE.....	110
4.	Conclusion	111
V.	Testing	112

1.	Software Performance/Loading Testing	113
1.1.	Desktop Application Loading Testing	113
1.2.	Desktop Application Performance Testing.....	116
1.3.	Mobile Application Loading Testing	117
2.	Stability Testing	118
VI.	Conclusion	119
1.	Personal achievement	120
2.	Main issues and perspectives.....	121
VII.	Appendixes	122
	Appendix A	122
	Appendix B	124
	1. XML Library.....	124
	2. BSCOM Library.....	124
	Appendix C	126
	Appendix D	127
	1. Room/Component Structure.....	127
	2. Users Structure.....	128
	3. Notifications Structure	129
	4. Rules Structure	130
	Appendix E.....	131
	1. Message Class.....	131
	2. GUIInterface Class	131
VIII.	Bibliography	134

List of Figures

Figure 1 - Energy saving in a smart house. If the window is opened, the radiator turns off.	13
Figure 2 - The general idea of our system.	17
Figure 3 - Three point communication of the system.	17
Figure 4 - Basic scheme of SOA architecture (Service provider and Service requesters)	20
Figure 5 - XML scheme of a simple request (left) and a composite request (right).	21
Figure 6 - User motivations for installing smart home products (Meyer & Schulze, 2006).	22
Figure 7 - User acceptance of smart home systems (Meyer & Schulze, 2006).	23
Figure 8 – System architecture based on a SMS protocol to interact with a remote user (Khiyal, 2009).	24
Figure 9 - System architecture proposed by (Alkar & Buhur, 2005).	25
Figure 10 - Open sensor board with pic30f3013 and Bluetooth UART.	27
Figure 11 - Sharp GP2D12 Voltage vs Distance (in cm) graph.	28
Figure 12 - Linearization of the GP2D12 signal (volts vs cm).	29
Figure 13 - The temperature sensor LM35DZ.	30
Figure 14 - Relationship between voltage and temperature (°C) in the LM35DZ.	30
Figure 15 - Other components. From the left to the right: a buzzer, a green led and a magnet sensor.	31
Figure 16 - The evolving smartphone market from 2007 till 2009 (McLean, 2009).	32
Figure 17 - Job trends for different mobile development platforms (Indeed, 2010).	34
Figure 18 - Job trends for Bluetooth and Zigbee (Indeed, 2010).	37
Figure 19 - Power breakdown for a connected mobile device in idle mode (Pering, 2006).	38
Figure 20 - Remote-Server Architecture: All the interactions between the user and the house pass through a webserver.	42
Figure 21 - The Remote Server concept in more detail.	43
Figure 22 - Home-Server Architecture: the user interacts directly with his/her own server (placed at home) wherever he/she is.	45
Figure 23 - Detailed scheme of the Home-Server Architecture.	46
Figure 24 - Three layers architecture of the SafeHouse system: the lamp on the left represents the embedded layer; the server in the middle represents the application layer; and the user on the right represents the presentation layer.	47
Figure 25 - The cycle of operation in the embedded layer.	48
Figure 26 - Both sides of the application layer share the same resources. This is made through a binary file.	49
Figure 27 - XML communication between layers. If the XML is not correctly interpreted by the Receiver Layer, a NAK is send back and the Sender Layer cancels the update of information. .	52
Figure 28- Simple Requests and Answers. Figure 29 – Composite requests and answers.	52
Figure 30 - The general structure of each layer.	54
Figure 31 - First XML exchanged between embedded and application layers.	55

Figure 32 - The first XML sent by the Application Layer to the Presentation Layer. All the definition of the screen is described there and also the method render form is performed in the end.	58
Figure 33 - The Super User is the only one that can create or delete users from the system. This gives an extra security to the system.	63
Figure 34 - Same target value conflict: different values on the source component trigger the same even on the triggered component.	66
Figure 35 - Interval interception conflict: when for some values of the source component two actions would be trigger in the triggered component.	66
Figure 36 - Basic Workflow of the Home Server: After the embedded devices being discovered, it launches the mobile server and the embedded process to update status and trigger rules and notifications.	71
Figure 37 - Workflow of the Discovery process.	72
Figure 38 - Bluetooth/Internet server workflow.	74
Figure 39 - Workflow of the Embedded Looping block.	77
Figure 40 - Workflow of the Rules block.	77
Figure 41 - Screens Scheme of the Desktop Application.	80
Figure 42 - Loading Screen of the Desktop Application.	81
Figure 43 - Login Screen of the Desktop Application.	82
Figure 44 - Loading Screen with empty fields generates an error message.	83
Figure 45 - Loading Screen with a not existing user generates an error message.	83
Figure 46 - Loading Screen with an incorrect password generates an error message.	83
Figure 47 - Main menu of the Desktop Application.	84
Figure 48 - Customization Screen of the Desktop Application.	85
Figure 49 - Notification Screen of the Desktop Application.	86
Figure 50 - Creation of a notification in the Desktop Application.	87
Figure 51 - Notification screen after creating a notification.	87
Figure 52 - Creation of a rule in the Desktop Application.	88
Figure 53 - Conflict in the creation of a rule in the desktop application.	89
Figure 54 - Rules Screen of the Desktop Application.	89
Figure 55 - Creation of an user in the Desktop Application.	90
Figure 56 - Error message in deleting the super user of the system.	91
Figure 57 - Charts Screen of the Desktop Application.	92
Figure 58 - Screens sequence on the mobile application.	97
Figure 59 - Login Screen of the Mobile Application.	97
Figure 60 - House Screen of the Mobile Application.	98
Figure 61 - Room Screen of the Mobile Application.	99
Figure 62 - Bad synchronization when two threads are accessing the same object.	105
Figure 63 - Good synchronization between threads: The Thread B waits for Thread A to finish the transaction.	106
Figure 64 - SMS notifications via an intermediary cell phone.	107
Figure 65 - Netbeans IDE 6.8.	109
Figure 66 - MPLAB IDE v8.20.	110

Figure 67 - Loading Testing for the Home Server when this one is two meters far from the embedded devices.	114
Figure 68 - Loading Testing for the Home Server when this one is 10 meters far from the embedded devices.	114
Figure 69 - Average Results for the Loading Testing for the Home Server.	115
Figure 70 - Performance Testing in the Home Server.	116
Figure 71 - Sharp GP2D12 non-linear and linear graph.	122
Figure 72 - Comparison between different output data in the Sharp GP2D12. The green cell is the average of the absolute error.	123
Figure 73 - Class Diagram for the Rooms/Components Structure.	127
Figure 74 - Class Diagram for the Users structure.	128
Figure 75 - Class Diagram for the Notifications Structure.	129
Figure 76 - Class Diagram for the Rules Structure.	130
Figure 77 - Message class. The attribute rightMessage indicates if the operation was succesful or not; the attribute message is used to display some error information or new information to be put on the GUI.	131

I. Introduction

Sensors and actuators are a vital part of our life. In the nature, they are present in all the life forms we know. Without them we would not survive even a second. Our eyes, our tact, our taste, our muscles, are intrinsically part of the human body. The big agent of this control is our brain, who responds to electrical signals sent from our body sensors, sending electrical stimulus to our actuators (e.g. muscles).

Nevertheless, humans are also dependent from other kind of sensors and actuators, the electronic or mechanical ones. For instance, when we put a dish cooking in the oven, there is a sensor to detect when the temperature overpasses the temperature that we established. After, this sensor sends a signal to the electrical plate (the brain of the oven) and this one sends a signal to the thermal plate (actuator) in order to shut down for a while, till the temperature drops down a bit.

In the last decades we have seen a progressive development of **Smart Houses**. The idea is quite simple: instead of having small brains in each machine, the house has a unique brain to control and monitor all the sensory and actuators built in. Many companies have developed systems for this purpose and increasingly more prototypes are coming to expositions and fairs. There are many goals with these systems: some simply give the user a more centralized control of his house; others are more concerned with power saving; and many are concerned with security.



Figure 1 - Energy saving in a smart house. If the window is opened, the radiator turns off.

Some of these systems are intended to be controlled just at home with a built in panel in the wall where the user can control everything. Other systems are intended to be controlled wherever you are through your cell phone. The idea of this project is to have a plug-n-play system for controlling sensors and actuators in a house or office, through wireless communication, to let the user control his house or office wherever he is at any moment of the day.

The idea of giving the control to the user wherever he/she is, it is only possible with portable devices. The most common and popular ones are the cell phones. Moreover, cell phones are getting increasingly important in our society. Nowadays, we can use our cell phone for almost

everything and there are uncountable mobile applications. Indeed, in the last years we have seen a transformation in this market. The concept of cell phone turned to the concept of smart phone, a device capable of higher processing and which allows the user to make all the actions that he makes on a computer system.

The entire system is idealized to achieve a Service Oriented Architecture. In this way, the format of inter communication between applications should be XML. With this, not only we have an easy-to-install system, but also we have an easy-to-modify system.

The system is made in such a way that the development of one of the components does not interact with the development of another, i.e. each component is completely independent of any other, meaning that the only part that all the components have to have in common is the format of communication between them (input/output). Therefore, the only requirement for a team is to assure that the format is readable and interpreted by each component.

1. Scenarios

At this point, it has been introduced the fundamentals of this project. Nevertheless, all of this does not make sense without a real user scenario. For instance, let us consider a common citizen, called Peter, who decided to implement in his house the *SafeHouse* system with the following equipment:

- For his bedroom: a temperature sensor, a magnet sensor (to detect if the windows is open or not), a light bulb and a heating system.
- For his living room: a heating system and a distance sensor (to detect if somebody is in that room).

After setting up the system, Peter goes to work, already with the *SafeHouse* mobile application installed on his cell phone:

1. During the morning, Peter realized that he left the light of his bedroom turned on. Quickly, he opens the *SafeHouse* app and in few seconds, he turns off the light. Peter just has made his good action for the environment.
2. After an intensive day at work, Peter decides to establish some notifications for the case that he forgets to turn off some light or the heating. For that, he accesses the desktop application and in some quick steps, he creates an email notification to be sent to his email address in case that some light or heating are on. Moreover, he creates a SMS notification to be sent to his mobile phone every time the window is open at home. In this way, he feels more relieved about the security at home.
3. Peter also creates a rule to run any time presence is detected in the living room. This rule would turn on the lights immediately. Peter feels now also his house

more automated. He goes to the living room immediately and checks effectively that the rule is making effect on the system.

4. Some days after the system is running and during the day when Peter is at work, a robber attempts to enter in Peter's house through the window. The robber destroys completely the window and this event triggers a notification to Peter's cell phone. Immediately, Peter calls the police and quickly they arrest the robber. Peter feels that the system is not only automated, but also protects from intrusions.

After introducing briefly this topic and explaining a possible scenario, we are going through the analyzing of existing systems, SOA, mobile phones and communication protocols in the next section.

II. Analysis

In this section all theory behind this project is going to be described. This section will begin with a presentation of a **general diagram** for the system: which system components are involved in it and how is the general interaction between them.

Hereafter, we will introduce the **service oriented architecture**: how the system is able to work based on services in each application, rather than based on a fixed point-to-point application. In order to achieve the concretization of the project, it is going to be made a brief introduction about **existing household systems**: which products are in the market and their potential for the future.

In the next section it is going to be introduced the **embedded devices** used in the system and which sensory they have incorporated, as well as the data extracted from each sensor or actuator. After that, we are going through the topic of **mobile phones**: how is the market of cell phones and which platforms are available for developers. Finally, the last part will refer to different **communication protocols**, especially within the wireless communications.

The analysis section is concluded with the **problem formulation** for this master thesis work, considering all the previous sub sections.

1. Overall System Architecture

As it has been introduced above, the key idea of this system is to let the user control his house or office in a glance through his cell phone. Each user action has to be made through an entry point in the house, possibly a simple server running in a home computer. This means that the user can interact with his house anywhere in the world through his mobile device.

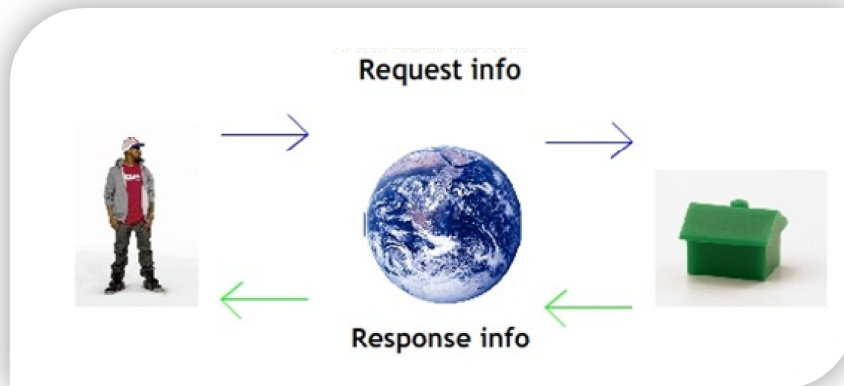


Figure 2 - The general idea of our system.

The most important idea of this system is to allow a user to, at anytime from anywhere in the world, request status information from his automated home. Upon a user request, the system responds with the desired data almost instantaneously. Of course, an entry point has to be considered to the system, i.e. a place where the user connects to this house. The system has to be, in this way, based on 3 point communication: The user (blue square); the house entry point (green square); and the sensors/actuators (red circles).

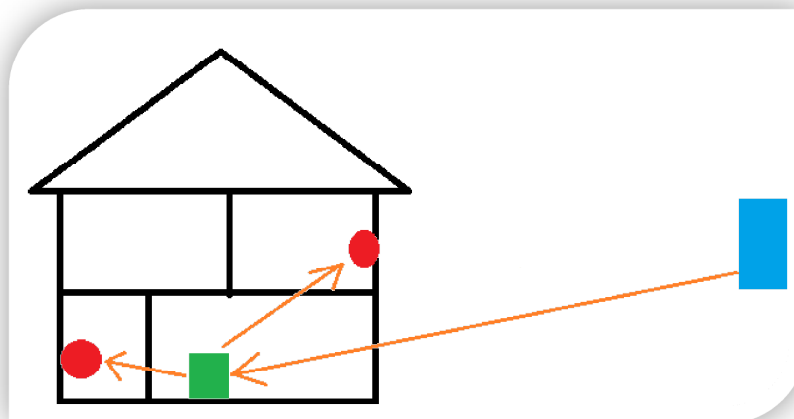


Figure 3 - Three point communication of the system

To reach the completion of such system, a research is going to be conducted through the different topics that are going to compose it. Moreover, before going through the physical components, it has to be introduced the architectures for such systems. Specifically, in the next chapter we are going through Service Oriented Architecture and how it can influence the development of software.

2. Service Oriented Architecture

In the traditional systems, the focus always stands in the development of the application for a specific propose, without taking in account other possible platforms or other kinds of communication. Moreover, the software model is most of the times separated from the business model. Thus, the concerning is to design and develop the software components independently of the business components. Due to this splitting, most likely the developers design the system in an enclosed way, i.e. they implement a standalone application. In another words, even with a good result in the final product, any change in a block of the program will cause changes in the blocks depending on that one. This is really expensive for a company, which afterwards is going to have several costs in the system support.

With this comes the key idea of a Service-Oriented Architecture (SOA): **combine the software model with the business model** (Bieberstein, 2008) (Josuttis, 2007). In this way, each required business service it will become a software service. All the blocks of an application are no longer enclosed together and inflexible for changes. Each service is independent and it can run wherever we want with any kind of communication protocol. Moreover, each service can be reused in another business model or changed without compromising other services. For sure, it is harder to achieve this when a team is doing its first project. But afterwards, a small modification in one of the services or the construction of a new one makes a brand new application ready to be used by other customer. This is really important in terms of costs: the flexibility to separate the services by their functionality. In the middle of this picture, a well-known format comes naturally as the format for service interconnection: the XML format. This format fully enters in the SOA logic. With a small change in a service, we just have to add to the XML a couple of new fields to be understandable by that service, without compromising other services. This is really powerful and it can make a significant difference in the time spent in the development and in terms of costs.

2.1. Key elements for a Service Oriented Architecture

In order to realize a Service Oriented Architecture, several rules have to be followed (Huhns & Singh, 2005):

- *Loose coupling*: A class or a structure in a software application has to have the less possible knowledge of other components in the system. This means that if a requirement change occurs, the cost of that change will be minimal.
- *Implementation neutrality*: Good software architecture should not be dependent on programming language details.
- *Flexible configuration*: One of the strongest principles of SOA. With this, different components can be bound together later in the project.
- *Persistence*: Services should endure long enough and have a correct way of handling exceptions.

- *Granularity*: This is strong related with the combination of the software model with the business model. The services should not be seen in detailed interaction among them, but instead as a high-level view more related to the business model.

Respecting these few rules may contribute for a better software development. This does not mean that the software development will be faster in an initial stage, but for sure, it will become easier to correct or to add/change features.

So far, we have seen SOA and its rules, but in order to see a better picture of SOA architecture, it has to be introduced two main concepts: the service provider and the service requester (Schmidt, 2005). Obviously, the service provider is the one that provides all the services available to one or more service requesters. The crucial thing here is that the service provider can provide different kind of services for different purposes without changing the structure inside of it. In this way, different service requesters (green triangles) will retrieve different data (orange arrows) from the service provider (blue hexagon) as we can see in the Figure 4 below.

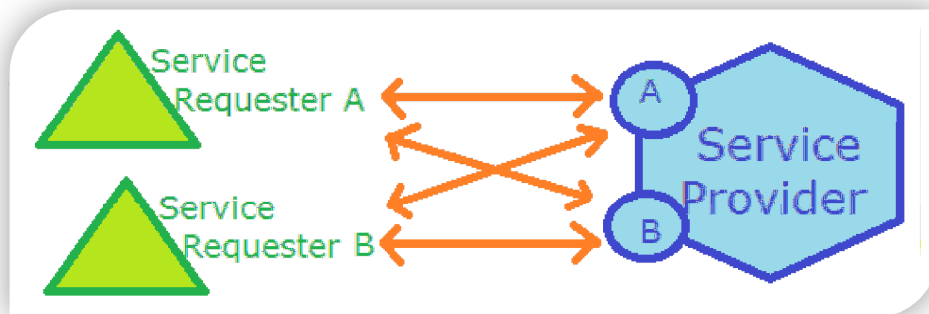


Figure 4 - Basic scheme of SOA architecture (Service provider and Service requesters)

In other hand, we have two kinds of services: simple and composite (Papazoglou, 2003). Simple services are the ones that take care of one particular action within the service provider. These simple services can be used independently or reused to make a composite service. Thus, a service requester can either make a simple request or a composite request. This also follows the idea how structured the XML is: an XML has always a root element. If it is a simple request, the root of this element is simply the name of the service. If it is a composite request, the root of the element has to be different and incorporate the different simple (or composite) sub-requests. We can see how the XML is structured in a simple and composite request in the Figure 5 below.

There is another component intrinsic to SOA: the service registry. The service registry is where the service providers are going to store the services information for the service requesters. Before a service requester “connects” to a service provider, it should first find the service it is looking for. This service registry does not need to be stored in the same place where the service provider is running. This allows even more flexibility to the service providers to distribute the different services across different machines. In the end, when the service requester asks for a service, he retrieves the address where that service is running.

Alternatively to this service registry, it can be made a service tunneling. In this way, the service requester sends a request directly to a main entry point of the service provider, but afterwards this one redirects the call to the location of the service. This is similar to a communication via proxy.

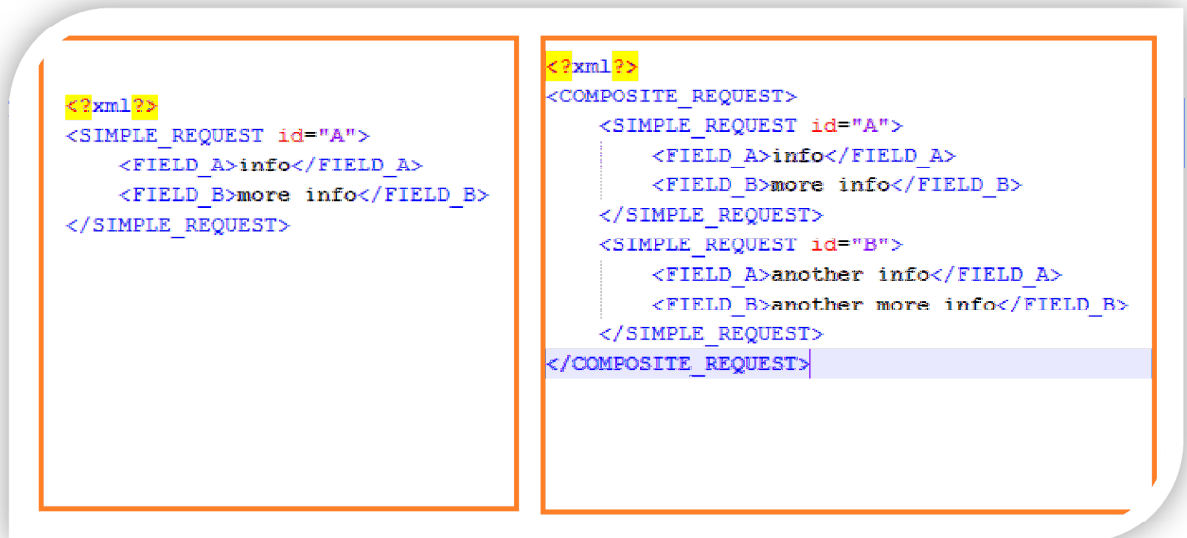


Figure 5 - XML scheme of a simple request (left) and a composite request (right).

Hereafter, SOA architecture is suitable to be implemented in any kind of solution. It provides a well-structured and flexible way of doing software and it allows the software to be constructed based on services. Thus, it has all the sense to use SOA in the *SafeHouse* system. But before progressing into details of the components of the system, it is crucial to introduce the existing system within the household market and research field.

3. Existing Household Systems

The concept of Home Automation has existed for many years. In different conferences research projects have been presented and in expositions many prototypes have been shown to people in general. The trend is for these systems to become simpler. Associated with this trend was the expansion of wireless systems and mobile phones. Nevertheless, the market is predominantly targeted at upper class people (Meyer & Schulze, 2006). A cheap alternative will be able to tap into a much larger customer group.

Also, there are different motivations for a household system as the reader can see in the figure 6 below. Among all the motivations, certainly the most important for users is the security. This is explained by the fact that the urban threats in different countries are becoming higher.

There are many standards for home automation devices. Yet most devices and systems cannot communicate between themselves (Warner, 2009). This requires skilled professionals to configure this communication between these devices. Then again this configuration of such a system will require further configuration if the user wishes to modify his/her system. A possible solution to tap this fact would be to develop a SOA system for controlling smart homes. This kind of architecture has the flexibility of quick changes and to be “plug-n-play”, as referred in the previous chapter.



Figure 6 - User motivations for installing smart home products (Meyer & Schulze, 2006).

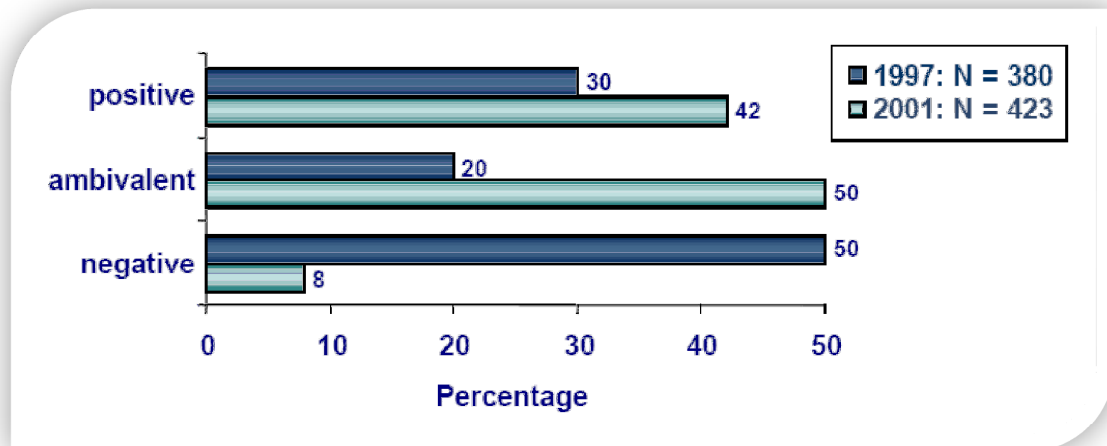


Figure 7 - User acceptance of smart home systems (Meyer & Schulze, 2006).

Below, a list some negative arguments and problems with smart homes according to (Meyer & Schulze, 2006) is shown:

- ✗ Smart homes are too expensive.
- ✗ Installation problems, no plug-and-play.
- ✗ Programming is too complicated.
- ✗ High repair costs.
- ✗ Privacy issues.
- ✗ Dominance of technology.
- ✗ Access rights, security issues.

Contrary to these problems there are some guidelines of what a smart home should be according to (Meyer & Schulze, 2006):

- ✓ Easy to use
- ✓ Understandable
- ✓ Affordable
- ✓ Easy to install, plug-and-play
- ✓ Easily expandable
- ✓ Time saving
- ✓ Ensure privacy and security
- ✓ Energy Saving

Problems still exist with smart home systems. This project aims to develop a solution that solves as many of these problems and, at the same time, adheres to the general guidelines of what a smart home should be.

One of the most known standards is the X10 standard. It is very popular and most (older) systems use it (Burroughs, 2010). We will look at X10 in more detail later in Section 5. There are many others such as Echelon Lonworks, and National Instruments LabVIEW modules, all of

which offer the possibility to add internet servers for remote web monitoring and control of a home. However, a trend is seen in newer systems which are in favoring more to use the newer wireless technologies. In industry many companies exist that provide home automation solutions. The problem is, not only the platforms are different from company to company, but also the hardware many times is proprietary and so, incompatible for other kind of products.

A study published by (Adão, Antunes, & Grilo, 2008) is of relevance to this thesis. Their project aims at developing an online system that connects to a user's currently existing alarm system and allows the user to view the current status of the security system via TCP/IP. The user would also be able to arm and disarm his/her security system remotely from any computer or mobile device with an internet connection. Their problem formulation was that despite the existing companies there was a need to re-use current fully functional home systems instead of replacing them. In their study they created a "micro web-server" from a Modtronix SBC45EC board. The board contained a PIC18F452 microcontroller. Although their system is specifically designed to incorporate older security system it still has relevance to the proposed system because their general architecture can be used as an idea to the architecture of this project.

(Khiyal, 2009) proposes a solution for controlling home appliances and providing home security using the Short Message Service (SMS) protocol and wireless technology. The Home Application Control System is divided into 2 sub-sections all controlled from a local computer. One sub-section is for appliance control and the other is security control. A GSM modem acts as a mobile server and communicates with the remote user via the SMS protocol. The following picture shows their systems architecture.

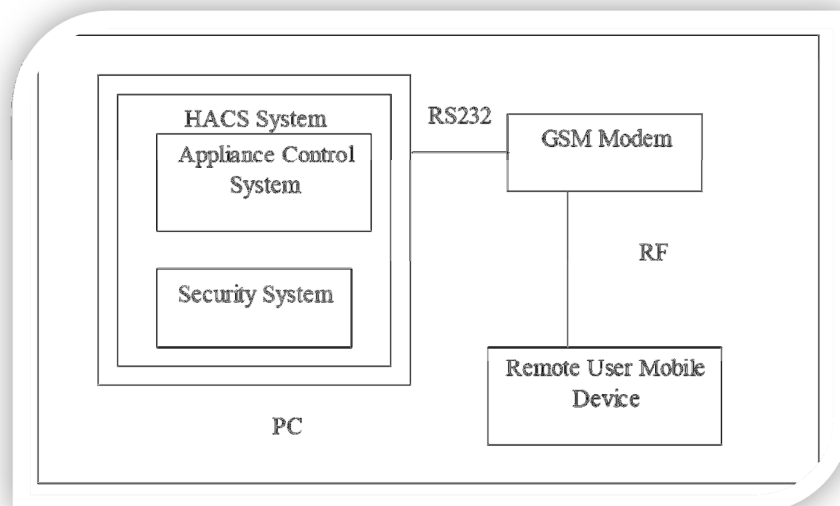


Figure 8 – System architecture based on a SMS protocol to interact with a remote user (Khiyal, 2009).

The paper however does not mention how communication is achieved between the PC and the appliances or the security system. However it is interesting to note they chose to use the SMS protocol to allow remote access.

Another solution is proposed by (Alkar & Buhur, 2005). Their system requires a local computer to act as user interface, database and webserver. The system requires the use of a master node that communicates between the computer and the other nodes. The master node communicates with the local PC via cable (RS 232) and uses Radio Frequency 433MHz to communicate with the slave nodes; this is shown in Figure 9 below. As we are going to see, similar architecture was thought in this project. However, a disadvantage to this set up is that all reliance is placed on the master node; if it fails all other nodes will cease to function. In (Alkar & Buhur, 2005) a SSL algorithm is used in order to ensure security on the webserver in addition to a user login and site certificate. This project used a PIC16F877 microcontroller.

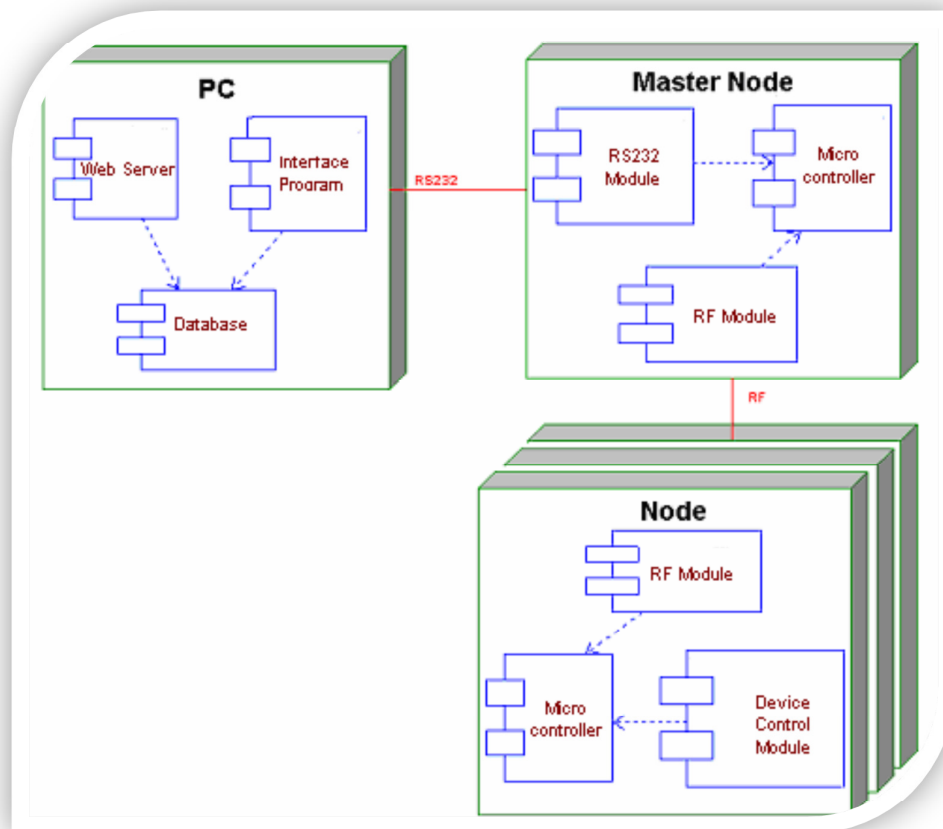


Figure 9 - System architecture proposed by (Alkar & Buhur, 2005).

A limitation of this system is in the addition of new devices to the home network. Currently devices are manually added by the user through the user interface. A simple handshaking protocol is used to do this. Here there is room for improvement. Value can be added to the existing product by automating this process. This will alleviate the user's responsibilities and make the system more plug-and-play. This author reported a penetration range of 100 meters of the RF medium in a concrete building and 200 meters in open ground. Another problem presented in this project is the failure of the system when power to the wall socket is terminated. This problem exists in all systems that require electricity, including the *SafeHouse* system. Therefore, it can be used a back-up battery or an emergency generator, but these

solutions do not give instantaneous power supply when the main source shuts down. The best approach for this inconvenient would be an UPS (uninterruptable power supply) where there is one or more attached batteries to the main energy source and giving time to reestablish the main power line.

In (Burroughs, 2010) the X10 wired approach is used in conjunction with the PIC16F877 microcontroller. Although it is not intend to use this wired approach it is still useful to note that the PIC series of microcontrollers are well suited and popular for home automation applications. This means support and maintenance should be widely distributed and available.

According to (Warmer, 2009) a service oriented approach which is event driven with web services is the best solution for a smart home system. The advantage of using web services is that different systems may vary in design and can function independently on different platforms, while still interacting with each other on a high level standard. Using a SOA approach allows services to support many various applications. Most importantly SOA approach allows autonomy, for the customer and even the device. Autonomy by definition means self-governance. It is used here in the sense that the smart house system can regulate itself and is independent of external forces. That is to say the home owner has full control of the system. This eases legal issues as well.

2.1. Conclusion

In the previous sub sections we have seen different companies and products in the market. Either through built in panels in the house or through remote applications over IP or GSM, all of them are intended for the same goal: give the user the power of controlling his house, either indoors or/and outdoors. Probably the tendency will be to turn the houses more automated, independent of user actions, but that control can be mistrusted from the user point of view. In the design part, we are going to consider these different solutions presented for the development of this master thesis work.

Now that it has been presented briefly the SOA architecture and some existing systems, we are going through in more details to each component involved in the system, starting with the embedded devices.

3. Embedded Devices

As referred in the beginning of this report, sensors and actuators are everywhere. In fact, they are incorporated on the so called embedded devices. This kind of devices is everywhere: in our washing machine, in our refrigerator, in our television, in our internet modem and what so ever. We do not even realize the importance that they have in our daily life and without them, most of our daily tasks would be hard to concretize. Let us just imagine what would be an oven without an embedded device to control the temperature inside: the oven would be so hot that would put our life in dangerous just in a matter of minutes. This is just simple example, but reflects how dependent we are from these useful devices.

Embedded devices have typically a microcontroller or a digital signal processor in their core. Moreover, they are ideal to perform a specific task with low power consumption. An interesting solution is the **open sensor board (OSB)** from Aalborg University and Technical University of Berlin (Department of Electronic Systems, 2009), which provides an open source sensor hardware platform for development and integration with different kind of sensors/actuators. Hereafter, it is introduced the platform that is going to be used in this project:

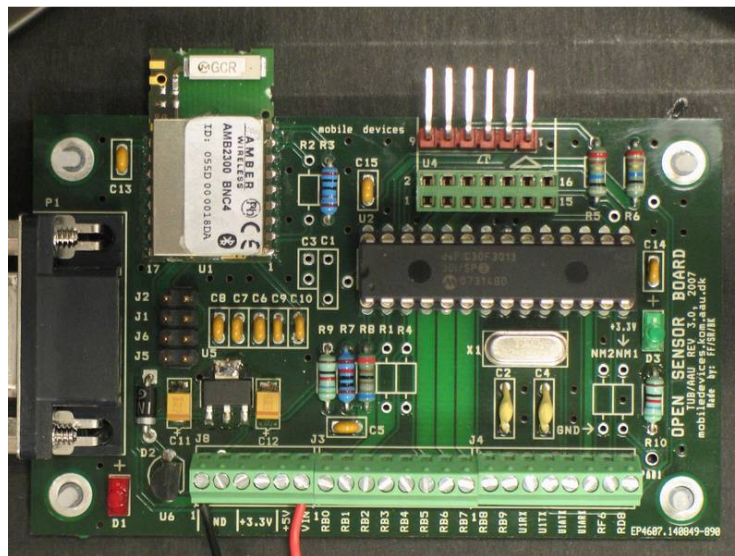


Figure 10 - Open sensor board with pic30f3013 and Bluetooth UART.

Furthermore, it has been provided from the E-Lab at Aalborg University two different OSB with different sensors and actuators. In the following sections it is introduced each kind of component used in the OSBs.

3.1. Distance Sensor

A distance sensor is very useful in many situations. A concrete example of a distance sensor in our daily life is the parking sensor in recent automobiles. This device gives an alarm each time that the driver approaches a car with reverse gear. The alarm gets louder and more repetitive as closer as the cars are from each other. In a smart house, a distance sensor can be used in many situations: when someone gets close to a safe lock where we have valuable things; when someone enters in a room or in the house; etc. For the effect it is going to be used a Sharp GD2D12 (Acroname, 2010). This device gives as output a non-linear analog signal for distances between 10 and 80 cm. Since the PIC micro-controller has an analog-to-digital converter (ADC), the problem addresses the fact that distance sensor gives a non-linear signal. To better understand this, let us take a look in the following graph:

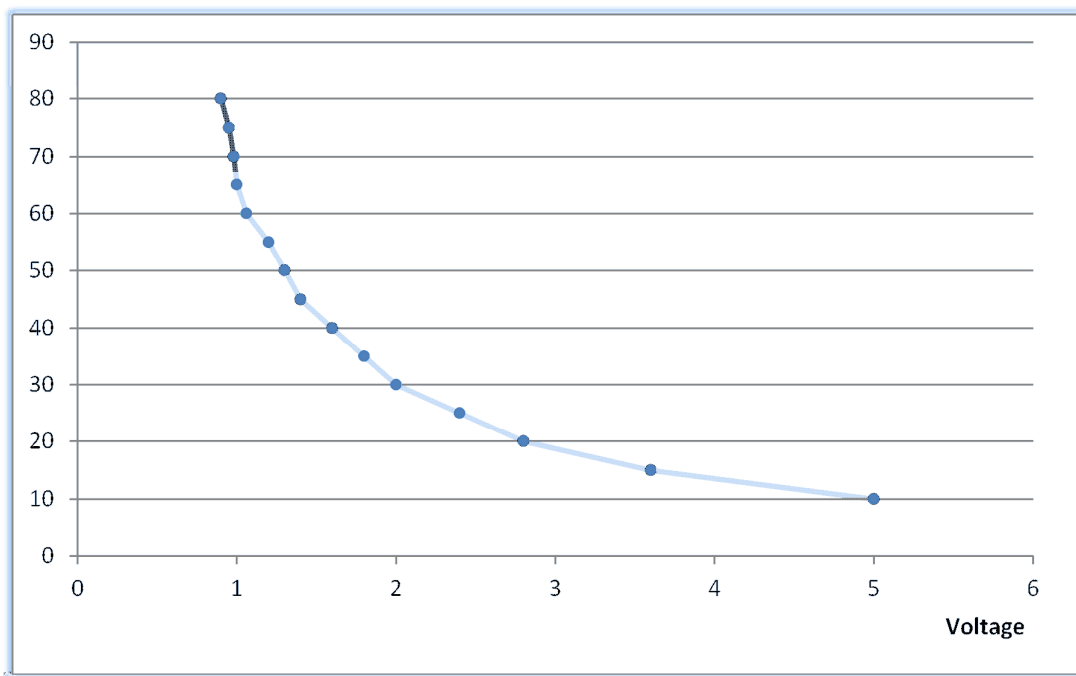


Figure 11 - Sharp GP2D12 Voltage vs Distance (in cm) graph.

As the reader can see, the graph is not easy to understand in terms of output (voltage). Instead of making a case-by-case interpretation of the signal, i.e. if we get 3V means one thing or 1V means a different thing; it was made a linearization of the signal using 1st grade Linear Regression. However, if we simply apply a linear regression to the entire graph, we obtain huge deviations from the real output. For more detailed information, the reader can see in more detail the Appendix A.

Instead of applying a linear regression over the entire graph, the graph is divided into three sections: voltage between 0.9 and 1.4 Volts; voltage between 1.4 and 2.4 Volts; and voltage between 2.4 and 5.0 Volts. Finally, a linear regression is applied to each section and we obtain the graph represented in the Figure 12 below.

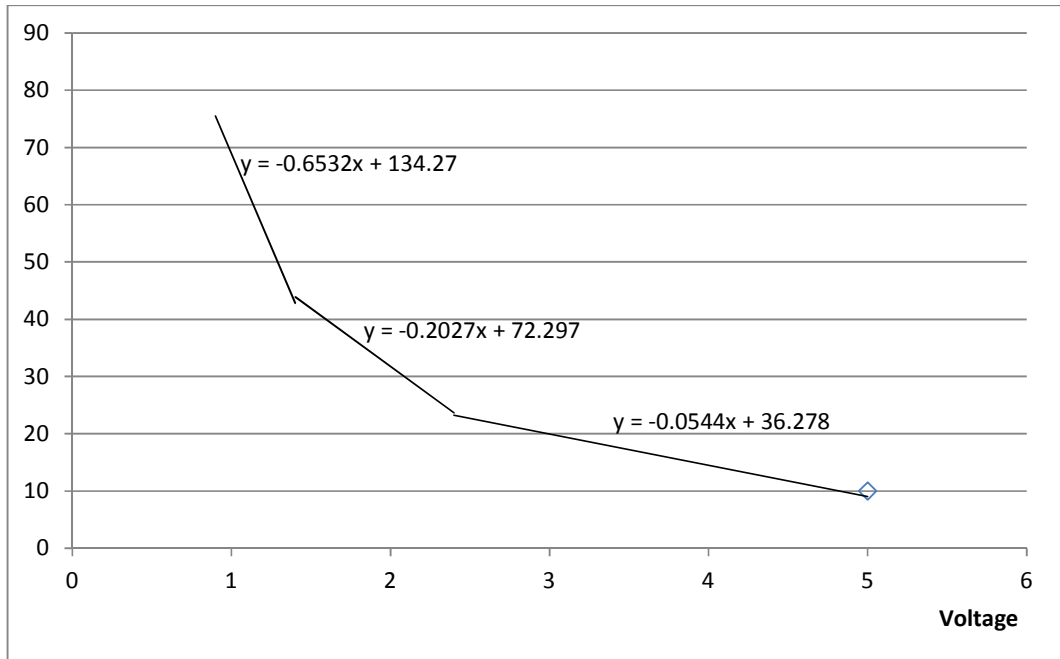


Figure 12 - Linearization of the GP2D12 signal (volts vs cm).

If we apply the formulas above, we can have a comparison between the real distance values and the linear ones.

Comparing the real values and this approximation, we can see that the error margin is low enough to be considered in this project. Let us remember that the primary goal with the distance sensor is to detect “someone nearby something” and even a maximum difference of 5cm (instead of 60cm it will give 65cm) will not make a big difference in a smart house implementation. In other hand, we cannot use this linearization in a parking mechanism for the obvious reasons mentioned above. In general distance sensors are always giving different results depending on the room temperature and supply voltage. However, in most of the cases it is useful, since the operating temperature of this component is between -10 and 60 degrees Celsius.

Another idea could be to make a 2nd grade linearization. With this, we would obtain an accurate approximation of the results with the real values. Nevertheless, this solution is not sustainable for a micro controller, which does not have a powerful processor and will take time to make that calculation. Finally, the values can be stored in a table, which will be quick but very sensible to changes, i.e. if the distance sensor is changed by other with different characteristics, it has to be stored the new values again.

A last approach would be to simply set the distance sensor to **true** or **false**, meaning that the value is true if someone is nearby (in the range of the sensor) and false otherwise.

In the next part of this chapter we are going through about another important component in our system: the temperature sensor.

3.2. Temperature Sensor

A temperature sensor is perhaps one of the most useful that we can have anywhere. It is used everywhere without realizing its presence there. Indeed, it is one of the sensors most spread all over the world. In the project *SafeHouse*, it is going to be used the LM35DZ (Corporation, 2010).



Figure 13 - The temperature sensor LM35DZ.

Unlike the distance sensor, this temperature sensor gives a linear response, more precisely 10mV per degree Celsius, working in temperatures between 0°C and 50°C. In the following graphic is represented the relationship between temperature and voltage. As the reader can see it is fairly straight forward to obtain the temperature.

In the next chapter we are going through other sensory implemented on the open sensor board, concretely a green led, a buzzer and a magnet sensor.

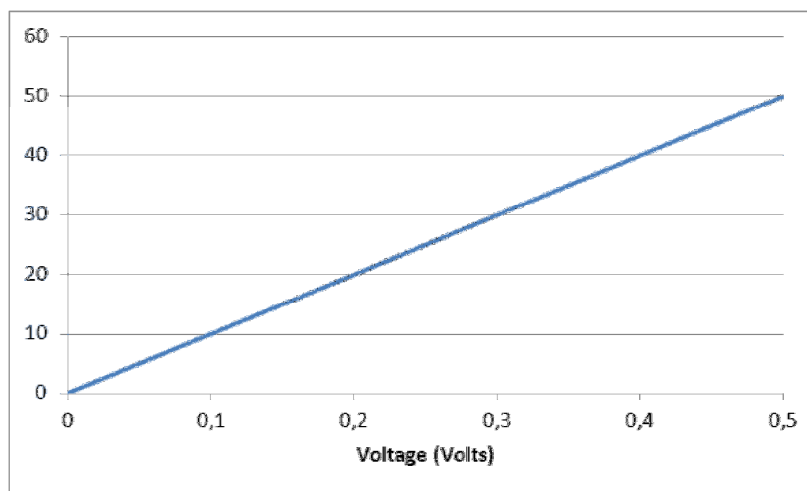


Figure 14 - Relationship between voltage and temperature (°C) in the LM35DZ.

3.3. Other Sensors

Besides the temperature and distance sensors, it was provided other components. One of them is the magnet sensor. This component gives a high-low output signal, meaning that the signal is in low output when a metal is in contact with it and in high output otherwise. This magnet sensor can be used near entrances, such as doors or windows, to detect if that entrance is open or close.

In order to simulate a normal light bulb and an alarm, it has been used a small green led and a small buzzer. The reason is simple: the intention of this project is to give a proof of concept of a smart home system.

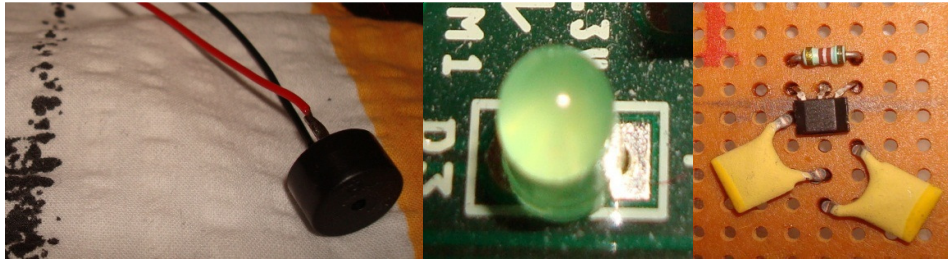


Figure 15 - Other components. From the left to the right: a buzzer, a green led and a magnet sensor.

In the next chapter, we take a step further and it is introduced mobile devices, where it is intended to implement a user-friendly and quick application to access the house.

4. Mobile Devices

It is well known that mobile devices are becoming increasingly powerful. In the last years, the barrier between mobile phones and computers had become really narrow. It is possible to develop applications for mobile devices as we do for computers and the operating systems for mobile phones have increased in number and robustness. In this way, it makes sense to think about a collaborative platform between cell phones and other devices around them, particularly to access our house and control it. Nevertheless, one of the main limitations of the mobile devices is exactly the different platforms that we can find for them.

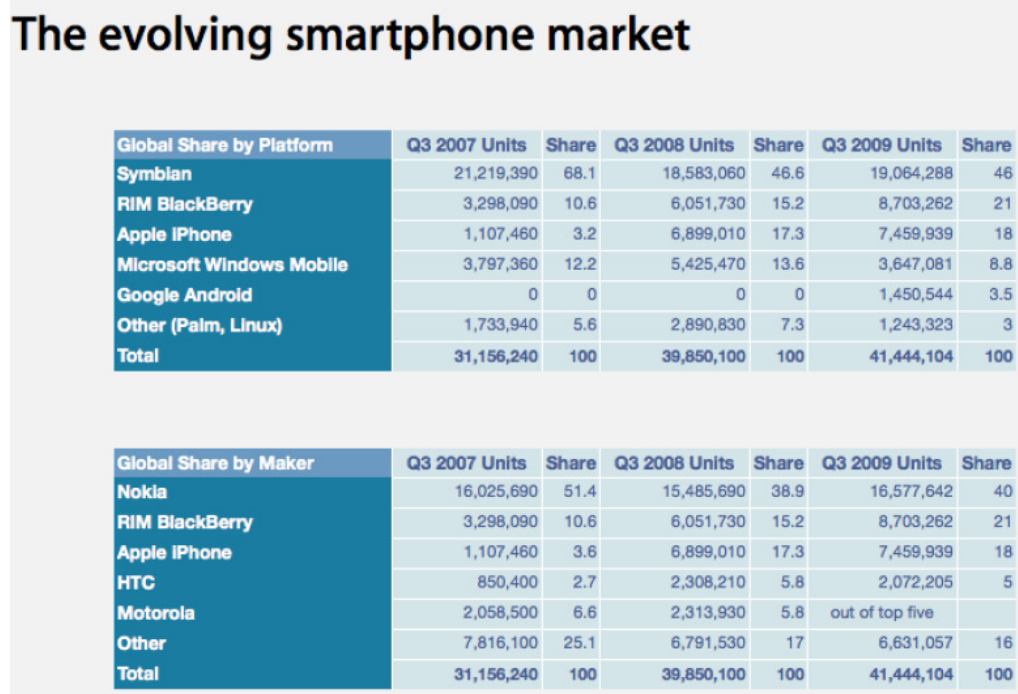


Figure 16 - The evolving smartphone market from 2007 till 2009 (McLean, 2009).

In one hand, we have the worldwide spread platform to most of the Nokia devices – Symbian™ - in its most popular versions: S40 and S60. In other hand, we have one operating system per manufacturer, what does not help to choose the platform to work with. Besides, a developing team has to concentrate effort in learning each platform instead of focusing in the development itself. Soon, the need for a hardware-independent framework was needed. In this matter, Sun© has integrated a Java virtual machine into mobile devices, the so called MIDP (Mobile Information Device Profile) or JSR 118 (Group, 2002). This virtual machine was one of the first attempts to unify the development for mobile devices. Because of this, Java comes naturally as a good choice for mobile development.

In other hand, the world-wide corporation Google© has launched recently its first operating systems for smart phones, the Android OS. Following-up the launching of the operating systems, Google also has launched the Android SDK, a Java-based framework for Android OS development (Xuguang, 2009). With this comes another advantage for programming in Java.

Nevertheless, the SDK is still in development and thus, it was decided to put this option apart. Besides, the android devices available were limited to few numbers at the time of this thesis. Thus, the choice tended for the development in Nokia.

For Nokia devices with Symbian S60 operating system (Nokia Corporation, 2008), there are different options for the development: the native Symbian C++ (Pérez, 2007), the brand new Qt framework (Forum Nokia, 2010), the J2ME (Java 2 Micro Edition) and Python for S60 (Nokia Corporation, 2008).

If a developer wants to go deep in the capabilities of his mobile device, he should for sure user Symbian C++. With this language, it is sure that the developer can do anything with his mobile device, respecting the device's limitations, obviously. Nevertheless, with the use of a more powerful language we lose portability of code, meaning that if we develop an application for Symbian S60 5th edition, perhaps problems would arise porting to a S60 3rd edition. Another disadvantage is the time to develop an application for Symbian C++. Generally speaking, we have to be more aware of device details, instead of concentrating in the core components of the application. For instance, in the case of this thesis, this would become unsustainable in terms of time.

Another option could have been the brand new Qt platform. This platform is a way of Nokia giving a simpler access to the device functionalities. With this platform, the time of programming is reduced, comparing to the native Symbian C++, since it avoids a lot of complicated details regarding the GUI rendering. At the time of this project, Nokia was still launching platform components and documentation. Because of that, it has been decided to not explore a completely new platform.

In another hand, we have Python. Python is one of the easiest languages to use in programming. There are uncountable frameworks and development kits based on Python and no exception goes for mobile devices. There is the so called pyS60 in different flavors depending on the edition of the Symbian. With python we can have an application in a really quick time and with not so much effort. In other hand, as referred above, python comes in different flavors, meaning that for each edition of Symbian we have to use a different development kit. This is a big disadvantage when considering porting your application for a wide-range of devices.

Finally, we have J2ME. This platform uses Java as programming language and it has the advantages and disadvantages of a high-level programming language. In one hand, the development of code for a high-level language has proven to be faster. A prove of this fact is the job trends comparing the four different platforms introduced (Figure 17 below). An obvious disadvantage is that with Java you do not have access to low-level resources, but normally this situation can be over turned in some way. Nevertheless, the main advantage of Java goes exactly for the range of Java enabled devices (Regnier). An extra argument goes for the fact that Java as already known by the development team of this project.

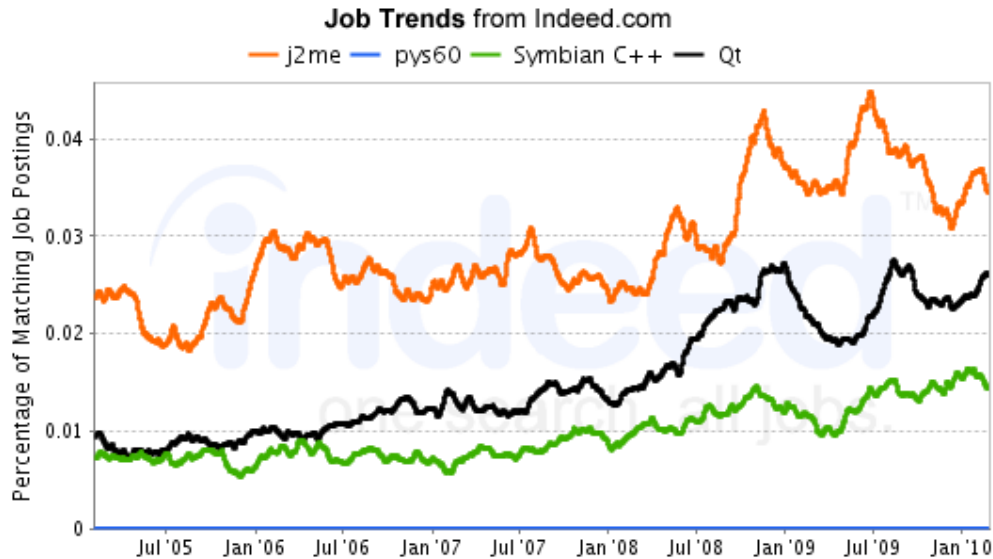


Figure 17 - Job trends for different mobile development platforms (Indeed, 2010).

4.1. Conclusion

Considering what it has been discussed before, the decision was to use J2ME. First of all, the development team felt more comfortable in using Java as programming language and with the framework itself. Secondly, it presents a good balance between GUI control and performance, meaning that it is quite simple to make a user interface and at the same the application runs fast. Last but not the least, it has been chosen this platform because there were already useful libraries developed for J2ME during the internship of Sérgio Pedro in PDM Technology with the permission of Jørgen Schiønning, CEO of PDM Technology. The reader can check which libraries are being used in Appendix B.

Another main advantage of using J2ME instead of pys60 or one of the C++ platforms was that J2ME gives the portability of code across platforms. There are plenty of manufacturers implementing the Java Virtual Machine on their devices and with this, not only Nokia devices are being targeted, but yet a wide-range of manufacturers. The Figure 17 above proves also that fact, even though the Qt platform is increasing in job postings, mainly because it is a new platform and there are few professionals with experience on it.

At this point, we have introduced our initial idea for our system and presented some of the existing systems. Hereafter, we have introduced a flexible and elegant architecture (SOA) and introduced the most important devices in our system: embedded devices and mobile devices. In the next section we are going through how the system would communicate within components, i.e. which wireless protocols we have available and which ones are more suitable for our thesis work.

5. Communication protocols

Since the internet has given its first steps in beginning of the 70'ies many things have evolved and how we connect between each other has also changed. Moreover, in the last decade we have seen a great and expansive evolution of wireless communications. Indeed, the wireless communications have made a revolution in our life and in the internet itself.

Since the development of the Internet, different models were using to describe it. The TCP/IP model (Tanenbaum, 2003) describes the communication protocol as divided into different layers, from the Application Layer to the Physical Layer. In the middle of these layers comes the IP protocol (Network Layer). The IP, independently of the lower layers, is the basic carrier for all the communication protocols. Thus, it is fundamental to say that any research made on this field has as basis the Internet Protocol.

In this master thesis, the communication protocols are divided into different sections: by type (wired or wireless) and by range (short and long range). In this section we are going to discuss different types of protocols and in the end it is made a final conclusion about them and which one(s) it has been decided to use.

5.1. X10 Industry Standard (Wired communication)

No project on home automation would be complete without mentioning X10 (Burroughs, 2010). The X10 standard was developed in 1975 in Scotland by Pico Electronics of Glenrothes, in order to facilitate communication between electronic devices in home automation. It works using the currently existing home wiring or through radio communications. This was the first technology of its kind and over the years higher bandwidth technologies emerged; most of which too use the homes existing wiring. Communication is achieved by modulating a pulse over the wiring from one connected device to another. The presence of a pulse indicates a binary 1 and absence a 0. Despite the newer faster technologies on the market today the X10 standard is still popular because it is wide spread, inexpensive and parts are available worldwide. All of which, no doubt, are characteristics of a formula for success.

A huge advantage of using such a system is that no addition wiring is needed, except for new hardware. However, there are certain significant drawbacks to this and similar but faster standards. Among the disadvantages, there are:

- ✘ Takes approximately 1 sec to transmit command data.
- ✘ It has poor propagation in split-phase electrical distribution.
- ✘ Affected by line noise from other devices.
- ✘ Receives interference from other X10 signals in the building.

Thus, X-10 has some lack of security in it, meaning that close neighbors using X-10 may interfere or even control each other. This is a main disadvantage for a smart home system that pretends to be safe for the user.

It is also an old technology and, if trends follow the way they have been, will soon be replaced by wireless technology. Thus, a wireless protocol is more suitable for the goal of this thesis.

5.2. Radio Frequency Identification – RFID Standard (Wireless communication)

Among the wireless communications, we have the RFID (Weis, 2003). This technology has many different purposes, but one of the most known is to identify objects, i.e. replacing the old fashion barcode. RFID systems are composed by three different components:

- The transponder, which carries the object identifying data, i.e. this component carries all the information belonging to one specific product.
- The transceiver, which reads and writes tag data. It means that this component can retrieve information from the transponder or even change that information.
- Records database, where information about the transponders is written. For instance, before the transceiver approves a certain transponder, it has to check first this database.

The RFID tag can transmit data up to 1000m, depending on its type. For instance, the most typical and cheap type is the passive one, which has a range of 10m. This one is completely “turned off” in absence of a transceiver. Also, this radio frequency has the advantage of passing through walls, since it is a low frequency signal. However, the RFID require to be designed for a specific application and they are sensible to environmental changes. Also, the mobile devices and the computer provided for this project does not incorporate a RFID module, which will limit the usage of RFID in this project. Mainly because of this reason it has been decided to not use RFID and search other solution.

5.3. Wireless Personal Area Networks (WPANs)

The WPANs are intended for communication in short range. Among different standards, there are three familiar standards: Bluetooth, Wi-Fi (IEEE 802.11) and Zigbee (IEEE 802.15.4).

Bluetooth (Klingsheim, 2004) is a wide-spread technology and present in the most of the mobile devices. Indeed, it has made its own revolution within the short range radio frequencies. First, it transmits at reasonable speed for short range, up to 721 Kb per second. Secondly, it is reasonable cheap, allowing many manufacturers of computers and cell phones to implement one Bluetooth chip on the device. Besides, the Bluetooth is organized into pico-

cells. Each pico-cell can have 8 devices connected with a “master”, but obviously one of the devices can be in other pico-cell and thus allowing a Bluetooth network to be implemented. One of the main issues with Bluetooth in its first version (BT 1.0) was the range: 10m. Currently, the most spread and used version of Bluetooth is the 2.0 that has up to 100m, even though the versions 3.0 and 4.0 are already developed. The most promising characteristic of the latest is the ultra-low power consumption.

One of the technologies promising to replace Bluetooth is Zigbee , which is affirmed to be cheaper and simpler than Bluetooth. It is the same goal as Bluetooth, i.e. Zigbee is intended to be a wireless personal area network, targeting radio-frequency applications. Thus, Zigbee has different applications, from smart energy applications to telecom services. Unfortunately, Zigbee could not be explored in more detail, since it is not equipped yet in most of the mobile or computer devices, as we can see in the job trends in the Figure 18 below.



Figure 18 - Job trends for Bluetooth and Zigbee (Indeed, 2010).

In other hand, Wi-Fi has revolutionized the wireless networks. It allows a user to connect to a LAN and consequently to the internet. Moreover, Wi-Fi has the advantage of being world-spread, being common for a user to catch a signal almost everywhere. In other hand, Wi-Fi has a short range of transmission, up to 30m indoors and 90m outdoors, operating in ranges between 2.4GHz and 5.0GHz.

Finally, in terms of consumption (Pering, 2006) and referring only Bluetooth and Wi-Fi, Bluetooth has a clear advantage over Wi-Fi, consuming almost 10 times less than Wi-Fi. Since the sensor boards used are battery-driven, this is a fact we have to take into account if we do not want to have a crying expensive bill in the end of the month. Hereafter, Bluetooth becomes the natural choice to use in the sensor boards, since these ones have to plug-in into the house electric grid.

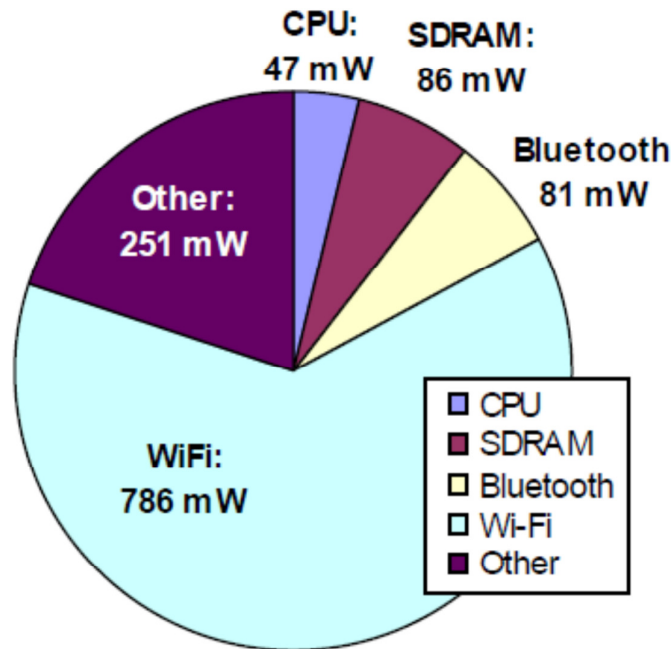


Figure 19 - Power breakdown for a connected mobile device in idle mode (Pering, 2006).

Either Wi-Fi or Bluetooth do not allow a user to access his house from anywhere in the world. This only can be achieved using GSM or 3G communications, which is the standard for mobile communications. Most of the smart homes systems that allow connection with the user are based on GSM/3G standards, especially with the spreading of antennas and satellites all over the planet.

5.4. 3G Mobile Communications (Wireless Communication)

As we have said before, 3G allows a device, mobile or computer, to connect to the internet from anywhere in the world, as long as there is an antenna and network provider signal. Because of being the standard in mobile communications, it has been decided to use for outdoor communication with the user's house. However, the use of 3G should be strictly limited to outdoors, since it has extra costs to the user, unless the user has a 3G package with his network provider.

Moreover, 3G is faster than the old generations of mobile communication such as GSM. The data rate of transmission is up to 5 Mbit/s, what is quite fast if the user wants to access his house remotely.

5.5. Conclusion

Let us remember that in this master thesis work, there are 3 main components: embedded devices, house entry point (a desktop server) and the mobile device. There are also three different standards it has been decided to use: Bluetooth, 3G and Wi-Fi. Bluetooth will be the chosen standard for the communication between the entry point and sensor boards. Bluetooth wins over Wi-Fi mainly because of the consumption. The Wi-Fi will be used indoors whenever the user has a WLAN access point and he wants to use it. Finally, the 3G will be used when the user is somewhere without other Internet access point.

During this analysis section we have been through all the topics of the *SafeHouse* project: it has been introduced the system, SOA and existing systems. After, it has been referred embedded and mobile devices and finally this last section about communication protocols. Hence, the research is in place to make the problem formulation of this project.

6. Problem formulation

In Section 2 (page 19), we went through an important architecture, the SOA architecture. This architecture allows this project to be composed in a software system based on services. In the case of controlling a house, we would say that each service will represent an action that can be taken in the house, specifically on the micro controllers. The micro controller will, as well, interpret that action and dispatch to the sensor/actuator requested. Thus, it is important to define that all of this system is based on SOA architecture; it does not matter if we are talking about an embedded device, a mobile phone or a desktop application.

Secondly, our system will be based on a three-point communication: the mobile application; the house entry point access; and the embedded device.

With consideration with these facts, it has been made the following problem formulation: it does not matter where and when the user is, he should access his house and make quick actions in a mobile phone according to his will. Moreover, at home the user should have a wider overview of his house through a desktop application, visualizing logs, creating or deleting other users, rules or notifications to be sent. *SafeHouse* should give to the user the power of control and change the system in his own way without requiring any expertise knowledge.

Hereafter, there are some requirements that the *SafeHouse* should fulfill in order to achieve a complete result:

- The system should be plug-n-play in a sense that every embedded device, after being programmed by the development team, should be detected immediately and no further configuration should be made in the characteristics of the components.
- A desktop application providing to the user important functionalities in order to give him/her the control of his/her house:
 - o automate the house by allowing the user to create rules;
 - o update the user by allowing the user to create notifications;
 - o give access to more users by allowing the user to create new users and give them permissions;
 - o provide an easy way to customize names of the devices;
 - o give the option to check logs of the system;
- A mobile application with a simple interface where the user can change/check in a glance the status of some component.

Therefore, we are now in conditions to go further and explain the design of this system in order to achieve these requirements.

III. Design

In the previous chapter different topics have been discussed, from Service Oriented Architecture to Communication Protocols. It was analyzed different systems and technologies important for this master thesis. Therefore, the problem formulation was constructed based on the research made during the analysis.

Hence, this chapter will start to refer to the **Architectural Choice** used in this project: starting with the initial idea and ending with the final choice and which advantages and disadvantages come from each solution proposed. Following this idea, it will be introduced the **Communication protocol** integrated in this solution for the interaction between each point in the system.

Hereafter, we are going through the **Structures of the system**: which elements are used in this system and for what they exist. At this point, we are going to explain into more detail the following components of the system: **Home Server**, **Desktop Application** and **Mobile Application**.

1. Architectural Choice

From an architectural point of view, different approaches have been thought to concretize this master thesis. A remark should be made that it was always intention to implement a SOA, independently of the choice made in the end. Thus, two different concepts were thought during this period: a **Remote Server** concept and a **Home Server** concept. Both solutions would concretize the idea of a smart home and both solutions are being used either in researches or in final products.

The main difference between them is the number of servers that such systems may have in the different cases: in a remote server all smart homes would be connected to the same server and would not require any public IP address in the user house; in a home server, each user would have a server at home and might have a public IP address in order to be able to receive external Internet connections. We are going through each of these solutions and in the end explain why I have chosen the home server architecture.

1.1. Remote-Server Architecture

It is true that nowadays the Internet is structured in web-applications organized in central servers, storing and sharing data from users and serving all the users in the world for different purposes. This is the trend and it will continue to be. Following this already known concept, the first idea was exactly based on Remote-Server architecture: a server running 24/7, able to provide all the web-services different users would need.

In this way each home would be connected to this remote server as the Figure 20 shows.

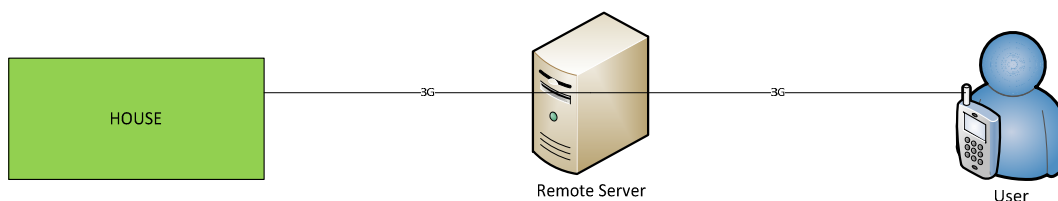


Figure 20 - Remote-Server Architecture: All the interactions between the user and the house pass through a webservice.

Hence, the user could access his/her home through the server and every event happening in the house would be sent to the remote server. Thus, the server would register all the information from both sides. It is important to say that this web server was thought to be a HTTP server, serving houses/users upon HTTP requests. Thus, this server would offer a web page where the user, through a browser in the mobile phone or computer, could see all the information and interact with the house. Besides, this architecture would need a mobile server as entry point in the house, to serve as interface between the remote server and the components in the house.

The idea behind this architecture is simple: as we have a physical door to enter in our house, we would have a digital door, the 3G modem, to control and monitor our house remotely. Besides, this modem would communicate with the user cell phone through SMS each time an important event would occur. In other way, the configuration of the system would be simple, since each house is commanded by one small device (the modem), which can be obtained by a reasonable cheap price. In the Figure 21, this concept is shown in more detail.

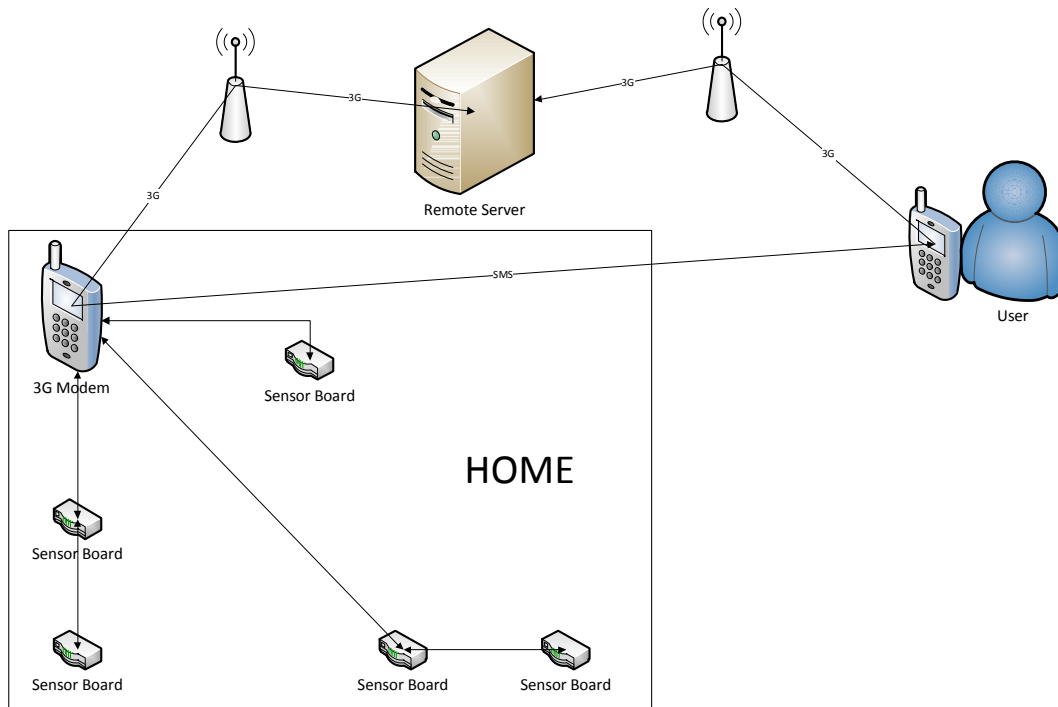


Figure 21 - The Remote Server concept in more detail.

As we can see, this architecture is divided in three main points:

- **Home:** At home the system would be commanded by a 3G modem that would interact with each sensor board through Bluetooth or other radio frequency. Also, the sensor boards at home would serve as hops to other sensor boards. An event occurred in a component attached to the sensor board would originate an action in the modem. Hence, the action would be packed into a SMS to the user cell phone and also packed into a HTTP POST to the remote server.
- **Remote Server:** The remote server would be divided into two different components: services running to serve each house and a website. Obviously, these two components would interact with each other: for instance, if the user would decide to turn off the lights remotely, this action would be registered and in the next HTTP request from the 3G modem (at home) the lights would be turn off.
- **User:** To the user would be provided a webpage where he/she could see updates from his house and at the same time control and monitor some of the components at home. Also, the user would receive a notification by SMS/email when an important event happens at home (e.g. the window becomes open during the day).

There are advantages and disadvantages in such architecture. First of all, this architecture would not require any application or installation in the user cell phone what is a big advantage. The user would simply access the browser and see live status of his house. The only thing required between 3G modem, remote server and the user would be an internet access. Also, the system would be portable between different places. For instance, if the user would change place in the meantime, it would be only required to move the equipment to the new place. It would be plug-n-play in this sense.

Nevertheless, critical disadvantages exist, either in performance or in ethics. In terms of performance, this system would not offer a fast way to access to the components when the user is at home, i.e. every request from/to the user, either at home or outside, would pass through the remote server. If, for instance, the remote server is located in United States and the user is living in Denmark and in that precise moment the user is at home, it does not make sense that to turn off the lights a HTTP request has to go from Denmark to the US and comes back to the user house in Denmark and shuts down the lights in the end. Another disadvantage belongs to the 3G modem. In general, devices integrated with 3G do not have a public IP address, meaning that it would be always the modem to start a connection with the server and this connection would have to be kept alive or the modem would have to establish a new connection to retrieve or post information. This means that there is a high dependency on the 3G modem. If this component fails, all the system fails immediately.

In terms of ethics, it could be thought by many users that such a system would compromise their privacy at home. It is true that social networks are increasingly getting world wide-spread. For instance, Facebook already counts more than 400 million users in all over the world. But one thing is to share pictures or information we want, other different thing is to provide information to a remote server about our behavior at home. As social researches prove, the user does not accept well the fact of sharing his/her behavior at home and this raises an ethic problem. However, this issue could be tapped by guaranteeing the privacy of this remote server, as it happens in net banking systems.

Hereafter, taking the advantages and tapping most of the disadvantages of such system, another solution was thought and followed. This solution is based on a home-server with public IP address. With this, the communication will be much shorter in distance to any user and no user would share his/her information to any central server.

1.2. Home-Server Architecture

This architecture taps some of the disadvantages pointed out of the remote server architecture and keeps intact the main strength of the previous architecture: give the user the power to control and to be notified whenever he is in the world. Moreover, it is possible to create a Wireless Private Area Network at home without requiring any internet access. This can be made running simultaneously a Bluetooth or other Radio Frequency server with the Internet server. In other hand, this architecture requires each user to have a public IP at home. This may have two disadvantages at the first sign: the ISPs (Internet Providers) control the public IP addresses and the number of available public IPs are becoming really limited (in IPv4). Regarding the first issue, there is always the possibility to get a server with public IP address through dynamic DNS, i.e. each time the server would start a new IP address would be attributed, keeping the same host name. In other hand, the IPv6 already exists and it is a matter of time to be fully implemented everywhere, solving the problem of limited public IP address, at least for the years to come.

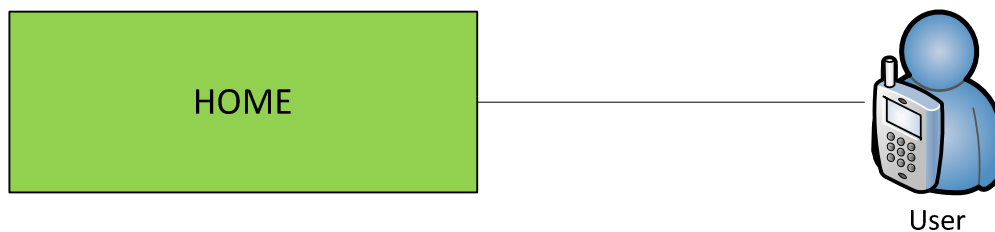


Figure 22 - Home-Server Architecture: the user interacts directly with his/her own server (placed at home) wherever he/she is.

As we can see in the Figure 22 above, the interaction between the user and his home is made directly to a server located at home and this same server keeps track of the components (sensors and actuators). The same goes for the case when the user is at home. The interaction is made first with the local server and after, between the server and the components. Therefore, this solution is much faster and more comfortable to any user. Of course, an application is needed to be installed in order to retrieve information from the sensor boards, but this application is intended to be plug-n-play, not requiring any step besides installing in the computer. Also, the notifications for the user can still be made, either through a third-party web service (e.g. Skype, VoipCheap) or through an auxiliary cell phone at home. Besides, the user needs to have a small application installed in his cell phone in order to communicate remotely to the house, but again, this application does not require any configuration from the user point of view. How the system is configured will be topic of discussion in the Implementation section.

In the Figure 23 below a more detailed infrastructure of the system is shown. For the case, three users are shown to explain the different kind of interactions that the users (assuming more than one person living per house) can have with their house. The home server is running and retrieving data in a fixed schedule from the sensor boards. The user 1 is at home and he is using exactly the desktop application provided by that home server. He has full control and

overview of the status of the system in a wide screen. The user 2 is at home but he just wants to perform a quick action in one of the components. For that he uses the mobile application and in few seconds that action occurs in the component he selected. The user 3 is at work but he wants to turn on the heating in his room to be warm when he arrives home. Instead of calling one of the inhabitants of the house to make that action, he simply accesses his mobile phone and in the same way as user 2 did, he performs an action in that component. The difference between user 2 and user 3 is the medium of communication to the home server. User 2 is using a local wireless network through Bluetooth or other radio frequency. User 3 is using 3G or Wi-Fi.

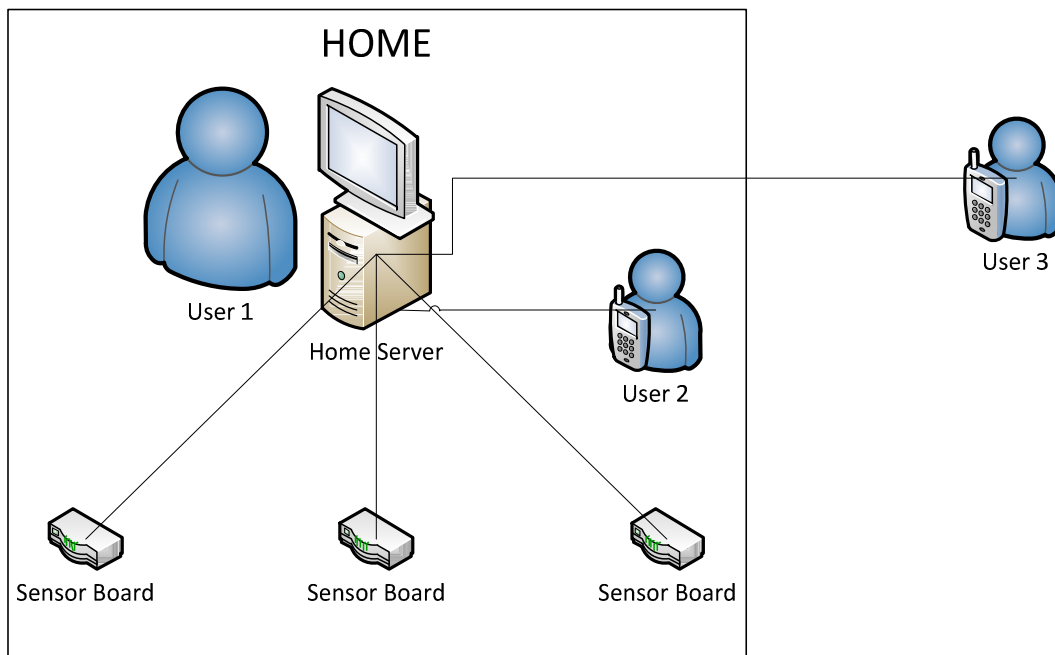


Figure 23 - Detailed scheme of the Home-Server Architecture.

At this point, the overview of the system was shown. Nevertheless, till now the architecture was more focused on a hardware point of view. For a full comprehension of the system is necessary to go into a software point of view. Thus, the interaction is based on three different layers in the system:

- **Embedded Layer:** running on each sensor board, it is responsible to listen for incoming information from the home server and perform the actions described in that information.
- **Application Layer:** running on the home server, it is responsible to communicate with mobile applications running the presentation layer. It is considered part of this layer the process responsible to retrieve and update components in the system. Moreover, parallel to this layer, it is running a process responsible to handle GUI events on the desktop computer.
- **Presentation Layer:** running on the cell phone, it is responsible to receive information from/to the application layer. This layer does not only receive data respecting to the embedded devices (sensor boards) but also receives the

description of the screen to be displayed. This gives the modularity to change the other parts of the system without affecting this one.

The most powerful argument of this type of architecture is that we do not need to configure again application layer or presentation layer if the user wants to change components in the embedded layer. This is achieved using XML as the format of interconnection between layers. In each XML, all the services that should run are described with the required arguments to run them. Moreover, if one the layers is updated it is straight forward to add that new information to the XML and this information to be interpreted by the other layer. Thus, some change in one of the layers does not affect the rest of the system. This concept of SOA is being adopted by many enterprises in the Business Process Management with successful results.

Obviously, the embedded layer has to be configured again, but that is a small change that does not affect the rest of the system. In other hand, each of these layers is running on a service-based. For instance, the application layer has more than one service that can be run, each responsible to handle an event from the presentation layer. The Figure 24 below represents this three-layers scheme.

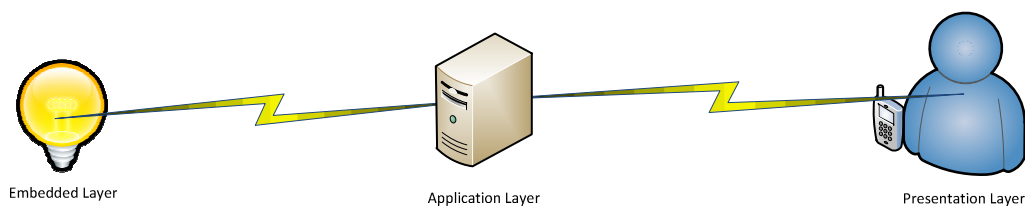


Figure 24 - Three layers architecture of the SafeHouse system: the lamp on the left represents the embedded layer; the server in the middle represents the application layer; and the user on the right represents the presentation layer.

In the next sub chapters we are going through each of these layers.

1.2.1. Embedded Layer

As it was explained before, the embedded layer is responsible to receive the information from the home server, make the changes in the components or retrieve the status and send back an answer. Thus, the cycle of operation in the embedded layer is as it follows in Figure 25. As remark I would like to refer that the general structure of each layer is the same with small nuances. This happens because the communication protocol developed allows it. The communication protocol will be explained in more detail in the section 2 of this chapter.

It starts with a **hardware interruption** when the first byte is received from the home server. After this, the application starts to **receive the data** till the end of the buffer. Hereafter, it is constructed a string with the data received and it is sent to the **xml parser**. In the parser, the string is transformed into a structure with the different xml nodes and with all the values put into fields. It is now the time to dispatch the action contained in the XML. This is made by the **dispatcher**, where finally the tags in the xml will be analyzed and it will perform an action

according to those actions. Finally, in a reverse order, **an answer is constructed and sent** to the home server.

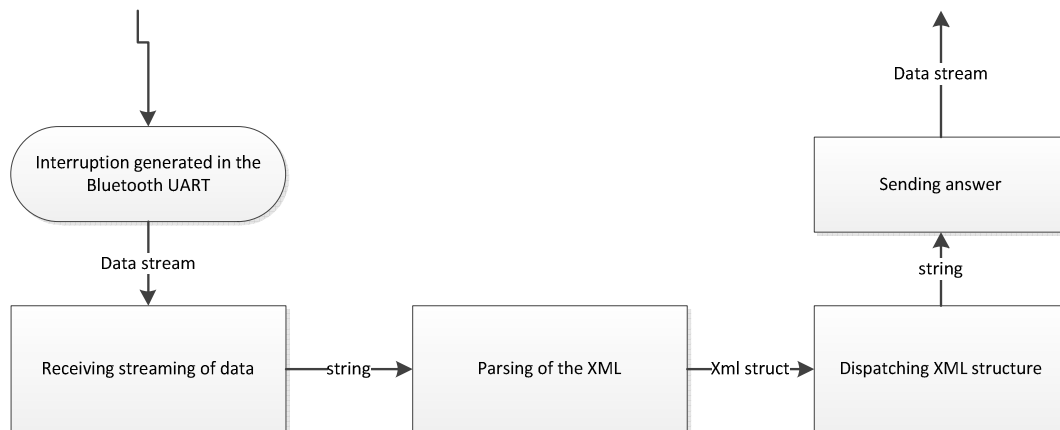


Figure 25 - The cycle of operation in the embedded layer.

Regarding the actions that can be taken in the embedded layer, we are going in more detail in the section 1.2.2. Thus, we will continue with the remaining layers of the system.

1.2.2. Application Layer

This layer is specific for each house and provides the services for the user, from his cell phone, in order to change or check status from the system. As it was said before, parallel to the application layer, it is running a process responsible for the management menu on the desktop computer. This management menu is a more advanced way for the user to change the status of the system. It allows the user to make rules in the system, create new users, visualize logs or create notifications to be sent to a mobile phone. I am going through in detail to this application in the section 5 of this chapter. For now, let us focus on the structure of this layer. This layer has two different sides: one that is constantly retrieving and updating the status of the components integrated in the embedded device and other to allow communications to/from the presentation layer running on the cell phone. In one side, the server has to have Bluetooth integrated or a Bluetooth stick to be able to communicate to the different embedded layers or to the presentation layer when this one is in a short range. In other side, the server has to have a public IP address to be able to receive connections from the presentation layer from distance.

In the middle of these two sides, there is a middle structure where each side stores the new information. This middle structure can be a database or any kind of file. For simplicity, a binary file had been chosen to store and retrieve information within the application layer. Let us see better this scenario in the Figure 26 below.

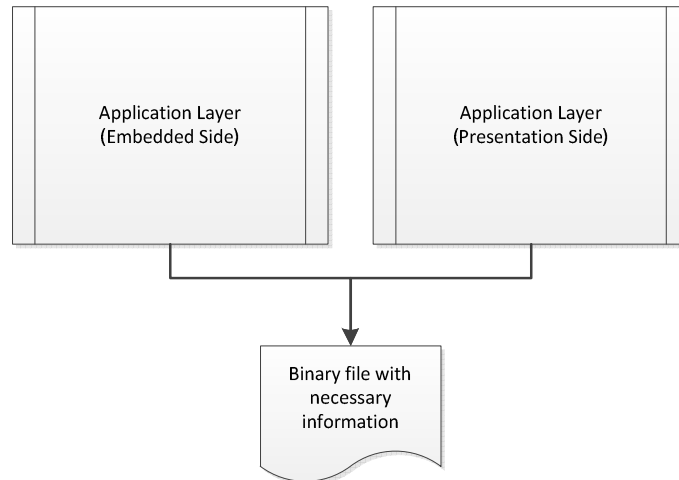


Figure 26 - Both sides of the application layer share the same resources. This is made through a binary file.

This copes with the different principles of SOA. In first place, the access to file is made in the same way by the different processes. Secondly, even if it is required to construct a database, the only thing changing is the methods to read and write from/to the database. There is no another change required in the software. Once more, these little decisions can make a big difference when a project needs to be modified. It can mean one day or one month of work in some cases.

The structure of the application layer is in everything similar to the embedded layer. However, differences exist in the structure, mainly because it is allowed a higher level language in the application layer and also the multi-threading, what does not happen with the embedded layer. Instead of explaining here in detail this structure that is in everything common to the structure of the presentation layer as well, we are going into detail in section 2.1.2 of this chapter.

For instance, two different processes are running at the same time on the application layer:

- **Mobile Presentation Side:** this service is listening for incoming connections from a mobile phone, receiving the XML, parsing it and dispatching the action to be performed. Finally it is sending an answer to the presentation layer confirming or not the validity of the XML.
- **Embedded Side:** this service is running always, looping for retrieving status from the embedded layers or/and updating their components status. Also, this service applies possible rules that the user has configured.

Finally, I would refer the presentation layer in the next sub chapter.

1.2.3. Presentation Layer

The main actor of this system is the user and to give an ease access to his house, the user has an application installed in his cell phone, able to communicate with the home server through different communication standards. Moreover, this application is not customized for any specific house, meaning that the user can communicate with different houses and not change anything in the application. The key for this to happen is exactly SOA: since the architecture is based on services, it is the application layer running on his house that takes care of the information displayed in the presentation layer. With this the user can have control of two or more different houses without changing any detail on his device application. Moreover, the user can change or check in a glance any status of a component in his house. In the section 6 of this chapter we are going in more detail into the mobile application.

Let us now focus on the presentation layer itself. The structure of the presentation layer is in everything similar to the application layer except for one small detail but that makes all the difference: it has a renderer, meaning that the XML has the screen configuration of the presentation layer. Thus, the presentation layer does not incorporate any screen structure as a normal application does. It simply has a renderer that converts a node of a XML into a known structure. In the case that we want to update the presentation layer with a graphical element not implemented yet, it is simple: we add a method to interpret that new node in the XML. Moreover, if some elements are not being used anymore, the question is even simpler: no change is required to the presentation layer.

In this way, the presentation layer works as a browser. The similarity is obvious: both receive a type of format that they have to render. In the case of a web browser, the most typical is the HTML format. In this case, it is the XML format. And indeed, the web browser does not need to be configured again and again for each website. It is uniform and works with everything. Obviously, new elements are always coming and that is why updates exist. The presentation layer is in everything similar. If there is a standard of elements to use, it works with any kind of application. If some new element is needed to add, it has to be updated to accept that new element. This is elegant and efficient in applications development.

As a remark, this structure of the presentation layer was initially developed in PDM Technology during my internship in the fall semester of 2009/2010. Moreover, it has been proved during this master thesis that this platform is efficient and useful. The scheme of the XML and which elements are being accepted will be topic of discussion in the sections 2.3 and 6 of this chapter.

Hereafter, in the previous chapters we went through the different proposed architectures. We have seen that the Remote Server does not cope with some of the requirements established and then, the Home Server architecture was introduced. This works with SOA as strong principle and based on a three layers scheme. I went through each of the layers in general, but the concept of layer and the communication between them will make more sense to the reader in the next section: Communication Protocol.

2. Communication Protocol

In the previous chapter it was introduced the chosen architecture. It was explained the basic structure of each layer and what is the main goal of each one of the layers. In this chapter we are going through how the communication is made between each layer. It is true that was already referred that XML is used as format of inter connection. However, it was not explained yet how the XML is structured and how each layer works with that format.

In this chapter I am going through XML as the base of communication, i.e. the XML structure in general and the layer structure to deal with the XML; and also the different interactions: from one side between the application layer and embedded layers; and from other side, between the application layer and presentation layers.

2.1. XML as the base of communication

The extensible (X) markup (M) language (L) was created for a main purpose: to uniform the definition of a web application. It is the base language for web services and it defines the data structures of web services, as well as they functionalities. It is also a file format, meaning that any program can store the information in an xml file. The *SafeHouse* system is not an exception; the XML used defines how the application works and how the different layers communicate with each other.

2.1.1. XML Structure

An important concept is defined in this communication protocol: for each request, an acknowledgment or non-acknowledgment is send back to the other layer. This means that the information is only updated internally if an ACK is received from one of the layers. Let us consider, for instance, the case that the XML is sent to one of the layers and that layer is not available anymore. In that case, nothing is received by the sender layer and the update is canceled. Therefore, that layer can chose between sending again or give up. This concept is not new and it is based on the reliable communication in TCP protocol. What was created here was simply an upper layer to treat when a wrong answer is received, as HTTP is an upper layer working over the TCP protocol. Thus, let us have a look into the Figure 27 below. In that following example, a XML was sent by the Sender Layer (SL) to the receiver layer (RL). The XML was received by the RL but the information contained on it was invalid. The RL refuses that XML and sends back a NAK (in XML format as well). The SL receives the NAK and cancels the updating of the information. This is also important in terms of security. If a hacker interferes and scrambles all the XML in the communication channel, the action simply does not happen and it does not crash the application.

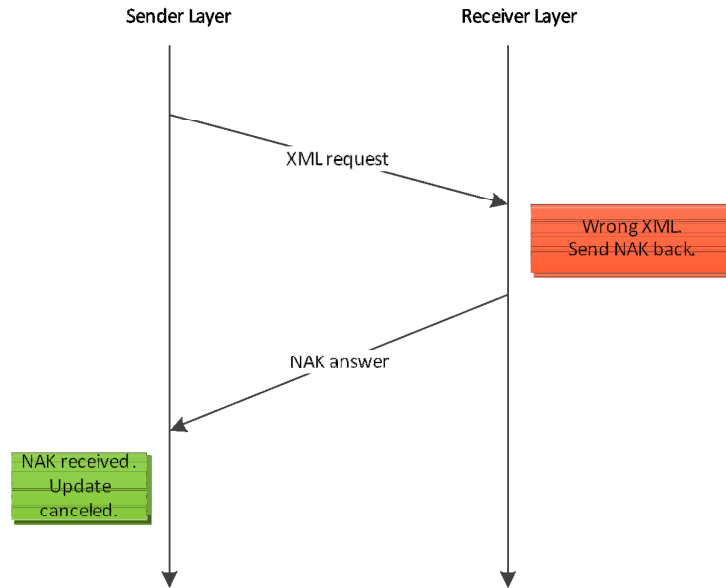


Figure 27 - XML communication between layers. If the XML is not correctly interpreted by the Receiver Layer, a NAK is send back and the Sender Layer cancels the update of information.

Another important topic in this XML structure is that allows simple and composite requests. Let us remember the section 2 of this chapter (page 51) where it was discussed simple and composite services. This XML follows exactly the same idea. If the Sender Layer wants to execute more than one action in the other side, it simply incorporates the different requests inside a main node. And this can be repeated as many times as we want inside the XML, i.e. we can have composite requests inside composite requests. The same works for the answers. In the next two figures, it is shown simple and composite requests/answers. For the case, an interaction between application layer and embedded layer was used to exemplify. But it is important to keep in mind that this works for any interaction in the system between layers.

```

Simple Requests
-----
<SET id="component_id">value</SET>

<GET id="component_id"/>

Simple Answers
-----
<SET id="component_id">value</SET>

<GET id="component_id">value</GET>
    
```

Figure 28- Simple Requests and Answers.

```

Composite Requests
-----
<REQUEST>
  <SET id="id_1">value</SET>
  ...
  <GET id="id_N"/>
</REQUEST>

Composite Answers
-----
<ANSWER>
  <SET id="id_1">value</SET>
  ...
  <GET id="id_N">valueN</GET>
</ANSWER>
    
```

Figure 29 – Composite requests and answers.

Hence, the XML can be composed in simple or composite requests/answers. Either of way, the XML is not more than bytes in the communication channel. An important thing to consider is how each layer should recognize the end of the XML, the so called EOF (End Of File) in Software Programming. There are two solutions for this issue: or by a termination character or by attaching the number of bytes in the beginning of the data stream. Both solutions were designed: in one side and since the embedded layer has limited resources to operate, i.e. limited memory, a header containing the number of bytes to be read would be useless, since it could only receive a limited amount of data (around 512bytes); in other side, the presentation layer has good resources and in this case a number of bytes as header would make all the sense. Thus, to/from the embedded layer a new line character has to be added to the stream of data and to/from the presentation layer 4 bytes in the header indicate the number of bytes of the XML to be read. As a remark, a character represents one byte in UTF-8.

Last but not least, it was consider the encryption of the XML, in order to protect the data. An optimal system would use a public/private key system, such as RSA. By consideration of development time, a simple Vigenère algorithm was implemented (Stinson, 2006). This is a symmetric encryption algorithm, meaning that encryption and decryption are completely symmetric to each other. It is based on a key selection from both sides and switches the data characters using that key and the Table of Vigenère (Rijmenants, 2010). This algorithm is prone to hacking and an application checking the frequency of some characters can decipher the key and then, the message. Even though it does not solve entirely the problem, but instead of Table of Vigenère using the 26 characters of the alphabet, it was used an expanded table of that one using 96 ASCII characters (the first 32 characters of the ASCII table are special and not included in the table). In Appendix C it is explained in more detail the encryption.

After this explanation of the XML structure, it is time to explain how each layer is structured to deal with the XML.

2.1.2. Layer Structure

In the previous sub section was explained how the XML is structured for the communication between layers. It is now the moment to explain how each layer is structured to deal with the XML and which basic functional blocks compose a layer in the system. Let us have a look at the Figure 30 below. The cycle of operation starts when the layer receives a new data stream. This data stream is deciphered and is thrown out any useless bytes (bytes indicating the number of characters in the XML or any termination character). A string is constructed and passed to the next block. As a remark, the embedded device works with pointers to an array of characters. However, the concept of string is going to be used in any layer. Hence, the XML parser receives this string and in order to facilitate and structure the data used by the layer, it is converted into an XML object (in the embedded device, into an XML structure). For a better understanding of this XML object, it is recommended to view again the Appendix A where is

described in detail the XML library. It is with this XML object that the next block is going to work with: the Dispatcher block. This dispatcher is the block that differs from layer to layer. In the embedded layer, it is responsible to execute actions concerning the sensors and actuators. In the application layer, it is responsible to treat a request coming from the presentation layer or to treat an answer from both layers. In the presentation layer, the main function is to render the screen. After the execution of the proper action, the Dispatcher sends back an answer to the sender layer. The process to construct this answer is the symmetric of when the data was received. The xml object is transformed into a string, the string is encrypted and finally a stream of characters is send back.

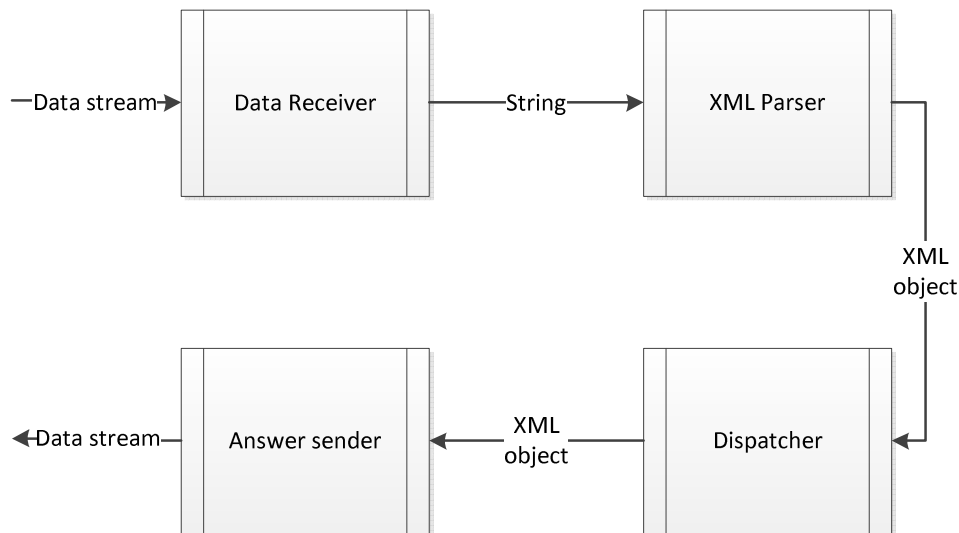


Figure 30 - The general structure of each layer.

About the data receiver and answer sender there is not too much to say. They are responsible to pass from one type of data to another and to cypher/decipher data. The XML parser accepts any kind of XML: the only requirement is that the XML has to be well-formed; otherwise sends an invalid format answer back. The validity of the XML is going to be treated afterwards by the dispatcher. This block performs different actions according to the information tied to the XML. Since we have three different layers, it is preferable to go through each of interactions in the next sub chapters and explain in more detail each dispatcher.

2.2. Application and Embedded Layers Communication

The embedded layer possesses different components integrated to the micro controller. Each of those components has specific properties: for instance, a representation for a led should be a boolean (on/off) but a representation for the temperature sensor should already be an float (or integer). Both types are primitive data types. Other thing to take in consideration: there are components which the user cannot change the status, he can only retrieve; for example, it is possible to control the status of a led but not the temperature sensor (the user can change the temperature only by turning on or off the heating or opening or closing the window). Following

this logic, the application layer has to send in the beginning a query to retrieve which components the embedded layer possesses and which methods can be used in each of those components. Also, it retrieves the type of data: boolean or float. The Figure 31 below shows the first information exchanged between the layers.

The reader can see some properties of each component:

- The attribute **id** of the component: an identification tag for that component in that specific embedded layer. This can be used to identify the component in a more conventional name, for example, Light or Heating.
- The attribute **id** of the type: this goes to identify which kind of data can be retrieved from this component, either a Boolean ("bool") or a Float ("float").
- Inside the node **METHODS** there are two possible child-nodes: **GET** and/or **SET**. If GET appears in that node means that the application layer can retrieve the status of that component. If SET appears in the node means that the application layer can change the status of that node according to the type retrieved previously.

```

Application Layer -> Embedded Layer
-----
<QUERY/>

Embedded Layer -> Application Layer
-----
<COMPONENTS>
  <COMPONENT id="LRDN">
    <TYPE id="bool"/>
    <METHODS>
      <GET/>
      <SET/>
    </METHODS>
  </COMPONENT>
  ...
  <COMPONENT id="TEMPO">
    <TYPE id="float"/>
    <METHODS>
      <GET/>
    </METHODS>
  </COMPONENT>
</COMPONENTS>

```

Figure 31 - First XML exchanged between embedded and application layers.

The application layer, after storing this information, can send commands to the embedded layer. These commands can be either GET or SET. Moreover, the commands can be sent separately or together, meaning that the application can send a simple request (one command) or a composite request (more than one command). Summing the simple requests, they can be three different ones:

- **Query:** as explained previously, this is used each time the application layer needs to know which components are integrated in the embedded device.

- **Get:** a get is used each time the application layer wants to get the current status of a particular sensor or actuator.
- **Set:** a set is used each time the application layer wants to set a new status to a particular actuator.

Each time that the application layer receives a positive answer, it updates the internal status. Otherwise, it throws out the updates. In the next sub section we are going through the other side of this system: the communication between the application layer and the presentation layer.

2.3. Application and Presentation Layers Communication

The presentation layer allows the user to interact remotely with his house. This is made communicating with the application layer at home. Since the presentation layer is integrated in a faster processing device (cell phone) than the embedded device, it allows the exchanging and storing of more information. As explained before, the application layer sends to the presentation layer not only the status of the components at home, but also sends the full configuration of the screen in the presentation layer.

Let us consider the initial situation that the user starts the mobile application. When this happens, a connection is opened with the server that immediately provides a dedicated server to interact with the presentation layer. From this moment all the screens appearing in the mobile application are structured in the XML received by this one. In this way, it can be said that the dispatcher in the presentation layer is responsible to render the information on the screen. For this interaction to happen both layers have to be synchronized in the internal status. Thus, the XML received is stored internally and it is only updated when an acknowledgment is received.

The XML between these two layers is then, divided into two parts - one describing the internal object to update and other describing each method should be performed: rendering the form onto the screen, rendering an alert onto the screen or exiting the application gracefully. Both of them are simple requests that can be composed in the same XML. In the Figure 32 below the example of the login screen is shown. This XML is sent by the application layer after the connection was established. As the reader can see, the first thing to be done is to create a new object in the presentation layer. This object contains not only the information to be rendered on the screen, but also the **handlers** within each button. These handlers are methods in the application layer ready to take care of a user interaction in the presentation layer. For instance, if the user fills in the fields username and password and presses **Login**, this will cause the packing of the current screen information into an XML and send back to the application layer. Also it is contained the handler to be called on the application layer. The application layer then, treats that XML and reacts according to that: if the username and password are correct sends a new screen definition containing the embedded devices at home. Otherwise, sends a render alert method saying that the username or/and password inserted are incorrect.

Regarding the presentation layer there are different methods that can be executed in there:

- **RENDER_FORM**: takes the current status of the internal XML object and renders it on the screen.
- **RENDER_ALERT**: takes the value contained inside this node and renders a custom alert on the screen.
- **EXIT_APP**: closes the connection and exit the application gracefully.

```
<COMMANDS>
  <CREATE_OBJECT>
    <FORM id="1">
      <TITLE>SafeHouse(TM)</TITLE>
      <TEXTFIELD>
        <LABEL>Insert your username:</LABEL>
        <TEXT>username</TEXT>
        <SIZE>20</SIZE>
        <TYPE>any</TYPE>
      </TEXTFIELD>
      <TEXTFIELD>
        <LABEL>Insert your password:</LABEL>
        <TEXT>password</TEXT>
        <SIZE>20</SIZE>
        <TYPE>pass</TYPE>
      </TEXTFIELD>
      <BUTTON>
        <LABEL>Login</LABEL>
        <TYPE>OK</TYPE>
        <HANDLER>onLoginBtn</HANDLER>
      </BUTTON>
    </FORM>
  </CREATE_OBJECT>
  <EXECUTE_METHOD>
    <RENDER_FORM/>
  </EXECUTE_METHOD>
</COMMANDS>
```

Figure 32 - The first XML sent by the Application Layer to the Presentation Layer. All the definition of the screen is described there and also the method render form is performed in the end.

In the side of the application layer there are different handlers to treat the information and to change the screen while the user is interacting with the screen. It could be explained here those handlers, but for a better understanding they are going to be explained in the section 4.2.

Hereafter, I have been referring the communication protocol and how the XML is structured between each layer. At this point, the reader has an understanding of the architecture in general and the communication protocol particularly. In the next section, we are going further and go into the structures of the system, i.e. what is a component or a user for the system and how they can be manipulated.

3. Structures of the system

It has been shown that the *SafeHouse* system is divided into three layers: embedded layer, running on the embedded devices (sensor boards), application layer, running on the home server, and presentation layer, running on the mobile application. Afterwards, we went through the communication protocol and how each layer is structured to work with the XML. Finally, it has been referred how is the interaction between each of the layers. At this point, we can go into details of the system itself.

In the *SafeHouse* system there are four important packages or software structures in order to achieve the established requirements: room/component, responsible to store information about the rooms within the house with the components integrated in the *SafeHouse* system; users, responsible to manage the users in the system and their permissions to access the system; rules, responsible to automate the system itself, i.e. to trigger an action when a certain condition happens; and the notification, responsible to send a sms or email to the user in case of happening a certain event.

Besides these four structures, there is an extra one responsible to generate statistics about the values of the different components. This structure is composed exactly in the same way as the others, but for the purpose of this master thesis it is not considered as a main structure. However, it is going to be referred in the end of this section.

Moreover, each structure of the system is represented exactly in the same way in terms of accessing the structure within the system. Thus, each structure is composed by:

- **Representative objects:** these objects are the ones that define the structure itself, i.e. it is the object-oriented way of representing known structures. For example, a user in the system is represented by an object User. The representative objects will become clear in the next sub-chapters.
- **Object Manager:** the manager is a composition of the representative objects belonging to the same structure. Any interaction within the system has to be made through the manager: adding a new object; editing an existing object; or deleting an existing object.
- **Object I/O:** each structure contains a class with methods to write and read the object manager to/from a file. The project was developed passing each object manager to a binary file, but if another type of file is chosen or if a database is implemented, only this class changes in the system.

In the next sub-chapter we are going through each of previous referred structures: room/component, users, rules and notifications.

3.1. Room/Component

In the *SafeHouse* system was considered that each embedded device represents a room and all the components of that room are connected to the same embedded device. Thus, a room for the user means an embedded layer for the *SafeHouse* system. Moreover, each room is an aggregation of components in the object definition.

3.1.1. Room

In the *SafeHouse*, a room is defined as **an embedded device located in a part of the house with components integrated into it**. Some considerations had to be made in order to transpose a room to an object definition. Thus, a room has the following attributes:

- **macAddress:** this attribute identifies the mac address of the Bluetooth UART integrated in the sensor board. It is used to identify the room internally and externally. Since the mac address is unique for each Bluetooth dispositive, it is assured that each room has a unique ID, the mac address.
- **name:** this attribute identifies the room from an user point of view. Typically, this attribute will have the name of which room where the embedded device is: living room, bedroom, attic, outdoors, etc. This attribute is set initially to the mac address but it can be edited through the Desktop Application.
- **components:** this attribute is actually an array list, where in each position of the list is a component belonging to this room. The definition of component will be explained in the next sub chapter.

The room can also be updated in the case that one new component is added or removed to/from that room. With this, it is kept the plug-n-play characteristic of the *SafeHouse*: no steps are required from the user side to modify the components within it. Thus, let us now define what a component for the *SafeHouse* system is.

3.1.2. Component

In the section 2.2 of this chapter (page 54) it was defined what represents a component between the application and embedded layers. This is not much different of what represents a component inside the system, except for some nuances. Thus, a component for the *SafeHouse* system is defined by **a sensor or actuator with particular characteristics and it is integrated in an embedded device**. Thus, the component has the following attributes:

- **id:** the unique identifier of the component within the embedded device. It was already explained in section 2.2.
- **type:** this represents the type of the value that this component can have. It is a representation of the primitive data type in an object-oriented way. This type can

be one of two: `TypeBoolean` (representing a Boolean) and `TypeInteger` (representing a numeric data type). We are going through the type in the next sub chapter.

- **hasGet:** this attribute is a Boolean primitive type telling the application if this component has a GET method or not.
- **hasSet:** similar to the previous one, except that indicates if the component has a SET method or not.
- **value:** the current value of the component. This is a string representation of the value of the component and needs to be interpreted together with the attribute type of the component.
- **location:** this attribute is a reference to the room object which this component belongs to.

When the room is updated, all the components belonging to it are updated as well. Once more the characteristic plug-n-play is expanded to the components as well. If, for instance, instead of a light that we can turn on and off we implement a new light that has 3 different levels of illumination, the user simply restarts the system and the component is going to be updated with those characteristics. Obviously, this requires a small change on the sensor board programming, but that is all.

3.1.3. Types of Data

The type of data is the attribute type of the Component. There can be two possible types of data: `TypeBoolean` and `TypeInteger`. A type is mainly an object to know which possible values are on a certain component. For instance, if a component is a light probably would only have "on" and "off" but if it is a temperature sensor would have values between a minimum temperature and a maximum temperature. This will become clear in the creation of rules and notifications in the chapter 3.3 and 3.4. Thus, both classes have two common methods:

- **getPossibleValues:** this method returns a list of strings representing the possible values for this type. If the type is a `TypeBoolean` a list with [on, off] is returned. If the type is a `TypeInteger` a list with [min, ..., ..., max] is returned.
- **getPossibleRulers:** this method returns a list of strings where in each position there is a possible ruler for this type. By ruler I mean "is" or "is more than" or "is less than". For example, for Booleans there is only the ruler "is", but for integers there are others.

Thus, we have introduced the room/component structure and at this point we are in conditions to go through the other structures. We are going to continue with the Users structure.

3.2. Users

The structure that defines the users in the system is also important for the system. At a first glance the reader could think for which reason there are users in the system if the system is targeted to domestic use. Well, there are different explanations for that. First of all, with an implementation of user structure, the system can be targeted to different purposes. For instance, if a hotel company wants to implement the *SafeHouse* system in its hotel, perhaps it would be interesting to restrict the use of the system to customers of the hotel. Even though the manager would have full access to the all the rooms but each customer would only control its own room. The same goes for a company with different offices. Secondly, even in a domestic situation happens that in many cases different tenants are sharing the same house. No user would like to give the control of his bedroom to other user living in the same house.

Thus, a system of user and super-user was implemented in the *SafeHouse* system with permissions for different users. We are going through this in the next sub sections.

3.2.1. Definition of user and super-user

For the *SafeHouse* system, the user is **the individual that is an inhabitant of a house that has identification to access the system and permissions to access certain rooms**. With base on the permissions and similar to an operating system, the user structure is divided into two: the super user with full access either in the desktop application or in the mobile application and the other users with permissions given by the super user. Thus, it is only allowed for the system to have one single super-user, but it is allowed to create as many users as the super user wants. Obviously, in a domestic situation where lives a family the super user can decide just to share his account between all the family members. As it was explained before, this structure was only defined to expand the system to different situations in home automation.

Even with this separation of super user and user, the object representing both of them is the same and it is called User. This object is characterized by the following attributes:

- **username:** this attribute represents the unique identifier of the user in the system and it cannot happen that there are two different users with the same username, as it happens in any kind of application.
- **password:** for security matter, each user has a password. This password is used to access the system, either through the desktop application or through the mobile application.
- **superUser:** this attribute is a Boolean indicating if this user is the super user or not. This attribute could be omitted from the object since there is a manager object dealing with the users, but it gives extra information about the user object.
- **permissionAccess:** this attribute is an array list containing in each position a reference to a room which the user can access to.

There is indeed an issue with this permission access system. If it happens a changing in the number of the rooms in the system, the user could have an access that already does not exist or the super user to not have the full access to the new rooms. For that, the user manager object is equipped with an update method: this method deletes references to rooms that already do not exist and add references to the super user of new rooms added to the system. This concept will become clear when we will go through the home server in section 4.

3.2.2. Creation and Deletion of an user

As it has been already talked about, there are some constraints to create an user in the *SafeHouse* system. First of all, either the username or the password has to contain at least 4 characters. Any number of characters could be used here, but 4 seemed a reasonable number of characters. Also, the user introduced cannot be the super user, i.e. it can only happen one super user in the system and that user is created in the first system start up or every time the system is reinstalled. The last constraint goes for the fact that cannot be introduced an user with the same username as an existing user in the system.

About the deletion of a user, there are only two constraints. Trivially, it can only be deleted a user that exists in the system. The second constraint goes for the fact that the super user cannot be deleted from the system. The reason for this is that only the super user has full access to both applications in the *SafeHouse* system and after losing this user, it would not be possible to control the system à posteriori.

As a remark, only the super user can create or delete a user in the system, meaning that both of the operations referred previously are only targeted to the super user. In the next sub section we are going through one of the most important structures of the system: the rules as automation of the house.

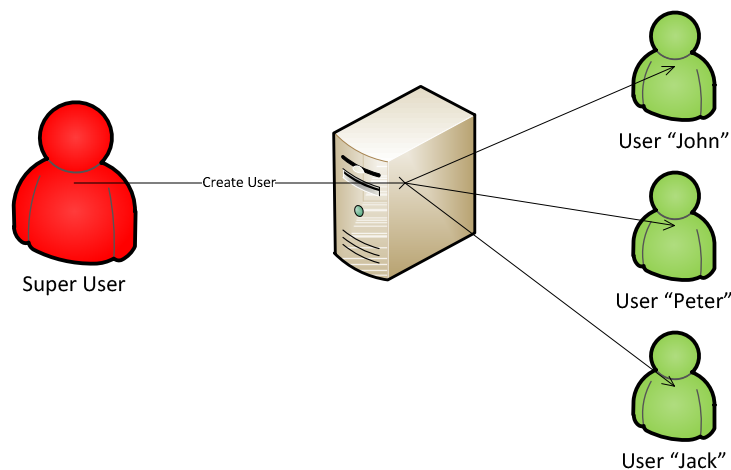


Figure 33 - The Super User is the only one that can create or delete users from the system. This gives an extra security to the system.

3.3. Rules

The rules are the structure that allows the system to be automated. It answers to the need of any user to automate tasks at home. The simpler example happens with the temperature at home. For instance, if the temperature in a certain room goes too high the heating should be turned off at that moment. Or even considering energy saving, if a window is open the heating should be turned off, because it is consuming energy uselessly. More situations can be the topic of the explanation to exist rules and each user would have his/her idea of which actions to perform. Thus, it is allowed to the user to create the rules or to delete them. There are no rules defined in the first system start up and then, it is the user who should define them afterwards.

3.3.1. Definition of Rule

In the *SafeHouse* system, a rule is defined as **an automated action defined by the user composed by a source component and a triggered component – a certain value reached in the source component will cause an action in the triggered component to change its status to a certain value.** In a simple logic point of view, a rule can be defined by the following formula:

$$\text{if } (x \text{ is } [\text{more than, less than}] \alpha) \text{ then } y \text{ is } \beta$$

Where x is the source component that triggers the action, α is the value in the source component that triggers the action, y is the triggered component where the action is going to be triggered and β is the value of the target component after the action is triggered. Obviously, in the case that the source component is from the type Integer, the ruler in the condition can be **is more than** or **is less than**. One important aspect of the rules is the fact that they can be scheduled, meaning that will be triggered an event only if the current time is in an interval comprehended between an initial time and a final time. Thus, the rule is defined by the following attributes:

- **component:** this is the source component of the condition and it is a reference to a component in the structure of rooms/components.
- **triggerComponent:** this is the triggered component of the condition and it is a reference to a component in the structure of rooms/components.
- **ruler:** this can be one of three: "is", "is more than" or "is less than" and it depends on the type of the source component.
- **value:** this is the selected value for the source component in the rule and it will trigger an action when reached that value.
- **triggerValue:** this is the value that the triggered component will have if the condition is fulfilled.
- **InitialTime:** this indicates the time from when the rule is active.
- **FinalTime:** this indicates the time from when the rule becomes inactive.

Once again and as it happens with the users, when there are updates of rooms and/or components, the rules are updated as well. For instance, if a component is not in the system anymore, the rule is deleted from the system if it contains as source or target a reference to that component.

Nevertheless, the rules are topic of more study in the *SafeHouse* and constraints for the creation of rules had to be created. For instance, let us say, that the user creates two different rules: in one the fact that the temperature goes high triggers that the heating turns off; in other, the fact that the temperature goes low triggers the same event on the same component. This would be a contradiction and it would cause unexpected events to the user.

3.3.2. Creation of a rule

In first place, the creation of a rule is only allowed to the super user for the same reasons already presented before. Thus, for the super user to create a rule he has to know the concept of a conflict with other rule. There are different situations when a conflict can happen and we are going through each of them:

- **Same component in both sides of the condition:** the source component and the triggered component cannot be the same. This could cause an unnecessary or unexpected action in the system. For instance, if the rule would have as source and triggered component the light of the living room, and the condition would be that when the light is turn on then turn on, this would be an unnecessary processing in the system or in the other hand, if the condition would be that when the light is on then turn off, this would cause that the light will be always off even when the user would turn on.
- **Same triggered value conflict:** this conflict happens only when there is already an existing rule with the same source component, triggered component and ruler. Let us consider the Figure 34 below. As the reader can see, different actions in the same source component would cause the same exact action in the triggered component. This would mean that for every state on the source component, the triggered component would have always the same value and the control of this component would become useless, since it will be 100% affected by that source component.
- **Interval interception conflict:** this conflict is a bit more complex and can only happen when the selected source rulers are “is more than” or “is less than”, i.e. when the type of the source component is an integer. Once more, for a better explaining is recommended to see the Figure 35 below. As we can see, for the values between 20 and 25 of the temperature, the heating will be turn on and after off. Even though the source values are not the same value, but the condition can intercept other existing condition. The same goes for components with Boolean values, but in this case is simpler to explain: the source value has to be exactly the same. Only for integers special attention had to be taken.

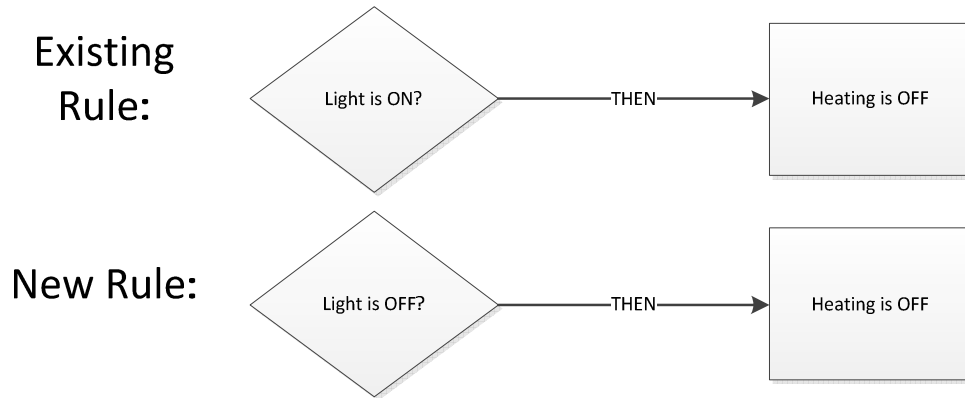


Figure 34 - Same target value conflict: different values on the source component trigger the same even on the triggered component.

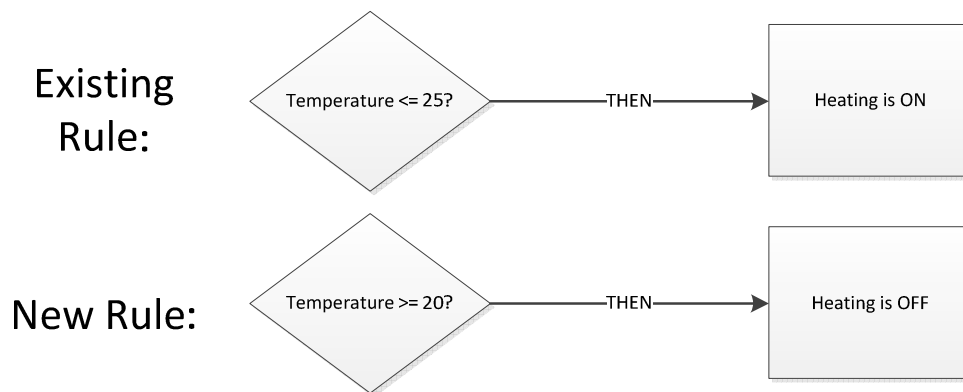


Figure 35 - Interval interception conflict: when for some values of the source component two actions would be trigger in the triggered component.

Obviously, more constraints can be considered for this case of study and the rules as automation would be an extensive topic. Let us notice that no multiple conditions for the same action were considered. This could be an extension for this system and such extension would make the user benefit increasingly in terms of interaction and control. Also, no looping conflicts were considered, i.e. an event triggered in one of the rules can cause a chain of events in the subsequent rules. This can be seen as good, but if no care is taken, it can happen that an event that the user is expecting to happen in the end is not happening because of the referred chain of events.

Finally, and about structures of the system we are going through the structure containing the notifications. This structure is responsible to organize a notification and what should be sent to the user when a certain event happens.

3.4. Notifications

The *SafeHouse* system would not be complete without a system to notify automatically the user whenever he is not checking any application. Thus, the notifications have been designed to ensure that the user has the freedom to go wherever he wants without worrying if some event is happening without being notified. Moreover, the user can receive notifications in two different formats: e-mail and SMS. How the notifications are sent will be topic of discussion in the section – Home Server. In this section will be explained how they are structured.

3.4.1. Definition of Notification

For the *SafeHouse* system a notification is **a safe way of notifying the user if some event (configured by the user) happens at home**. As explained before the notifications can be in two forms, e-mail and SMS. Thus, it was designed a division of the notifications into each type of notification: *EmailNotification* and *SMSNotification*. Both classes descend from a super class *Notification* and they share the same properties, except for the method *send*. Moreover, the notification can be formulated in the following way:

if (x is [more than, less than] ∞) then sendNotification

As we can see, the notification is really similar to a rule in the left side. The most significant difference is that instead of triggering an action in other component, it simply sends a notification to some recipient. It is exactly this recipient that makes the notification different from a rule in terms of attributes. Moreover, a notification has another important aspect to be considered: the frequency of the notification. For instance, there is no use for the notification to be sent every time the system checks the condition. Finally, a notification, as well as it happens with the rules, it can be scheduled. Therefore, an attribute frequency was created.

Thus, a notification has the following attributes characterizing it:

- **component**: this is the component that will trigger the notification.
- **ruler**: it has the same meaning as in the rules.
- **value**: the same as in the rules.
- **initialTime**: this indicates the time from when the notification can start to be sent.
- **finalTime**: this indicates the time from when the notification has to stop to be sent.
- **recipient**: the recipient of the e-mail or SMS.
- **frequency**: the frequency at which the notification should be sent. There were considered two options: Hourly and Daily, but obviously more options can be added.

Also, the notification is updated each time that a room/component in the house is changed. Nothing new about this: the users and rules work exactly in the same way. A drawback of this

structure of notification is exactly to not be aggregated with rules, meaning that the user has to create the rule and if he wants to get notified about that, he needs to create the notification as well. Finally, in order to conclude the Structures of the system we are going through the final one of the *SafeHouse* system: the Statistics structure.

3.5. Statistics

During the development of the *SafeHouse* system was thought to design and implement a statistics structure capable of providing averages of the values for each component. This structure is the responsible to give a concept of awareness to the user, i.e. collecting data over the time it allows the user to see how is the behavior of the inhabitants of his/her house. Due to the complexity of such block of the program, the statistics are absent of correlations or variations between components. It only allows the calculation of the average during a certain period of time.

Therefore, in the core of the statistics structure there is the object **Record** that contains information about each component of the system. Thus, this object contains three attributes:

- **component**: this attribute is a reference to a component belonging to the system. It is set when the object is created or every time the Record is updated.
- **dates**: this attribute is a list of dates represented in Unix time, i.e. the time in milliseconds since 1st January of 1970.
- **values**: this attribute is a list of values belonging to this component. This list of values has the same size as the attribute *dates* in order to relate a value with a date. For components with Boolean data type, this value can be one of two: 0.0 or 1.0 to represent “off” and “on”, respectively.

It was designed two different sets of data: the average in each hour during the last day; and the average in each day during the last month. The way of calculating each of them is really similar. Then, let us see the calculation for the last day.

3.5.1. Average of results in the Last Day

In first place, we have to consider at which frequency the values of each component are checked in the system. Thus, let us say that F represents the frequency in seconds at each component is checked. Moreover, let us say that N represents the number of samples taken per hour at the given frequency. Thus, N is calculated by the following formula:

$$N = m * \left(\frac{s}{F}\right)$$

Where m and s are constants and represent the number of minutes in an hour (60) and the number of seconds in a minute (60), respectively. For instance, for an $F=2$, N would be 1800. After this step, the method finally calculates the average of each hour, checking always if a certain value belongs to the hour being checked at that moment. For the average for each day in the last month, the proceeding is similar as we are going to see.

3.5.2. Average of results in the Last Month

This method is in everything similar to the previous one, so let us see the differences. In first place, the value N is different and this time calculated by the following equation:

$$N = h * m * \left(\frac{S}{F}\right)$$

Where h is a constant and represents the number of hours in one day (24). Afterwards, it is made the average for each day.

As a last remark and as it is happening with the other structures, this structure is updated each time an update happens in some room or component.

3.6. Conclusion

We have seen different structures of this system. We went through in first place to the rooms and components as main component of this *SafeHouse* system. Hence, it was introduced users, rules and notifications. The most interesting characteristic is the plug-n-play of each of these structures: if one change occurs in the central structure of rooms and components, they have option to be updated concerning those changes. The different structures are shown in more detail in the Appendix D, where a class diagram is shown for each structure.

In the next chapter we are going through the central and vital part of this system: the Home Server. As we are going to see, this component is responsible to launch the different processes, to launch the desktop application and to send notifications when an event happens.

4. Home Server

In the previous section we went through the basis of the *SafeHouse* system. We have seen which architecture was chosen in first place: the Home Server architecture structured in three different layers. Hence, we have been discussing the communication protocol and how the different layers communicate with each other. And finally, we have seen which structures exist within the *SafeHouse* system. Thus, we are already in conditions to go further and present the complete structure and work flow of the home server, i.e. the component that deals with all the information coming from external devices and that triggers rules and notifications.

The Home Server can be divided into different components: discovery of embedded devices using the Bluetooth discovery protocol; looping embedded devices to set and retrieve status; trigger of rules to be followed; trigger of notifications to be sent; server for external applications (mobile); and check periodically the status of each component for the statistics structure. To understand better this process let us have a look at the following workflow in the Figure 36 below.

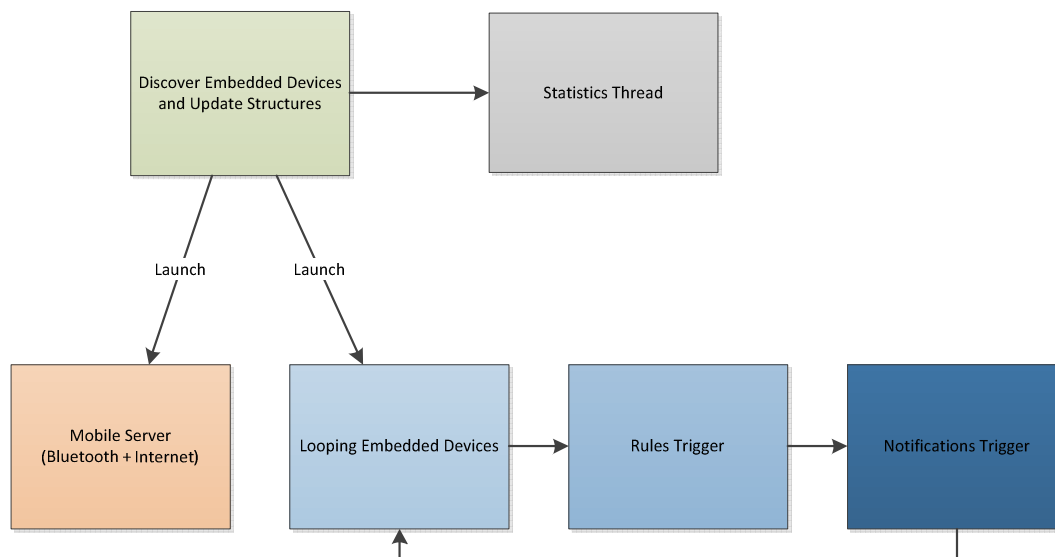


Figure 36 - Basic Workflow of the Home Server: After the embedded devices being discovered, it launches the mobile server and the embedded process to update status and trigger rules and notifications.

Thus, when the home server is launched the first thing to be done is to discover the embedded devices at home. This is made using Bluetooth discovery protocol. After the devices are discovered, it is launched two different processes: one to handle mobile connections – the mobile server – and one to loop the embedded devices and trigger rules and notifications. Moreover, there is the statistics thread running and collecting data over time. We are going through each of these components in the following sections.

4.1. Discovery of Embedded Devices

The discovery of embedded devices is, by all the means, one of the most important parts of this architecture and it is the one that allows SOA for the embedded devices. With this process, the system gets to know which devices and/or components are in the house and is able to update all the existing structures. The process of discovery runs in the beginning of each session launched by the user. During this process, there is a sequence of steps to be followed:

1. The Bluetooth discover service is launched. This service searches first all the devices in the range and after collecting all the devices, it searches for the Bluetooth services running within each device. A Bluetooth service is identified by the port and service name where it is running. Therefore, knowing which service name each of the embedded devices possesses, it is straight forward to get the information we need: the mac address, the port and the service name. These three identifiers are essential to connect to the embedded device and retrieve information from it at any time.
2. At this point, we already know which embedded devices are present in our house and we can query each of them. For that a XML with the query tag in the root node is sent to each of them. The answer from this query is a XML with the components that are present in each of them and the methods to access them. All of this information is transposed to the structure of rooms and components.
3. Since this process of discovery is repeated in the beginning of each session, it is necessary to load the existing devices from the previous session and update them with the new devices. This is important in case that a new component or a new device was installed in the house. The same happens if a device disappears from the system. In this way, no setting is necessary from the user side. After updating the rooms and components in the system, it is updated as well the notifications, rules and users. The updating of the structures was already explained in the section 3 of this chapter (page 59). Finally, the system is ready to launch the main processes.

In the Figure 37 below is shown a diagram representing the steps explained above. After the discovery is completed, the mobile server and the looping process will run. We are going through each of them in the following sections.

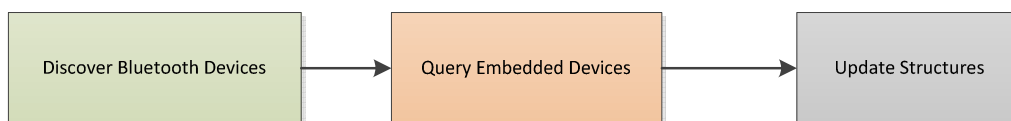


Figure 37 - Workflow of the Discovery process.

4.2. Mobile Server

As it was said before, the mobile server is responsible to handle and treat each connection coming from the mobile application. For each connection received from a mobile application it is launched a dedicated server for that application, independent of the physical medium used for that connection. Nevertheless, the mobile server comes in two flavors: an Internet Server and a Bluetooth Server, both ready to receive an internet or Bluetooth connection, respectively. The reason for existing two different means of connection is simple: it allows the user to have a cost-free connection at home through Bluetooth and a distance-independent connection through Internet. As remark, the Internet server is the same for any kind of connection: Wi-Fi or 3G, meaning that the user can have also a free-cost connection when he/she is not at home.

Therefore, there are some constraints that have to be followed in both connections:

- **Internet Server:** this server has to have a public IP attributed to it and an open TCP port for it as well. The port can be any, except for some special ports reserved to HTTP, SMTP, FTP, etc. Also, the mobile application has to know the IP and port of this server. The URL to connect to such server is as it follows:

```
socket://house_ip:server_port
```

- **Bluetooth Server:** this server has to have a unique identifier (UUID) when it is created and a service name to identify it to other Bluetooth devices. Moreover, this server is a RFCOMM server. The port where the server is running is not chosen by the application, but it is discovered afterwards by the Bluetooth discovery service in the mobile side. The URL to connect to such server is as it follows:

```
btsp://bt_mac_address:server_port;name=server_name
```

Thus, both servers are running simultaneously waiting for client connections (from the mobile application). For each connection received is launched a dedicated server to treat each presentation layer. In the Figure 38 below this scenario is represented. Both servers are running and waiting for incoming connections. When a connection is accepted, it is launched the dedicated server thread to treat the incoming presentation layer and the server goes again for the waiting state. This is the way how any TCP server works, independently of the application protocol running on it.

Thus, the dedicated server is launched, receiving as argument the connection established. Some attributes are initialized in the dedicated server when this one is launched. It is important to refer two important attributes of the dedicated server:

- **Form Object:** this is a replica of the configuration screen in the presentation layer (PL) and it allows the dedicated server (DS) to be synchronized with the PL at any time. Thus, it is this object that describes which screen the PL should render and it is this object that allows a full separation between layers.

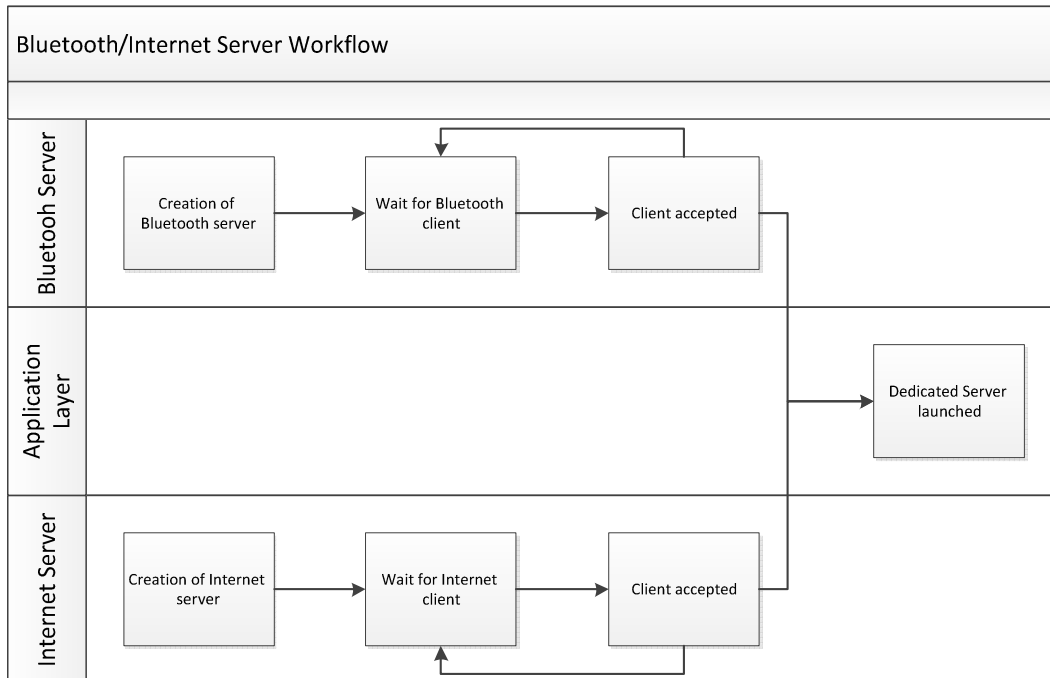


Figure 38 - Bluetooth/Internet server workflow.

- **Command Builder:** when some new information is being processed in the DS, it puts that information into a queue – the command builder. If the DS is ready to send a new XML, it takes all the commands in the queue and sends to the PL. When an acknowledgement is received afterwards, the local object is updated and the command builder cleans the queue.

Therefore, when the DL starts is sent to the PL the first configuration screen. This configuration screen is the login screen and it is going to be rendered in the PL. When the DS receives the acknowledgement, it updates the form object with this information and the command builder becomes empty. From this moment the DS is passive in the sense that only reacts upon a PL request. It is important to say that in each XML sent to the PL, particularly in the buttons to be rendered, a handler name is attached to those buttons, i.e. when the user presses one of these buttons in the screen it is going to be called this handler in the DS side. Then, in the DS a method is going to treat the form object and send an updated form definition to the PL.

To understand this process better, let us see step by step this process:

1. The DS is launched and it is created the first form object. This form object is transformed into XML and sent to the PL. When the PL acknowledges the DS, the DS stores this form object locally and waits for new information.
2. The DS receives new XML. This XML contains the form definition and the handler name. Thus, the DS updates its form object and sends an acknowledgement to the PL. After this step, the DS launches the handler (method) to treat the current form object.

3. After the handler has treated the form object, a new XML is sent and the process repeated.

The same process is repeated in the PL side, we are going through that in section 6 though. For now, it is important to know which handlers are available in the DS:

- **onLoginHandler**: this handler is responsible to treat the form object when the user pressed the button "Login". It is going to check if the user exists in the system and if the password is correct. Moreover, it checks which permissions the user has. If the user exists and the password is correct, sends a new form definition with the rooms that the user can access.
- **onExitHandler**: this handler is responsible to terminate the application gracefully. When this method is called, it sends a XML telling the PL to terminate and it terminates the connection and therefore, the thread.
- **onRoomSelectedHandler**: this handler is launched when the user already selected a room where he wants to see the status of the components. This handler is going to retrieve which components are in that room and send to the user in a new form definition.
- **onComponentChangedHandler**: this handler is going to change the status of a specific component selected by the user and send an updated form definition with the next status.
- **onBackHandler**: this handler is called when the user goes back to the rooms screen. It performs basically the same thing as onLoginHandler except that does not verify the user again.

The most important thing to retain from the dedicated server is the following: if we decide to make a completely new system targeting other market, it is only this dedicated server that changes, since the presentation layer works as a browser. Nevertheless there are limited elements in the presentation layer that can be rendered, but we are going through them in the section 6 about the mobile application.

Thus, in this section we have presented how the server works in general and how the dedicated server works, in particular. In the next sections we are going to present another important responsible to deal with the embedded layer and also trigger events in respect with the existing rules and notifications of the system.

4.3. Embedded Server

In the previous section we went through the mobile server and how it deals with connections from mobile applications, i.e. from presentation layers. In this section we are going through the other side of the home server (application layer), i.e. to retrieve and set up new information from/to embedded layers, as well as trigger rules and notifications. Thus, let us remember the Figure 36 in the section 4 (page 71). We have seen that the embedded server starts with looping all the embedded devices in order to update the status of all the components. Hence, it loops all the rules, and if the condition is happening and if the rule is respecting the established schedule then, it updates the status of the target component internally. Finally, it does the same for the notifications: if the condition is happening and it is respecting the schedule and frequency, it sends a sms or an email to the number/address described in the notification with the event information.

Therefore, we are going through each of these blocks in the following sub sections.

4.3.1. Embedded Looping Block

This block is responsible to connect to each embedded layer and retrieve each component status, at the same that sets the new status if the component has a set method. Thus, the first thing is to load the binary file with rooms and components. Once the file is loaded into an object, it goes for each for room and connects to the device representing that room. In each connection with an embedded layer, it goes through all the components integrated in the embedded device, and:

- If the component has a **set** method, it updates the component in the device with the new status.
- If the component has a **get** method, it simply retrieves the current status of the component and it updates the object internally.

In the Figure 39 below is represented the work flow of this block. After running through all the devices and components, it finally stores the object as a binary file and it continues for the next two blocks: rules block and notifications block.

4.3.2. Rules Trigger Block

After the components are updated internally, it is the moment to go through every rule stored in the system to check if the conditions are true. Thus, the first step of this block is to load the rules and the rooms/components from binary files into objects. In one hand, the rules contain the conditions that will trigger an action in the target component. In other hand, the rooms/components have the current status of every component. Thus, the next thing to do is

to check each rule. In each rule, it gets the current value of the source component and the value described in the rule. Thus, the following algorithm is performed with respect with the selected ruler for the condition:

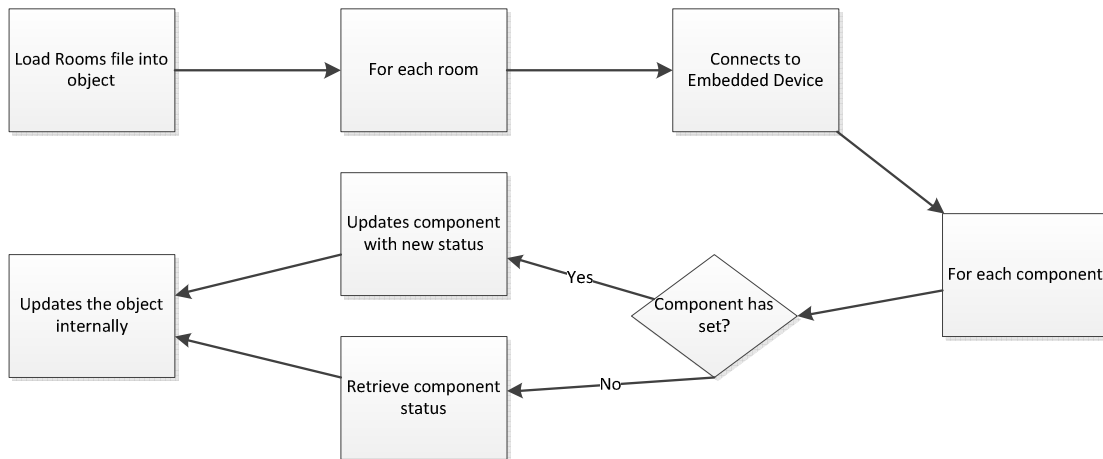


Figure 39 - Workflow of the Embedded Looping block.

Obviously this algorithm only runs in a schedule fixed by the user.

After running all the rules and updating the status when necessary, finally it stores the objects into a binary file. In the Figure 40 below the reader can see the workflow of this block. Therefore, the embedded server continues for the notifications block.

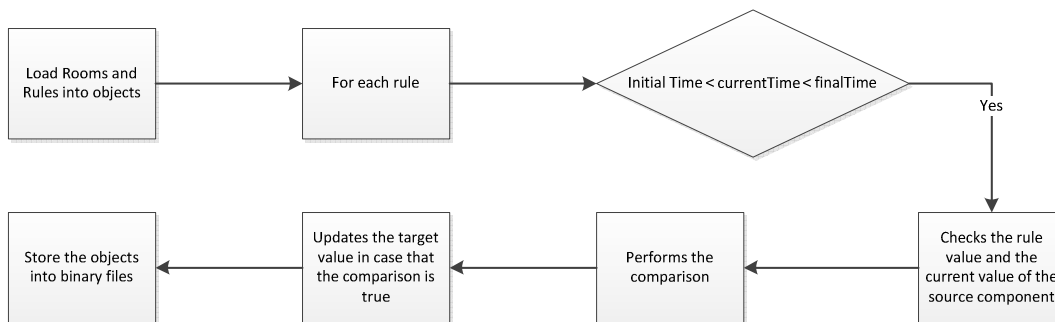


Figure 40 - Workflow of the Rules block.

4.3.3. Notifications Trigger Block

The notifications block work in the same way as the rules block. This means that also the rooms are loaded and the comparison is made in the same way. The only thing changing is the action triggered. In the notifications an email or SMS is sent to an email address or phone number, respectively. These two blocks could have been joined together, since they share similar characteristics. The reason why they were not shared is trivial: in one side we have house automation (the rules) and in the other side we have house safety (the notifications).

Still true that associated to a rule could be a notification (optional), but it was decided to split them into two different structures and follow that design.

Since there is such similarity, we are going through directly to the notification itself. When a condition is true according to the current status, there are two types of notifications that can be sent: SMS or email. For the SMS service it was chosen to concretize it through a HTTP Post to a VoIP web application, for instance, the *VoipCheap*. Thus, in that HTTP Post the username, password, recipient and message have to be inserted. For the email, it was chosen to use a SMTP connection to *Gmail* and send the email through it. For that, the same values have to be send. The message is constructed in the following template:

- “The **component_name** in the **room_name** is [more than/less than] **value**”, where **component_name** is the name of the component where the condition is true, the **room_name** is the location of that component in the house, the [more than/less than] are optional and just when the component is from the type integer, and finally, the **value** is the value in the condition that triggered the notification.

Moreover, the notification is only sent when is within the established schedule and when there was no notification sent in the last T minutes, where T is the frequency set up by the user.

Finally, we are going through to the Statistics thread where the statistics are collected over time.

4.4. Statistics Thread

This service or thread is responsible to check the status of every component at a defined frequency. The notion of frequency was already explained in the section 3.5 of this chapter (page 69). The reason for separating the statistics from the main service (Embedded Server + Rules Trigger + Notification Trigger) is simple: to not overload more the system during that main loop. Moreover, the file containing the statistics can become too big to be opened and saved every time the loop is made. Therefore, this simple thread was created in order to retrieve the status of each component. The steps to follow are simple:

1. Loads the structure of the rooms and statistics.
2. For every component in every room it adds a new register to the statistics structure.
3. Saves the structure of statistics.

With this separated thread, the system does not get compromised and it allows the creation of charts with this data as we are going to see in the next section.

Finally, we have concluded the notifications and therefore, the embedded server and the home server itself. We went through all the processes running in the home server and then, we are in conditions to introduce the desktop application, responsible to create or delete different objects in the system.

5. Desktop Application

The desktop application is launched when the server is launched and it allows the super user of the system to manage all the structures in the system, such as creating a new user or customize the name of the rooms, among other functionalities. Other users can also log in, but they are not allowed to access any functionality of the system. They can only see the logs of the system. This was explained already in the section 3.2 of this chapter (page 62).

In the Figure 41 below is shown the scheme of screens in the desktop application. It starts with a **loading screen** while the home server is discovering the embedded devices and updating the structures of the system. When this step is completed, it is launched a **login screen** where the user can sign in to the system. Finally, it is launched the **main menu** where all the structures are accessible through different buttons.

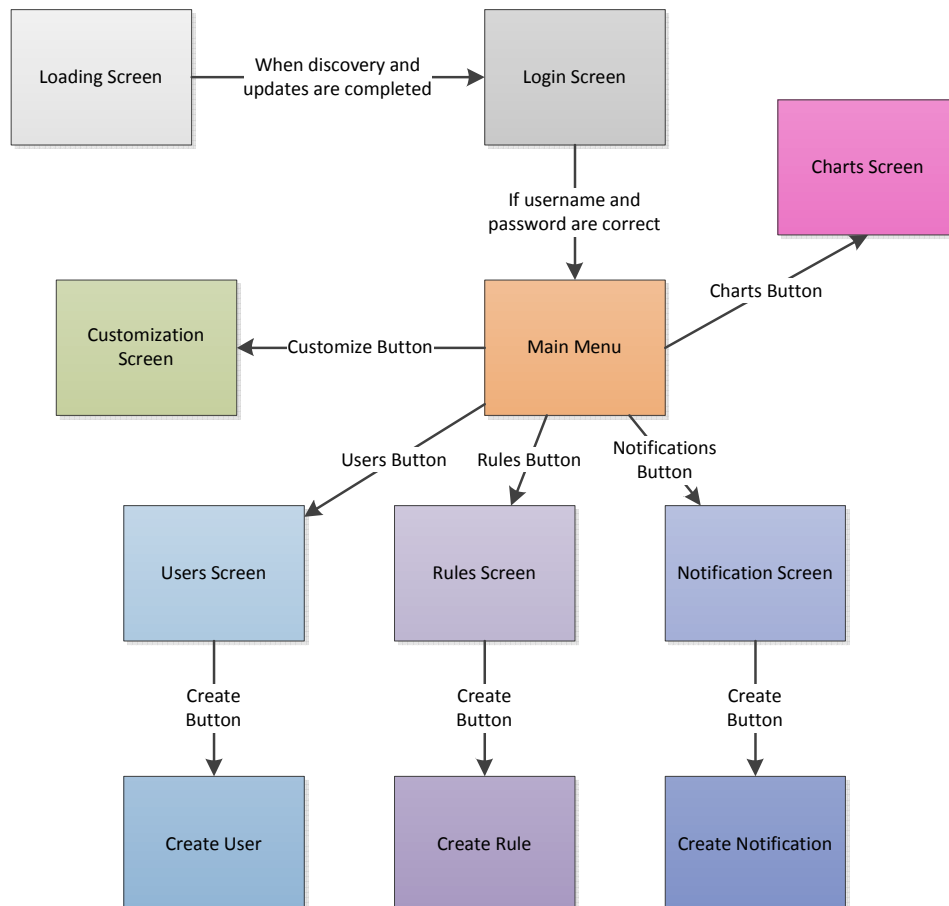


Figure 41 - Screens Scheme of the Desktop Application.

Therefore, most of these screens need to communicate with the structures of the system. Instead of loading the structures directly into the screen classes and handling them there, it is

provided an interface for these operations. This interface is called **GUIInterface** and it provides all the necessary methods to get or modify information from/to the structures. Also, in case of confirmation of some actions, such as inserting a username and password, it is returned an object **Message**. This object contains two attributes: a Boolean indicating if the operation was successful or not and a string representing a message to insert in the screen. For a full overview of this interface, it is recommended to consult the Appendix E where all the methods described in detail, as well as the *Message* class. In the next sub sections, we are going through each of these screens.

5.1. Loading Screen

As it was explained in the Home Server section, the first thing when the server is launched is the discovery of devices and updating every structure of the system. Thus, during this period, it is important to show a temporary screen informing that the system is loading, in order to respect usability principles. This could have been achieved in two ways: through a splash screen with an animated picture, or simply by showing a screen with a loading bar telling that the system is loading.

Therefore, it was chosen the second option: show a screen with loading bar. This screen will appear during the first moments of the start-up and the user cannot see other screen while this one is on the foreground of the desktop. This screen is shown in the Figure 42 below. We can already see some common elements for all the screens. For instance, the title of the screen is always in the following format: SafeHouse™ - *screen_name*, where *screen_name* is the name of the current screen. Also, the scheme of colors chosen was based on blue as background and white as foreground. The reason for this is again usability principles on the color schemes. Moreover, different smart home systems are using this scheme of colors. In other hand, we can see the application icon on the top of the window and a logo inserted in the bottom of the window.

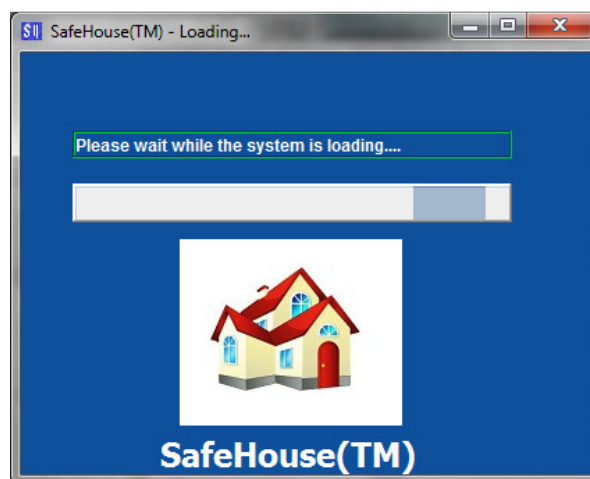


Figure 42 - Loading Screen of the Desktop Application.

When the discovery and updating are completed, this screen disappears and pop-ups a new screen: the **login screen**. At this moment, the server is fully running and the user can manage his/her *SafeHouse* system.

5.2. Login Screen

In most of the applications we always observe a login screen. This screen is a way of authenticating the user into the system. Thus, it is only allowed correct pairs of username/password. The login screen is represented in the Figure 43 below.



Figure 43 - Login Screen of the Desktop Application.

If the user inserts a correct pair of username and password, this screen disappears and it pop-ups the **main menu**. But in case that the user inserts some incorrect information, the screen has to inform the user about that error. There are three kinds of error:

- **Empty fields:** when the user did not fill in all the text fields. Thus, a message with “You must fill all the fields.” is presented in the screen.
- **Not existing user:** when the user inserted a username that does not belong to any user in the system. In this case, a message with “The user doesn’t exist in the system.” is presented in the screen.
- **Incorrect password:** when the username belongs to some user but the password is not correct for that user. In this case, a message with “The password inserted is incorrect.” is shown in the screen.

These three different situations are shown in the figures below. In the next section we are going through the fundamental screen of the system: the **Main Menu**.



Figure 44 - Loading Screen with empty fields generates an error message.



Figure 45 - Loading Screen with a not existing user generates an error message.



Figure 46 - Loading Screen with an incorrect password generates an error message.

5.3. Main Menu

After the system validates the pair username/password, the **login screen** disappears and finally it is pop-up the **main menu**. In the main menu itself it is only possible to see **logs of the system**, on the left side of the screen. These logs are inserted while the server is running and in different circumstances. For instance, if a user has logged in into the system or if a value of a component changes, a log is inserted into the system. Nevertheless, there are **five buttons** in right side of the main menu that allows the user to access the different structures of the system. Each of these buttons represents exactly one different structure and the names of them are obvious if we remember the section 3 of this chapter (page 59) about structures. Thus, each of these buttons opens a new screen. In the Figure 47 below the main menu is presented. We are going through each of these screens in the next sub sections.



Figure 47 - Main menu of the Desktop Application.

5.3.1. Overview Screen

When the user presses the **Overview** button in the main menu, a new screen appears, to allow the user to edit the name of the rooms in the system and also to see which components are in each room. We have seen in the section 3.1.1 of this chapter (page 60) about the Room object that the room name is initialized with the mac address of the embedded device representing that room. Thus, there must be a way for the user to change this name, for usability sake. This screen requires the user to know which mac address belongs to which room, what can be

consulted in the Bluetooth UART of the embedded device. It is true that this situation is not the easiest for the user, but it is a small requirement that has to be fulfilled from the user point of view. Moreover, this can be seen as a positive point for the user: this one is able to change the place of the embedded devices, being only required afterwards to change the name of the rooms in this customization screen.

Therefore, this overview window shows a different screen for each room, where is presented the current name of the room and a text field for the new name of the room. In the Figure 48 below is shown the last screen of the customization sequence. In each screen is shown in the bottom two different buttons. The left button is used to go back to the previous screen or to cancel the customization if it is the first screen. The right button is used to go for the next screen or to finish if it is the last screen. This customization is recommended to be done when the system is launched for the first time. Afterwards, customization only is required if the user changes the embedded devices from place.

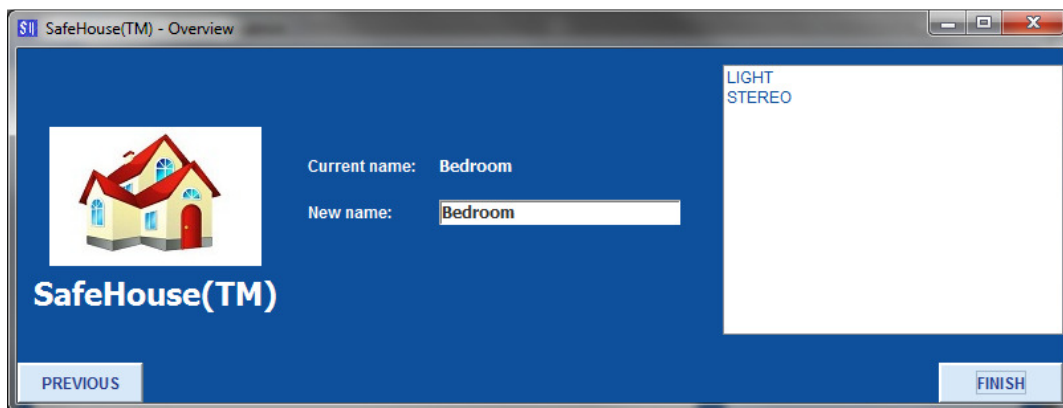


Figure 48 - Customization Screen of the Desktop Application.

5.3.2. Notification Screen

When the user presses the **Notification** button in the main menu it is launched the Notification screen. In this screen the user can see the already existing notifications, create a new notification or delete an existing notification. In order to accomplish this, it is presented the screen represented in the Figure 49 below. We see that the list is initially empty. Obviously, no notifications are set up in the first system start up and then, it is the user who has the control of creating notifications to be sent.

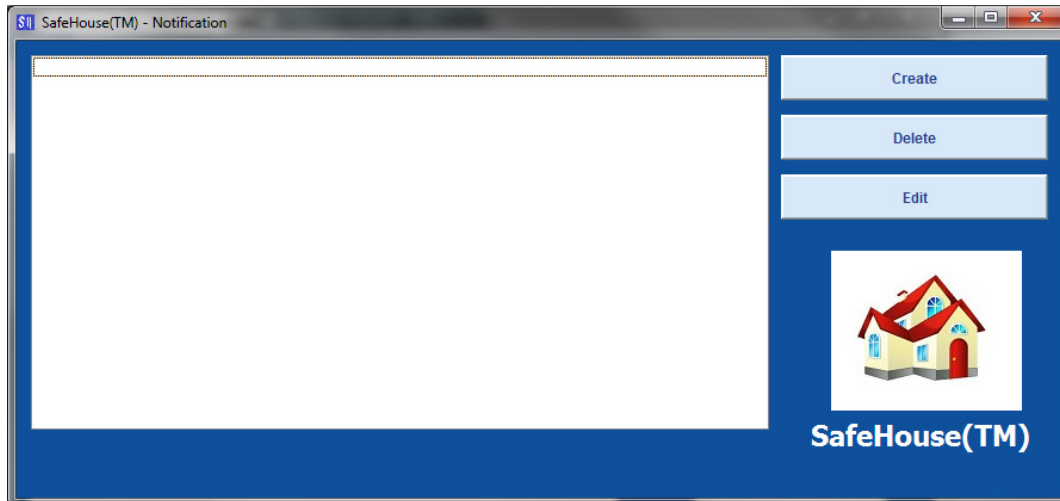


Figure 49 - Notification Screen of the Desktop Application.

Let us say that the user wants to create a notification by SMS every time the presence sensor in his/her living room is on (there is someone in the living room). Moreover, the user wants this notification to be sent only once per day and it should only be performed when the user is not at home (e.g. between 9:00 and 17:00).

Thus, the user presses the button **Create** and it appears a new screen to create the notification. In this window the user has to select the condition to trigger the notification. For that he selects in the first combo box the component **PRESENCE in Living Room**. Since it is a presence sensor and only can be used the word "is" as ruler, the user selects the value **on** in the third combo box. After this step, the user selects **SMS** as type and he/she inserts a cell phone number to where the SMS should be sent.

To complete the creation of the notification, the user just has to insert the initial time and end time when the notification should be sent, and finally the frequency **DAILY**. This entire scenario is shown in the Figure 50 below.

After filling in all the information, the user can finally press **Save** and this screen disappears right after and the user can see the brand new notification on the Notification screen (Figure 51 below). As a remark: as soon as the user creates the notification, this one will run immediately in the system if the condition, time interval and frequency are respected.

In the next section we are going through the Rules Screen. As we are going to see, both notifications, rules and users screens have the same scheme: the list in the left side and the options in the right side. Because of this fact, we are going directly to the creation of rules and users.

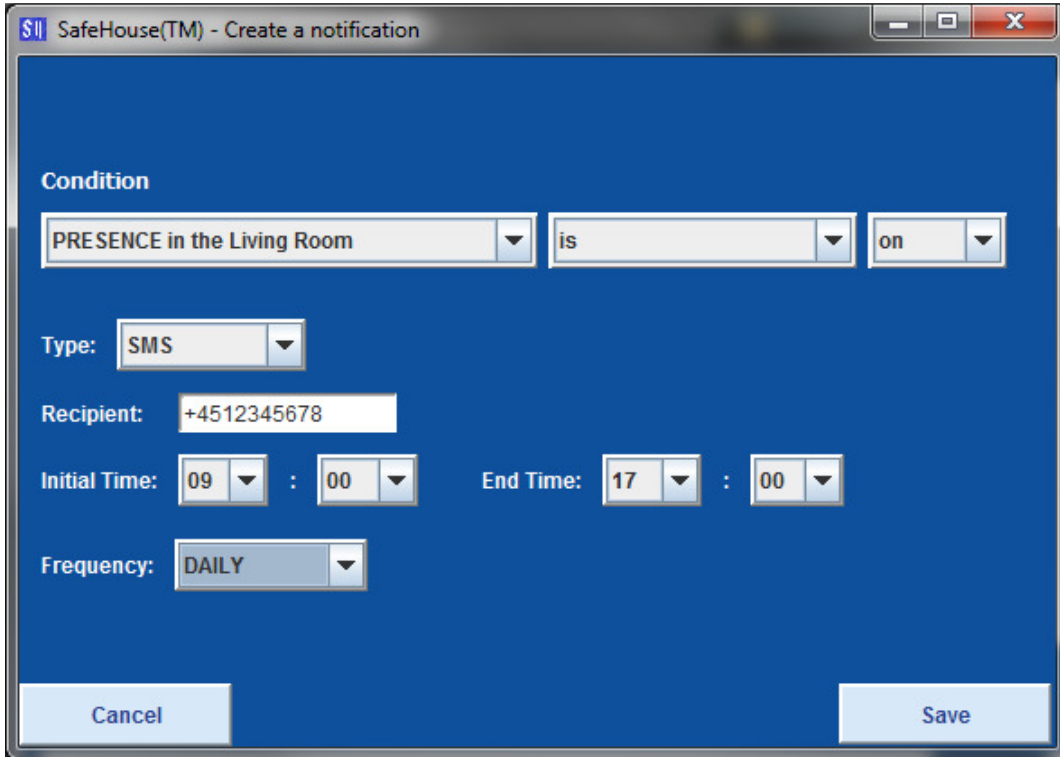


Figure 50 - Creation of a notification in the Desktop Application.

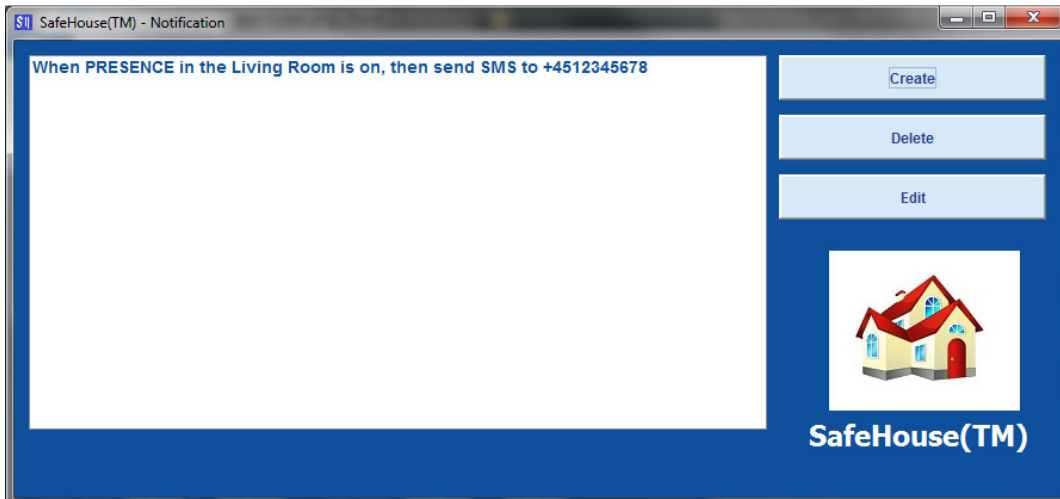


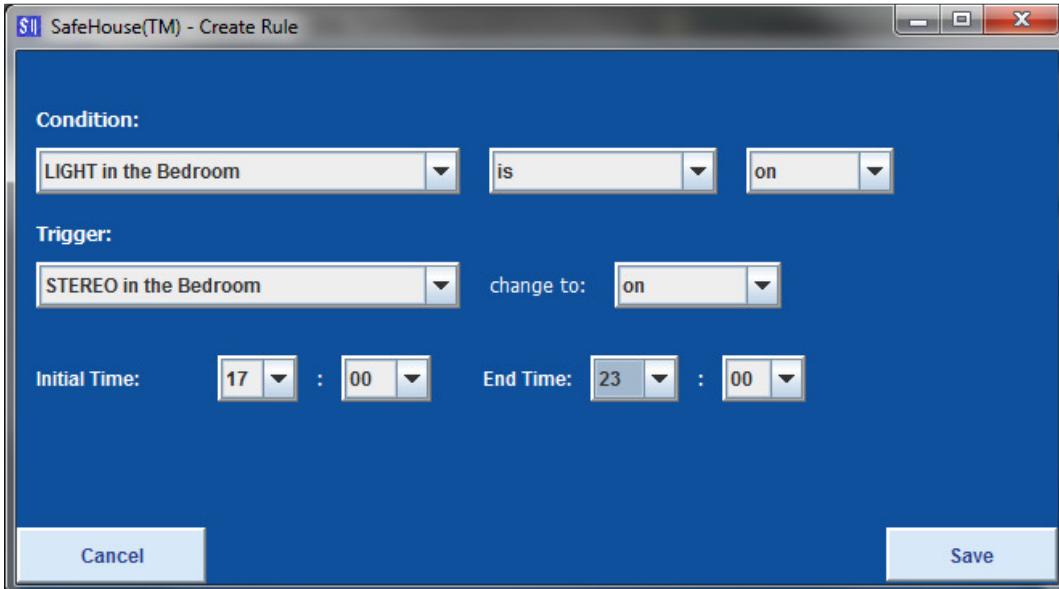
Figure 51 - Notification screen after creating a notification.

5.3.3. Rule Screen

The same scheme is used for the **rules screen**. Thus, if the user wants to create a rule, a window pop-ups and the user has to choose some information. Let us say that the user wants to turn on the stereo in his bedroom if the light is also on. Moreover, the user only wants this to happen when he/she is at home.

In this case, in the **Condition** he has to choose **LIGHT in the Bedroom**, **is** and **on** in the top combo boxes. In the **Trigger**, he has to choose **STEREO in the Bedroom** and change to **on**. Finally, the user selects 17:00 as initial time and 23:00 as end time, since it is when the user is at home after a day of work.

This scenario is represented in the Figure 52 below. A special attention has to be taken for conflicts with rules, as we have seen in the section 3.3.2 of this chapter (page 65). For instance, if after saving this rule, the user would create a new rule that turns on the stereo in case that the light is off in his/her bedroom, this would enter in conflict with the previous rule (different conditions in the same component would trigger the same action in the triggered component). Thus, an error message is presented in the screen saying that the rule is in conflict with an existing rule and then, it does not allow the user to create this new rule. This situation is represented in the Figure 53 below.



SafeHouse(TM) - Create Rule

Condition:

LIGHT in the Bedroom is on

Trigger:

STEREO in the Bedroom change to: on

Initial Time: 17 : 00 End Time: 23 : 00

Cancel Save

Figure 52 - Creation of a rule in the Desktop Application.

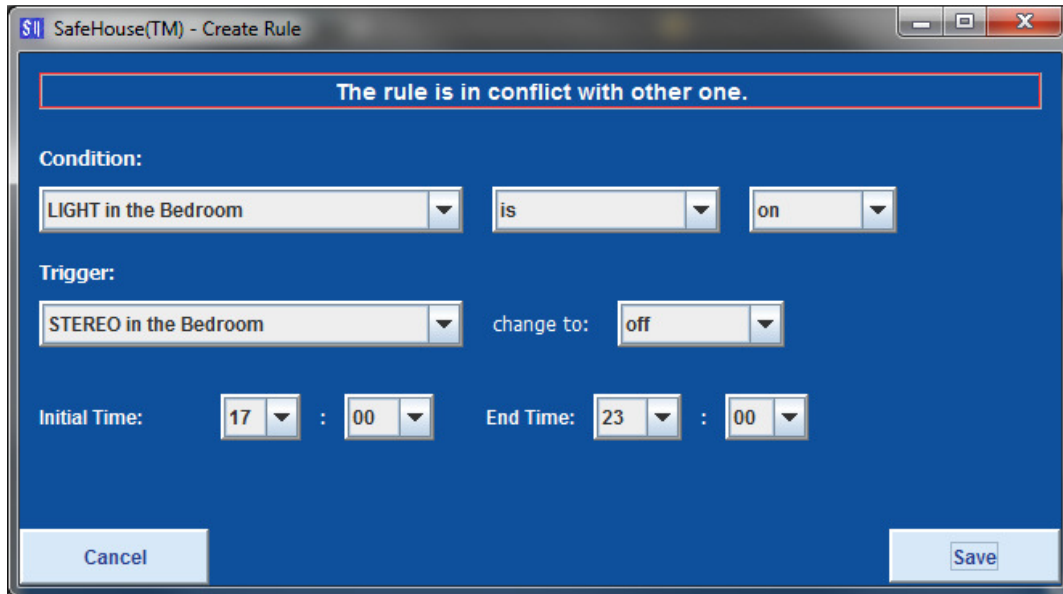


Figure 53 - Conflict in the creation of a rule in the desktop application.

In this case, the user can change some of the parameters of the rule or press **Cancel** and go back to the rules screen. The user can see the rules already crated in the rules screen (Figure 54 below).

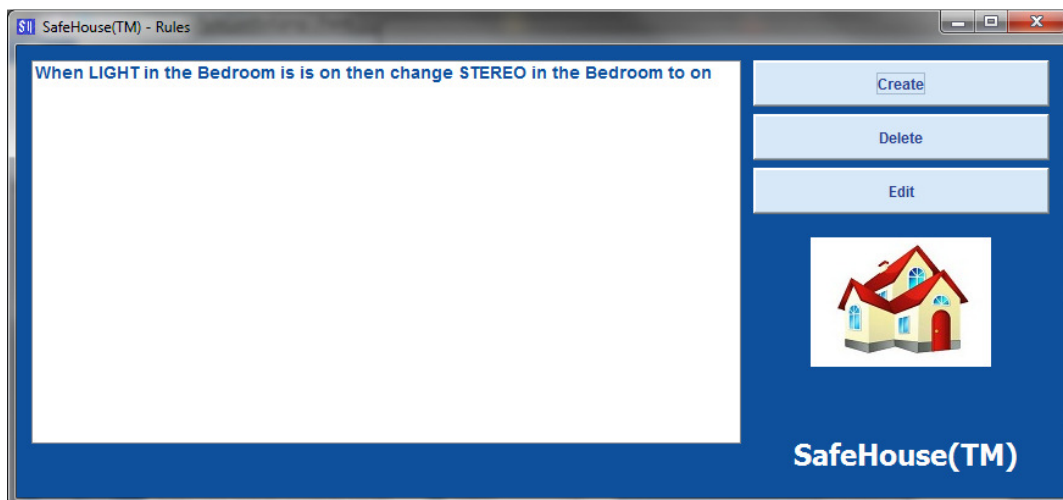


Figure 54 - Rules Screen of the Desktop Application.

5.3.4. User Screen

As said previously, the users screen keeps the consistency of the rules and notifications screens and the scheme of the elements are the same. Thus, if the super user decides to create a new user in the system he has to fill some parameters and also choose the permissions for this new user. Let us say that the super user wants to create a new user called **sergio** with password **sergio** as well. Moreover, the super user wants to give access to all the rooms in the house to this new user. This scenario can be seen in the Figure 55 below.

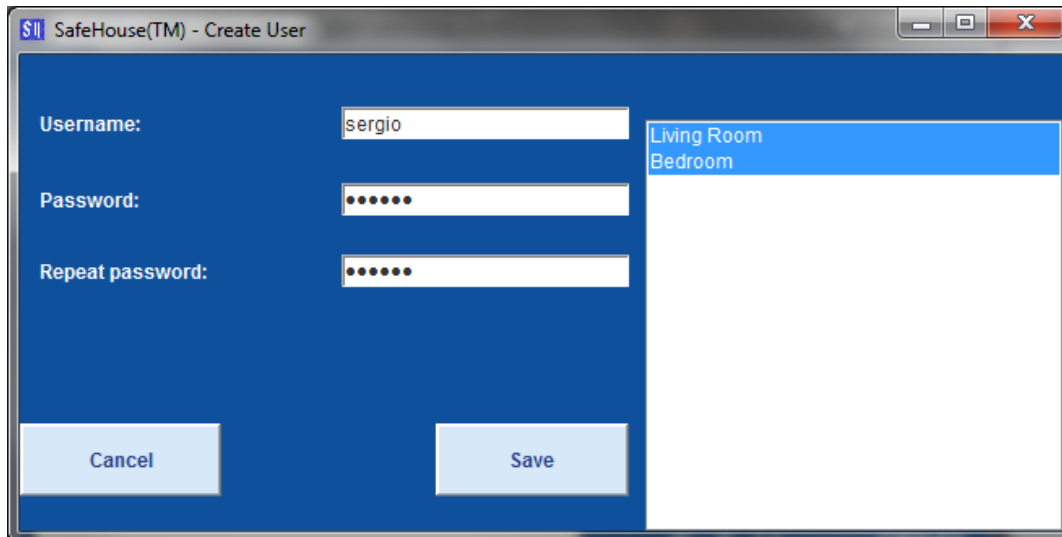


Figure 55 - Creation of an user in the Desktop Application.

Some errors can appear in this screen, according to the constraints of creating a user (section 3.2.2). First of all, every field has to be filled in. Moreover, both passwords have to be the same. Finally, it cannot be created a user with a username belonging to an existing user. Different error messages would appear depending on the situation.

Let us see an important case. In the users screen, if the super user decides to delete himself/herself from the list, the application does not allow that and then, it appears an error message in the screen informing the user about this fact. This is shown in the Figure 56 below.

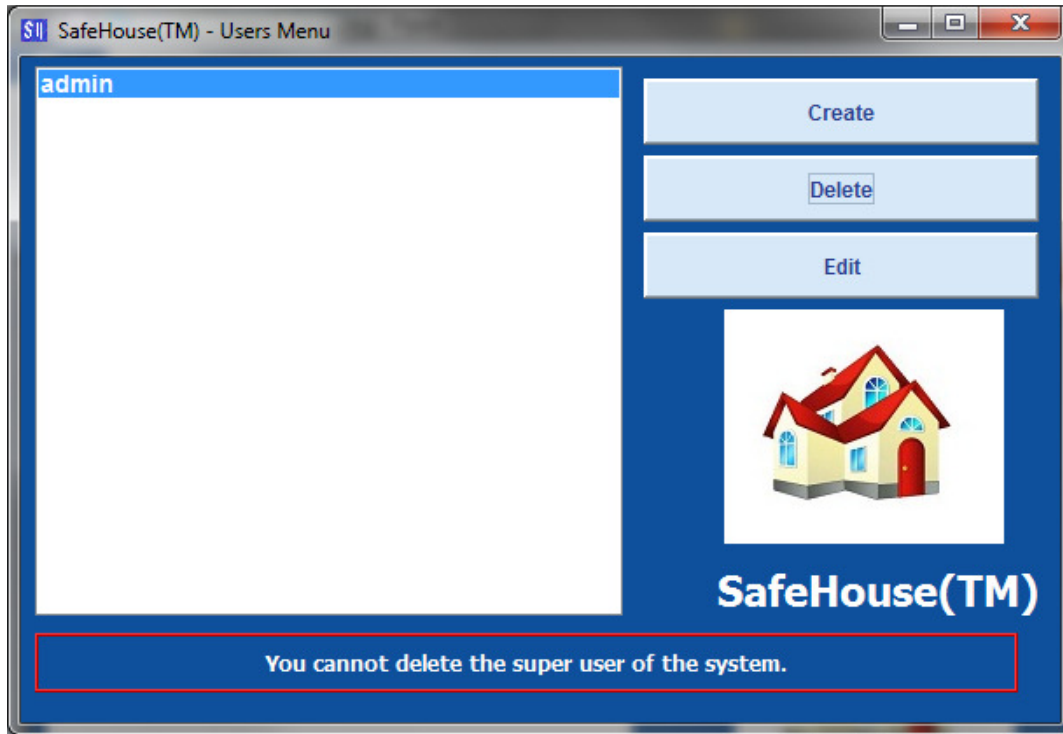


Figure 56 - Error message in deleting the super user of the system.

Last but not the least we are going through the last screen, the **Charts Screen**. As we are going to see this screen allows the user to visualize the status of the different components in his/her house.

5.3.5. Charts Screen

In previous sections we have seen that the system allows the computation of some simple statistics. These statistics are based on the average of values during a period of time. But this computation would not make sense if the system would not allow the user to visualize these statistics in a more user-friendly way, i.e. through charts. In the Figure 57 below we can see the charts screen in action.

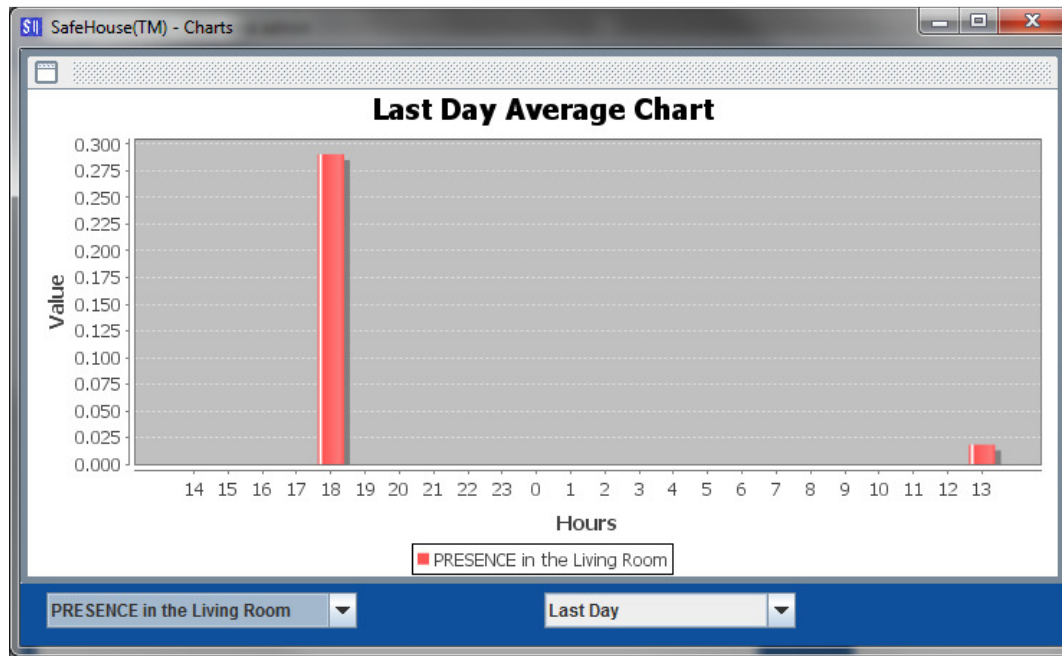


Figure 57 - Charts Screen of the Desktop Application.

As we can see in the vertical axis we have the range of values for that component in the last day. Obviously, for components with Boolean types, this can be seen as a percentage of time that that component was “on”. In the horizontal axis we have the range of hours for the last day, being the most eastern one the current hour of the day when the user is checking the chart. In the bottom we have the component that can be selected by the user. Automatically a new chart is generated and the user can see the chart for the new component selected. In the bottom right combo box the user can select to see the average of the last day or last month.

With this, we have concluded all the functionalities of the desktop application. We have seen the different screens that allow the super user to perform actions over the system and also some screens showing the constraints already explained in section 3 of this chapter when we went through the different structures of the system.

In the next section we are going through the mobile application that allows the user to interact wherever he/she is with the *SafeHouse* system and subsequently with the components of the house.

6. Mobile Application

In the desktop application we have seen that we have a management interface for the system viewing to a common user and to define the different structures to the super user. Therefore, the *SafeHouse* system would not be complete with a mobile application where the user can set and/or get in quick steps the status of some components. As we have seen before, the mobile application integrates a presentation layer able to receive the screen definition from the application layer. Nevertheless, it is still missing the elements that can be rendered in the mobile application and how the XML is associated with each of them.

Thus, we are going to start with the elements of the presentation layer that can be displayed on the screen and afterwards in the screens sequence of the mobile application.

6.1. Elements of the Presentation Layer

As we have seen in the section 4.2 of this chapter (page 73) there are certain elements that can be displayed on the screen and some elements that cannot be interpreted. Thus, only the most basic GUI elements were considered in the mobile application. As a remark, each screen in J2ME is represented by a Form object. The list of the elements is composed by the following elements:

- **Title:** the title of the form. This element is displayed in the top of the display.
- **Ticker:** it is an animated sub-title of the form. It is displayed right below of the title.
- **Choice Group:** or selection list, it is an element where the user can select one option (or multiple) among different options. Each choice group is composed by one label (the title of the choice group), the type (multiple or exclusive), and the different options, represent by a label and a field indicating if the option is selected or not.
- **String Item:** it simply displays a string in the form when information is needed to appear. It is represented by a label and a text.
- **Text Field:** when it is necessary that the user inserts some information for the application, a text field can be displayed in the screen. The text field is represented by the label, the text (where the user can edit it), the type (any kind of character or just numeric) and the size of the text.
- **Button:** the button is one the vital elements of the presentation layer. Clicking on a button will call the remote handler on the application layer. Thus, the button is defined by a label, a type (OK, Back or Exit) and a handler.

Moreover, except for the title, ticker and alert, the elements have an attribute id in order to identify each element and be able to update an existing element without deleting the existing one and creating a new one. Let us see now, in terms of XML, how each of these elements will look.

6.1.1. Title and Ticker

Both title and ticker have the same XML structure and they are represented by the following template:

```
<TITLE>title_text</TITLE>
```

```
<TICKER>ticker_text</TICKER>
```

Therefore, the only field that has to fulfill in order to render is the field between the identification tags (TITLE and TICKER). In case that the fields are empty, a standard title and ticker are rendered.

6.1.2. Choice Group

The choice group is perhaps, the most complex element in a form. The reason why is simple: it is a composition of options and these options are by themselves other elements. Let us take a look in the XML format of the choice group:

```
<CHOICE_GROUP id="cg_id">  
  <LABEL>cg_label</LABEL>  
  <TYPE>exclusive</TYPE>  
  <OPTION id="1">  
    <LABEL>option1_label</LABEL>  
    <SELECTED>True</SELECTED>  
  </OPTION>  
  ...  
  <OPTION id="N">  
    <LABEL>optionN_label</LABEL>  
    <SELECTED>False</SELECTED>  
  </OPTION>  
</CHOICE_GROUP>
```

In first place and as we have seen before, the choice group is identified by an id in order to be updated afterwards. Also the options are identified by an id for the application layer being able to update certain options in the same choice group. The type of the choice group can be one

among two: exclusive and multiple. The exclusive type means that only one option can be selected at any moment. The multiple type means, trivially that any possible number of selections is allowed. This number can be 0 or N, being N the number options in the choice group.

6.1.3. String Item

The string item is a simpler element in the presentation layer. It has also an id to identify it, but besides that, it is only identified by the label and text as it is shown in the following XML:

```
<STRING_ITEM id="stringItem_id">  
  <LABEL>label_of_the_string_item</LABEL>  
  <TEXT>text_of_the_string_item</TEXT>  
</STRING_ITEM>
```

Thus, as we can see, the application layer has to set these three fields in order for a string item to appear on the screen.

6.1.4. Text Field

The text field is a bit more complex than the previous element. It is represented by the id, the label, text, size and type. Let us take a look in the following XML:

```
<TEXT_FIELD id="textField_id">  
  <LABEL>label_of_the_text_field</LABEL>  
  <TYPE>any</TYPE>  
  <SIZE>N</SIZE>  
  <TEXT>text_inserted_in_the_text_field</TEXT>  
</TEXT_FIELD>
```

Where the type can have one of two values: any or numeric. Any other value inserted will not be interpreted by presentation layer and thus, the text field is not rendered. The size has to be a natural number and there is no constraint for the value of this number. Finally, the text is the text field itself, i.e. the text inserted by the user at any moment.

6.1.5. Button

As we have seen before, the button is the most important characteristic of the presentation and it is the one that allows the calling for remote methods in the application layer. For that, the name of the method has to be integrated in the button XML node. The format for the button is represented in the following XML:

```
<BUTTON id="button_id">  
  <LABEL>button_Label</LABEL>  
  <TYPE>OK</TYPE>  
  <HANDLER>onOkBtnHandler</HANDLER>  
</BUTTON>
```

The value of the handler node is the name of the method to be called in the application layer when the user presses this button. There is also the type of the button that can be one of three: OK, Back or Exit. The reason for this is how J2ME places the different kind of buttons in the screen, depending on each mobile platform the virtual is running.

We have seen the different elements that can be rendered in the screen, but we still did not see how these elements are used in the mobile application. This will become clear in the next section about the screens sequence.

6.2. Screens Sequence

In first place, it had to be defined the logic of the screens for the mobile application. It is important to say that this one was required to be simple and quick to access. Therefore, it would not be reasonable to have the same number of screens as we have in the desktop application. Since the target of the mobile application is to control the components in a glance, three screens were considered: a login screen where the user inserts his/her username and password, the house screen, where the user selects one of the rooms to accede, and finally, the room screen, where the user sees the different components of the previous selected room and can toggle the status of some of the components. The scheme of the screens is presented in the Figure 58 below.

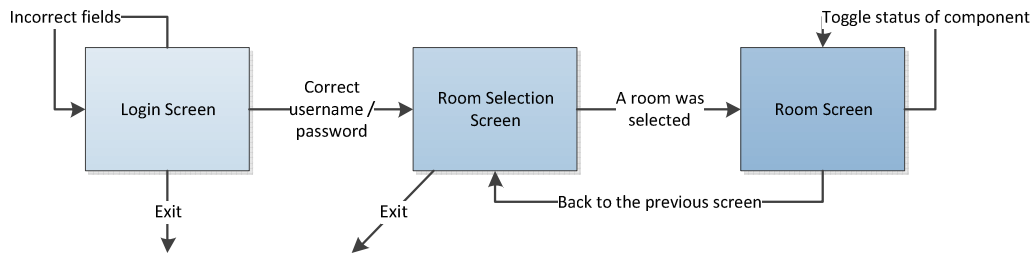


Figure 58 - Screens sequence on the mobile application.

As we can see, the sequence of screens is simple and no management options are available here. The user can simply see the components in the rooms that he/she has permission to access and change the status in the components with a **set** method. Let us start with the login screen.

6.2.1. Login Screen

In the login screen the user can see two different text fields: one to insert the username and other for the password. In case that one of these fields is incorrect an alert with an error message is presented on the screen. The user can also simply exit the application in this screen. This screen is presented in the Figure 59 below.

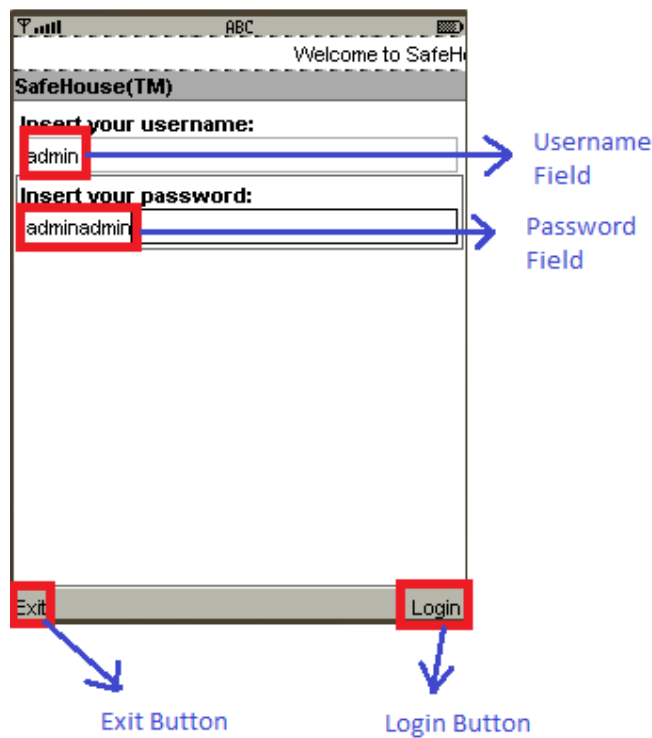


Figure 59 - Login Screen of the Mobile Application.

The error messages that can appear in this screen are the same as in the login screen of the desktop application. Thus, after the user inserts all the fields correctly he goes to the next screen: the House screen, where he can select one of the rooms in his/her house.

6.2.2. House Screen

Depending on the user that has logged in, it is shown the rooms of the house that accessible to that user. The allowed rooms are set up when a new user is created, as we have seen in section 5.3.4 of this chapter (page 90). The exception goes for the super user that has access to all the rooms. In this screen the user can select one of the rooms click **Go** or simply click **Exit** to terminate the application (gracefully). In the Figure 60 we see an example where the user can select between Living Room or Bedroom. After this step, the user goes for the third and final screen where he can see the status of components and toggle some of those statuses.

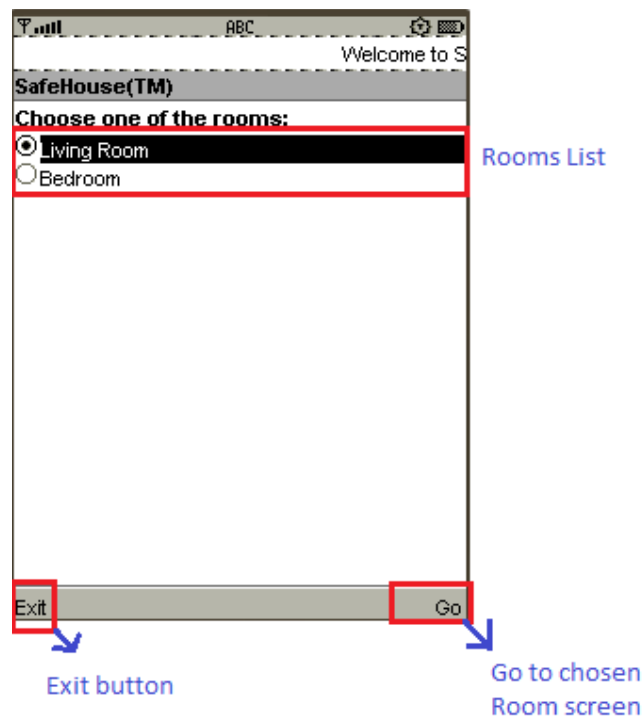


Figure 60 - House Screen of the Mobile Application.

6.2.3. Room Screen

Finally, we arrive to the final screen. This screen has a list of the components in that room and in the key **Menu**, we can select to toggle the status of components with the method **set**. Otherwise, we can simply check the status. After performing the actions the user wants, he can simply click **Back** to go again to the house screen. Obviously, the elements presented here are really simple and without any icon associated with them. The reason for this is to keep the flexibility of this platform. At this point, the presentation layer can be used with any system using this architecture. Other screen elements could have been considered, such as 2D or 3D Graphics. As we are going to discuss in the Conclusion chapter, this can be considered as future development.

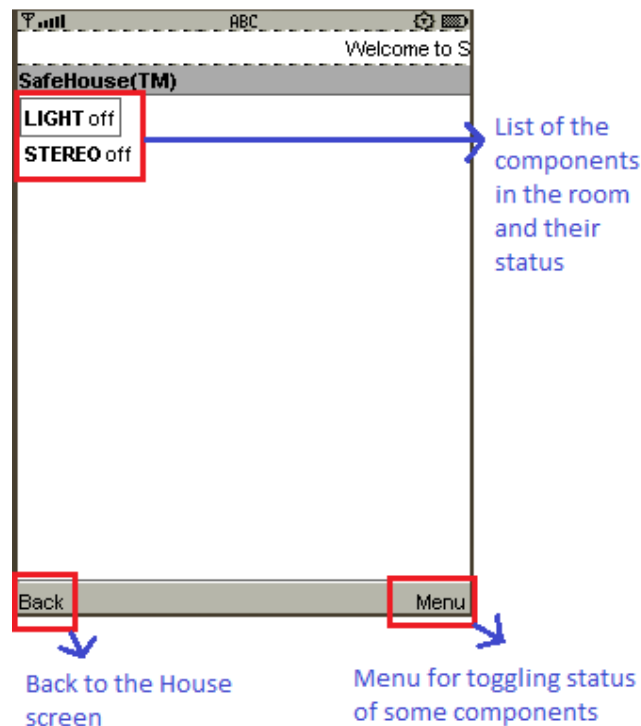


Figure 61 - Room Screen of the Mobile Application.

Therefore, we finalized the Mobile Application in both aspects: which elements are rendered in the screen and what is the sequence of the screens. All the logic behind the mobile application and particularly, the presentation layer was already discussed in previous sections (2.3 and 4.2).

7. Conclusion

With the Mobile Application we conclude the Design chapter. Let us remember that we have seen in first place the choices made in the architecture of the system. We have seen the concept of the remote server and its advantages and disadvantages. In order to tap some of the disadvantages, it was chosen the concept of Home Server as final solution.

Furthermore, we went through the communication protocol in different points: the structure of the XML, the structure of the layer and the interaction between layers. After this section, we went through the Home Server and the different processes running on it. Finally, and to conclude, we have seen the Desktop Application that allows the super user to manage the *SafeHouse* system and the Mobile Application that allows any user to access to the components in the house, wherever he/she is.

Thus, in the next chapter we are going to discuss the Implementation of the *SafeHouse* system.

IV. Implementation

All the development process was made following the SOA programming principles, from section 2.1 of Analysis (page 19), and therefore, all the software blocks in the *SafeHouse* were developed in that way. During the Design section we went through each view of the system, in a general view with the architecture and in detailed views with the structures, processes running and the different GUIs by last.

Nevertheless, there were **problems faced** during the development process and it is going to be discussed what was done to solve them in the first sub section and which problems remain without solution. Hereafter, we are going through the **deploying process** of the system: what the user needs to do in order to install the *SafeHouse* system. Finally, to conclude this section, we are going through the **IDEs** used during the development process.

1. Problems Faced

As in any development process, there are always problems that the development team has to adapt to and overturn on them. This happens in Engineering as it happens in any field of science. There are always unpredictable variables coming into the implementation process. During this master thesis issues were faced and for some of them a solution was not found, mainly due to time constraints. We are going through the problems occurred in this section. Let us start with one of fundamental components in the system: the embedded device.

1.1. Limitations on the Embedded Devices

There are some limitations on the embedded devices that in a real system would compromise the overall robustness. In first place, the RAM of the PIC used in this project is small: 2048 bytes. For instance, let us consider an embedded device with 5 components attached to it. Let us assume that each component is just a light. Thus, each of them would have the follow XML format:

```
<COMPONENT id="LIGHT 1">  
  <TYPE id="bool"/>  
  <METHODS>  
    <GET/>  
    <SET/>  
  </METHODS>  
</COMPONENT>
```

The number of characters in the XML from each component would be 84 characters. Multiplied by 5 would cause the data stream to have 420 characters. If we add the root node and the header, the stream will have 483 characters. And this XML has to be stored in a buffer at some moment before being send, what can cause a segmentation fault or overlapping in some cases. The reason for that is that the application running on the embedded device is at the same time storing other structures or buffers. Moreover, the heap or dynamic memory can only have a percentage of the RAM.

This problem would compromise the robustness of the embedded application. To avoid this kind of situations from the embedded layer, the application layer only sends a get/set to one component at each time. Obviously, it is not the fastest solution, but it avoids overlapping or segmentation fault in the majority of the cases.

Another solution from the application layer side and integrated in the BSCOM library, it is the fact that in case of failure in reading an answer from the embedded layer, the application layer tries a certain number of times to obtain an answer from the embedded layer. This is part of the communication protocol established for the *SafeHouse* system. If no answer is received, the application layer continues for other components and eventually for other devices. It was important to develop in robustness the communication protocol, exactly to prevent these cases.

Another limitation on the embedded devices is that they are battery-driven, i.e. if the main energy source is interrupted in some case, they stop immediately working. Due to schedule and since this master thesis does not focus on the energy consumption, but yet in a robust wireless platform for communication between devices and the home server, this problem was put aside and not solved. Nevertheless, this is going to be discussed in the Conclusion section as a future improvement.

Finally, using the Bluetooth communication inside the house raises another limitation in the system: the range. Both Bluetooth modems used in this project (in the desktop station or in the sensor boards) are already 2.0, capable of reaching other device in a range of 100m. Unfortunately, it has been proved that Bluetooth has a weak penetration when it reaches walls, being difficult or impossible to communicate with other devices. A possible solution for this would be the use of Bluetooth hubs: Bluetooth modems in strategic points, such as doors, to redirect traffic from a source modem to a target modem. Another solution could be considered: the use of RFID in the project. This would avoid better the problem of wall penetration and even the range could be up to 1000m. Once more, it is important to say that this master thesis was focus on a proof of concept, i.e. to develop the neck bone of this wireless platform for houses. Undoubtedly, in a further development, it would be considered this solution.

As conclusion, it was faced the memory problem as main problem on the embedded devices. Even though there is not a completely stable solution from the embedded layer, but the communication protocol established prevents crashing in case an answer is not received.

In the next section we are going in more detail to the file structure of the *SafeHouse* system, concretely in the Home Server.

1.2. File Structure of the Home Server

As we have seen in the section 3 of Design chapter (page 59), there are different structures manipulated constantly during each session of the Home Server. Obviously, those structures have to be stored and loaded from some place in the file system in order to keep the structures from session to session and also to allow less dependency between each service. Thus, two possible solutions to store the structures were thought during the implementation process:

- **Database:** A database containing different tables, where each table would represent a different Java class in the program. For this reason, the structures were constructed in such a way that the transposition to a database would be easy. Moreover, the database would provide mechanisms of query and most of them have synchronization of access between different threads, i.e. the database server does not allow two threads to access simultaneously to the same table at the same moment.
- **File:** A file is the most common way in an operating system or any application to store information. In general, a file is used to store less important information for the application, such as application logs. The reason for this is simple: to store information in a file it has to be defined a file format, such as XML, and each access to that file would require a parsing of the information contained on it. If the application accesses many times the file, this can become slower than a database. Also, storing in a file would not provide mechanisms like search information by a certain index, as in a database happens.

At a first glance, the database is the best solution and by all the means it would be the best solution for a final product. Nevertheless, an alternative solution was found: Java serialization. This mechanism allows that all the classes that implement the interface `Serializable` (included in the J2SE framework) are possible to be stored directly into a binary file, storing all the fields, references to other objects and methods in a binary file with any extension we want. Nevertheless, a problem arises with this: synchronization between threads. Since the *SafeHouse* system has multiple threads running at each time, it can happen that at some moment two or more threads are accessing the same object. This may cause inconsistent information being written to the file. Let us see the example represented in the Figure 62 below when two different threads access the same object. Thread A starts to read the object. In the meantime Thread B reads the object and it performs a quick action over that object, storing right after that object. In the meantime Thread A is still using the object and just stores it afterwards. This would cause inconsistent information in the stored object. Whatever Thread B has done, it is overlapped by what Thread A has done.

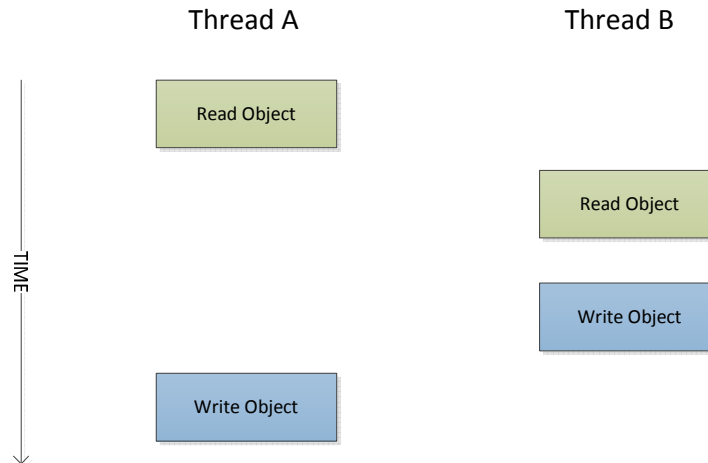


Figure 62 - Bad synchronization when two threads are accessing the same object.

Hence, a synchronization mechanism had to be created. As we have seen in the section 3 of Design chapter, all the structures contain an IO object with two methods: *writeToFile* and *readFromFile*. Thus, the first step was to create a lock system. When a thread is accessing the file either to read or write, it takes that lock and all the other threads wait till the initial thread frees the lock. The question was exactly when the thread should free the lock. For instance, there are two different cases in this I/O system:

1. A thread simply reads the object to extract some information;
2. A thread reads the object, changes that object and after it writes the object into the file.

Therefore, in the first situation the thread should free the lock immediately after it reads the object. There is no use to keep the lock for itself. In the second situation, in other hand, the thread should only free the lock when it finishes writing to the file. Some modifications were required then in the *readFromFile* method: it had to receive as an argument a Boolean to indicate if the thread is going to write afterwards or not.

With this solution, the problem was solved and the Figure 63 below represents the synchronization created.

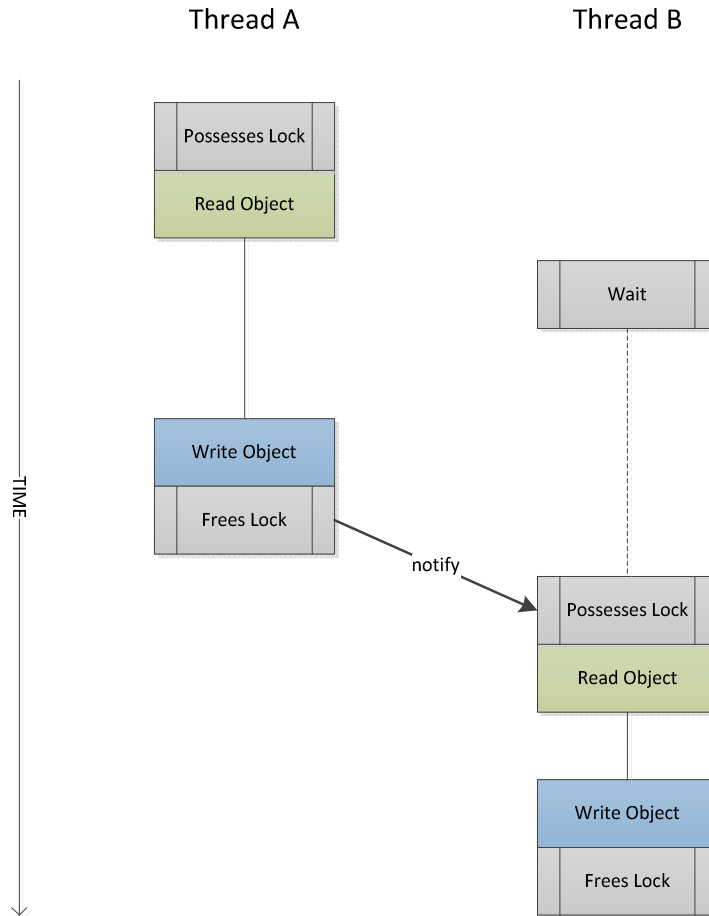


Figure 63 - Good synchronization between threads: The Thread B waits for Thread A to finish the transaction.

A problem may come if a thread possesses the lock for a long time: the other threads have to wait till that thread is finished for unlimited period. Another disadvantage may come with multiple concurrencies for the lock. Since a semaphore system is not implemented, there is not order in the queue to access the object. This will be topic of discussion in the Conclusion section. In the next sub section we are going through to one of the other issues faced during the development stage: the SMS notification. As we are going to see, this was not a true issue, but decisions had to be made in order to concretize it.

1.3. SMS notifications

As we have seen in section 4.3.3 of Design chapter (page 77), the Home Server triggers notifications either via e-mail or via SMS. Since e-mail is a free service and Gmail is the main e-mail service worldwide, it was chosen to use the Gmail SMTP as the e-mail bridge to trigger notifications. But SMS does not work in the same way and the network providers charge for each SMS sent. Obviously, there are some special packages in the network providers that allow a customer to send a SMS for free or for a really cheap price. Either of way, an issue arises with this: it is necessary to integrate a mobile phone in the system to send SMS to the user. This option is represented in the Figure 64 below. As we can see, it would be necessary to use a cell phone exclusively for sending SMS. This can be considered as a waste of resources and probably not so many users would accept a system like this.

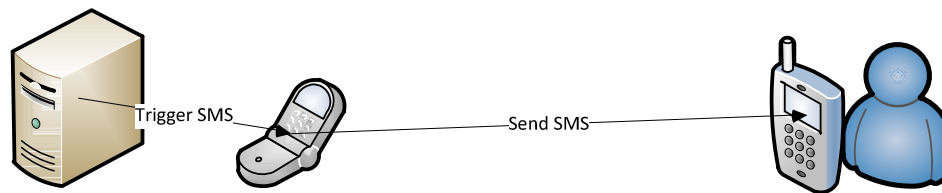


Figure 64 - SMS notifications via an intermediary cell phone.

Therefore, other solution was thought and conceived. If the home server has internet access, why not use this internet access to send a SMS through a web service? There are different companies providing this kind of services. For example, Skype is the most known one. It is possible to send a SMS through Skype using for that a HTTPS Post. For the case of the *SafeHouse* system, the company VoipCheap was chosen. The reason for this is simple: it is practiced reasonable prices per SMS – in Denmark it costs 3 Euro cents (0.22 Kr) per SMS. In this way and as proof of concept for the *SafeHouse* system, this web service was used. Thus, in each notification triggered by the Home Server, it is sent an HTTPS post with the following data:

- **Username:** the username of the account in the VoipCheap service.
- **Password:** the password of the account in the VoipCheap service.
- **Sender number:** the number which will appear in the recipient cell phone.
- **Recipient number:** the number to where the message is sent.
- **Message:** the message itself of the SMS.

In a real product, other alternatives can be thought. For instance, this SMS service would be included in the subscription of the *SafeHouse* system. Nevertheless, for this master thesis the topic of subscription is not relevant. Thus, we have seen the main problems faced during the implementation stage. In the next section, we are going through the deployment of the system.

2. Deployment of the System

In first place and remembering the whole report, the *SafeHouse* system is intended to be plug-n-play. This was one of the requirements and it was successfully achieved. Nevertheless, few steps have to be taken into account when deploying the system.

In first place, the user has to possess a Gmail account with the SMTP open for external applications. Moreover, the user has to possess a VoipCheap account with credit on it. In other hand, the Home Server has to possess a public IP address. An alternative to this would be to put the Mobile Server running on a remote server and communicate with the rest of the Home Server through SSH.

Therefore, the Home Server (HS) works with TCP to serve the mobile applications. Obviously, it cannot be used the port 80 or other more common ports, since these ones are reserved to HTTP and other web protocols. Thus, an extra port has to be open in order for the HS to work. This has to be set from the *SafeHouse* providers in the code of the application before deploying in both HS and mobile applications.

A last thing has to be done from the *SafeHouse* providers: the integration of the components on the embedded devices and further integration of these ones in the user's house. This step concludes the deploying process and then, it is ready to be installed and used.

Nevertheless, there two system requirements for the user's desktop computer:

- Java Runtime Environment installed.
- Bluetooth integrated or a Bluetooth USB stick attached to it.

For the mobile phones, there are few requirements as well:

- 3G and Bluetooth integrated. Bluetooth is optional, but 3G is strictly necessary, otherwise the mobile application runs incredibly slow.
- Mobile Information Device Profile (MIDP) 2.0 with CLDC 1.1.

If these requirements are fulfilled, the system is ready to be installed and used. The only advice goes for the fact that the super user should customize the name of the rooms and set up all the necessary users at the first time he starts up the desktop application. No more steps are necessary to follow to install the application.

Finally, in the next section we are going briefly through the IDEs used for the development of the *SafeHouse* system.

3. Integrated Development Environments (IDEs)

Nowadays, developing code without using any existing framework or IDE would be impossible. Frameworks provide easy access to functionalities to the developers that otherwise they would have to program from scratch. IDEs helps developers to keep the code organized in a project, to have code completion, to compile the code and to run the application.

3.1. Netbeans IDE

Netbeans is an IDE developed by Sun very similar to the so known Eclipse IDE. Netbeans had allowed the development of the *SafeHouse* system both for the desktop application and mobile application. Java Standard Edition (J2SE) and Java Micro Edition (J2ME) were installed together with the Netbeans IDE and they allowed the development of the desktop application and mobile application, respectively. Moreover, for the Java Micro Edition was installed as well the Java Wireless Platform, allowing the development of wireless functionalities, such as Bluetooth or Internet connections. For J2SE had to be installed a third-party library called *bluecove* (Intel Research, 2006). This library allowed the development of a Bluetooth server integrated in the Home Server, what otherwise would be impossible to develop.

In the Figure 65 below we have a perspective of the Netbeans IDE 6.8.

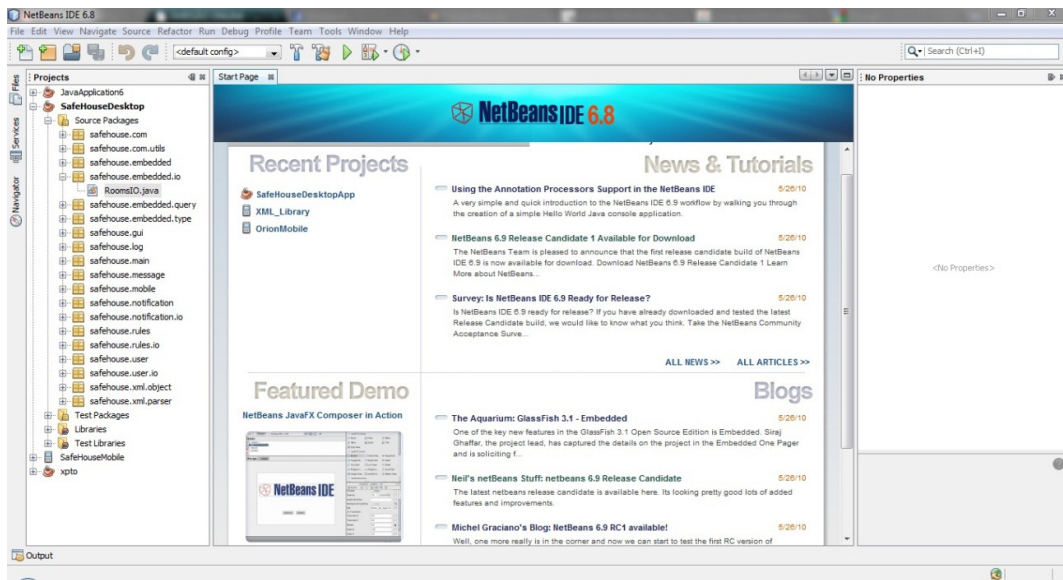


Figure 65 - Netbeans IDE 6.8.

Obviously, this IDE cannot be used for development on the PIC microcontroller. This is achieved using other IDE: MPLAB.

3.2. MPLAB IDE

MPLAB IDE allows the development for PIC microcontrollers. Moreover, it is possible to even control memory parameters, such as the heap size. This IDE also allows the developer to program or debug the PIC using for that the PicKit2, connecting the computer with the microcontroller. This IDE is also very important when we want to see how much memory is being used by the application and how much memory is left. It is also possible to check memory positions while the application is running on the PIC. In the Figure 66 below we can see a screen shot of the MPLAB IDE.

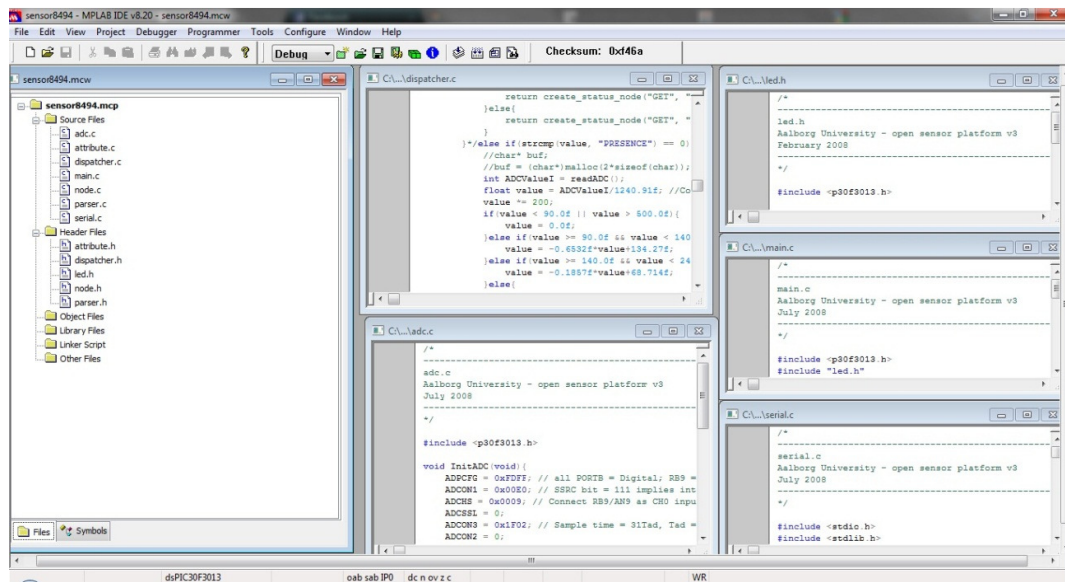


Figure 66 - MPLAB IDE v8.20

Nevertheless, this IDE does not include some important features such as code completion or auto-indentation. The lack of these two features is a huge drawback when a developer is developing an extensive code. Therefore, during the development process an initial version was always developed on a Linux system with a standard GNU C compiler. This allowed also using programs to check memory leaks, such as Valgrind. Just after this step, the code was exported to the MPLAB IDE and made the necessary modifications to work with real sensors and actuators. With this, it is concluded the IDEs used in this project.

4. Conclusion

Therefore, during the Implementation some problems were faced and then, solutions for those problems were presented in the section 1. In the section 2 was explained the deployment process from the *SafeHouse* providers side and also from the user side. Finally, the Implementation was concluded with the IDEs used during this master thesis.

In the next section we are going through the testing made to the *SafeHouse* system and which results were obtained from that testing.

V. Testing

Any development product would be completed without a testing stage of it. It is important in different ways and there are different types of testing. First of all, testing is important to check if the product meets the requirements before the design and implementation processes. This is even more important in a commercial product and can predict the success or the failure of a project. Testing is also important to know if all the functionalities are doing what they are supposed to, i.e. if they were implemented according to what was established during the design process.

Thus, there are different kinds of testing that can be performed on a software product:

- *Software performance/load testing*: this test is important to know exactly how long it takes for the product to load all the necessary components and to know the general performance of it during its life cycle.
- *Stability testing*: this test checks how the software reacts after being continuously working for a certain time.
- *Usability testing*: this test is made with a group of users to test their acceptance and adaptability to the GUI and the functionalities of the system.
- *Security testing*: this test is made to check if a program prevents attacks from hackers to the system.
- *Internationalization testing*: this test serves uniquely to check if the program adapts to different cultures and languages.

In the context of this master thesis, there are two important testing units to be made: Software performance/load testing and Stability testing. Let us not confuse Security testing with safety of the house. Security testing is focus on encryption of user data, not to prevent possible intruders in the house. Usability testing could have been made, but due to schedule, it was decided to leave aside. Internationalization testing would be also useful in a situation where a development team wants to commercialize a product in different countries, what is not the case of this master thesis.

Therefore, we are going through the two different testing units in the next two sections. It is going to be tested software performance and loading on both desktop and mobile applications, but obviously, the stability testing will only focus on the Home Server, i.e. desktop application.

1. Software Performance/Loading Testing

This section will be divided into three sub sections: Loading Testing for the Desktop Application; Loading Testing for the Mobile Application; Performance Testing for the Desktop Application. In each of them, the tests are going to be shown and then, the results from each test. Finally, each test is concluded by a brief conclusion.

1.1. Desktop Application Loading Testing

As we saw in the section 4 of Design chapter, the first step of the Home Server is the discovery of embedded devices and further update of the structures of the system. This takes a considerable time and there are different situations that affect this time. Therefore, and since there were only available two embedded devices, it was tested three different situations: when there were no embedded devices available; when there was one embedded device available; and when there were two embedded devices available. These three situations were tested in two different ranges: 2 meters far and 10 meters far. Thus, there were six different combinations for this testing and each of these combinations was tested 10 times.

In the Figure 67 below we can see the different results for the situation when the server is two meters far from the embedded devices. We can see already a curious detail: the server takes less time to load when all the devices are available. This happens because in the Bluetooth protocol stack the devices have to be paired between each other before a communication takes place. Moreover, in each discovery for devices, the server searches in the range available the devices already paired. Obviously, if these devices are available, it takes less time. Otherwise, the server will take a few moments more because it would be still searching for them.



Figure 67 - Loading Testing for the Home Server when this one is two meters far from the embedded devices.

In the Figure 68 below we can see the same situation happening. More interesting is the fact that the time taken when the server is 10 meters away is basically the same as when it is 2 meters away. With this, it is concluded that the distance does not affect the discovery or the connection, at least in a short range.

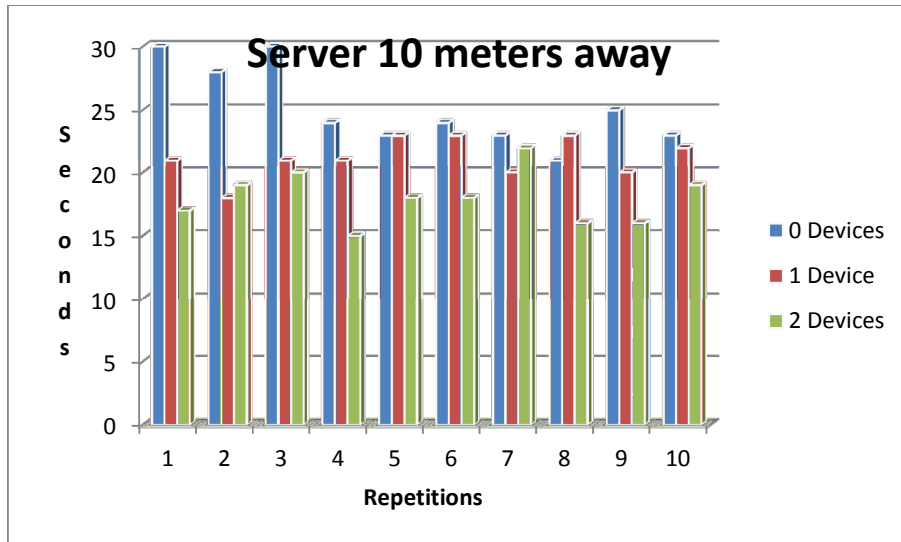


Figure 68 - Loading Testing for the Home Server when this one is 10 meters far from the embedded devices.

We can see the average of the results for each of the situations in the Figure 69 below. Comparing different distances, we see that only takes few milliseconds more when is more far away. About the number of devices, as more as available devices we have from the pre-paired devices on the home server as less time it takes.

DISTANCE	0 Devices	1 Device	2 Devices
2 meters away	24.7 sec	20.7 sec	17.4 sec
10 meters away	25.1 sec	21.2 sec	18 sec

Figure 69 - Average Results for the Loading Testing for the Home Server.

Moreover, the time taken to load the system can be considered reasonable if we take into account that with this time the server allows the plug-n-play characteristic of the *SafeHouse* system. The next testing unit will focus on the performance of the Home Server.

1.2. Desktop Application Performance Testing

We have seen in the previous unit testing that the Desktop Application loads and it updates all the structures in reasonable time duration. Nevertheless, it is important also to test how long the Home Server takes to perform a loop in the program. Let us remember from 4.3 that the Home Server performs three actions in each loop: checks the status of every component in the system; it checks the rules and performs the triggers; it checks the notifications and sends a SMS or e-mail. Thus, what is going to be tested in this unit testing is the duration of each loop l a total of N loops, making an average of those N loops. Moreover, it is going to be tested nine different situations:

1. System without any rules and notifications.
2. System with one rule and no notifications.
3. System with two rules and no notifications.
4. System with three rules and no notifications.
5. System with four rules and no notifications.
6. System with four rules and one notification.
7. System with four rules and two notifications.
8. System with four rules and three notifications.
9. System with four rules and four notifications.

The number of loops (N) chosen was 100. This value was selected empirically to allow a correct measurement. Thus, in the Figure 70 below we have the average results (in milliseconds) for each of the nine situations.

Situation	Average
1	806
2	867
3	883
4	910
5	915
6	1020
7	998
8	1201
9	1196

Figure 70 - Performance Testing in the Home Server.

As we can see, the system stands well a situation with a good amount of rules and notifications. The variations in the results are not significant. The reason for this is that the most slow operation in this “looping” is the loading and storage of files, and this loading and storage is done a constant number of times. It can be concluded one important thing with this performance test: if the user performs a change of status in his mobile application it will take at most 1.2 seconds for the operation to concretize from the server side.

In the next section we are going through the Loading Testing in the Mobile Application in different situations.

1.3. Mobile Application Loading Testing

A Software performance/load testing would not be complete on the *SafeHouse* system without a loading test on the mobile application. This is important to know how long it takes for the user to access his/her house from different places. There were tested four different situations: Bluetooth in a short range (10 meters), 3G in a short range (10 meters), 3G in a medium range (200 meters) and 3G in a long range (5000 meters). Unfortunately, the Wi-Fi network at AAU did not allow the connection to the server, most probably because the installed proxy at AAU does not allow that.

To concretize this series of tests the server was running in the group A6-311 at Aalborg University and for the short range situations the mobile phone was near the entrance of the room. In other hand, for the medium and long ranges the mobile phone was near the entrance of building and in the center of the city of Aalborg, respectively. The chosen mobile phone was a Nokia N95.

First of all, when using Bluetooth the mobile application takes a long time to start to receive information from the Home Server. This happens because each time the Mobile Application starts, it makes the discovery for Bluetooth devices and this step takes a long time to finish (around 60 seconds). A possible solution for this is to store the Bluetooth address of the Home Server after the first session of the Mobile Application. Nevertheless, after the discovery is made, the connection is quite fast, taking around 2 seconds to load each screen.

In the 3G communication, the mobile application takes few seconds to establish the connection (around 5 seconds) in the different cases (short, medium and long ranges). After that, it takes few seconds as well to load each screen. Actually, it took slightly more time when the mobile phone was indoors, but nothing considerable for the case. Obviously, this scenario was not tested in the limit, i.e. it was not tested with a considerable long distance, for example, 1000km or more. It would be interesting to test in those situations how long it takes to load each screen.

The conclusion of this testing unit is one: the use of Bluetooth between both applications is not a main advantage if the discovery is made every time the mobile application loads. Thus, this fact can be considered as a future improvement on the *SafeHouse* system.

Therefore, in the next section we are going through the stability testing on the Home Server (Desktop Application).

2. Stability Testing

Finally, to conclude the different unit testing, it had to be performed a test to check the stability of the Home Server. Therefore, the server was running uninterrupted during a period of 48 hours. During this period different situations were tested: creation of new rules and notifications; deletion of rules and notifications; creation and deletion of users; and customization of the rooms.

During those 48 hours the server performed 138019 loops with an average in each loop of 1252 milliseconds. Therefore, the system achieved the expected stability that had showed in the loading and performance unit testing. This test was far the most important. Perhaps the performance in some situations (many rules/notifications) can decrease a bit, but the system keeps its stability during all the time, meaning that there are no “holes” in the software. Moreover, this becomes more important when we are referring a system that deals with our house and any user would expect the system to run constantly all over the time.

With this, this section is concluded. We have seen that the performance and loading are acceptable in different situations, except when it is used Bluetooth as medium of connection between the Desktop and Mobile Applications. Moreover, the stability testing proved that the system is able to run for a long period of time without breaking up.

In the next section – Conclusion – we are going through the final conclusion of this master thesis and which future improvements can be considered.

VI. Conclusion

In this chapter we are going through about what it has been concretized and which blocks or functionalities did not go as it was expected, comparing to the initial expectations for this project. We are going through each of the requirements established on the Problem Formulation in the end of the Analysis section (section 6).

The system achieved the plug-n-play characteristic, fundamental to make the life easier for the user. Not only the components are updated in each start-up of the system, but also the existing rules, notifications and users are updated with the new components and rooms. The only thing required from the user point of view is the customization of the rooms. This small step requires from the user some attention to see where the devices are at home, but even though, this step can be made with the help of a member of a company selling such system.

The system allows the user to automate his/her house at any moment by creating new rules or deleting existing ones. Even though the user should have the control of the rules that he/she creates, but we cannot forget that there are situations that the system should not allow the user in creating rules that enter in conflict with existing ones. We have seen the cases these conflicts happen and more situations could be considered, e.g. the chain conflict. Nevertheless, the user can also program the rules to be followed in a certain time schedule. More situations in this schedule could have been though like allowing the user to select which days he wants the rule to run on that schedule.

The system allows the user to be notified when a certain event happens in his/her house. The notifications are working via SMS or via e-mail, with programmable schedule and frequency. The frequency had the goal of not letting a notification being sent every time since this will cause the user to receive several e-mails or SMS and in the case of SMS this would cost more for the user.

The system allows the visualization of charts about status of components of the system. This can be used for different purposes, being one of them the awareness to the user. Even though this requirement was not initially planned, but it was designed and implemented later on the project.

As we have seen during this project, only the super user of the system can control totally the system. This user can also create new users and set the rooms this new user can control from the mobile application.

Any user that logs in into the desktop application can also see live logs in the application. Thus, even though a normal user does not have access to edit the system, but he/she can see how the overall status of the system.

The mobile application was thought to give an easy and quick access to the user to change interactively the status of some components or simply check their status. Problem was found

in the communication through Bluetooth, taking this one several time to find the Bluetooth server on the Desktop Application.

About the Desktop application, this one can be exported in a web application as future improvement. This would be a critical advantage of such system, allowing the user to access the management of his house in any computer with internet connection.

Therefore, even though there are issues that remained without a complete solution. Such unsolved issues are inherits to a project of such complexity and with limited development time. Nevertheless, in general the requirements of the system were achieved.

1. Personal achievement

Besides what I have achieved as group VGIS 1027 or during the time that I belonged to the group VGIS 1020, I consider that I have achieved myself as well. It is always exciting and a good experience to develop a project of this dimension and I can say that I have done my thesis in a field that excites me particularly. Since the beginning there were methodologies of work and schedule that had to be adopted. Such methodologies were not unfamiliar to me but it is always a good point to improve them.

Moreover, even though the group initially created did not make till the end of the semester, but such situations have to be faced in different occasions during the life and this experience also contributed for my education as engineer and person.

In other hand, even though none of the technologies faced in this project were completely new for me, but I had to explore more deeply certain aspects of software development, comparing to the previous knowledge. The most challenging of this project was the development of the platform itself, i.e. how the components are interacting with each other and how they should react in different situations. For this, it had contributed my experience as internee in PDM Technology in the fall semester of 2009 and the lessons taken from there.

Finally, this project was also a good way to practice the topics that I have learned in Vision, Graphics and Interactive Systems during these two years of studies at Aalborg University, never forgetting the background of my bachelor in Information Systems and Computer Engineering at Instituto Superior Técnico, Lisbon. I am proud to have been successful in two different places and to have concluded my academic studies with this interesting topic.

2. Main issues and perspectives

The *SafeHouse* system has four main limitations that can be considered as future development:

1. It has been used Bluetooth as medium of communication between the Home Server and the embedded devices. This protocol stack has been proved to be enough fast, but it has been proved to have problems of wall penetration. This is critical if a user wants to implement the system in his/her house. A possible solution to tap this problem would be to use Bluetooth hops: devices in strategic points (such as doors) that redirect the traffic till the end point. Other possible solution would be to use RFID as medium of communication. This would allow a bigger range and it would tap the problem of wall penetration.
2. The embedded devices used are limited in terms of memory. This would be critical in a bigger system with dozens of components connected to the same device, where a large XML structure needs to be stored at some point. Two solutions can be considered for this fact: expand the memory of the device through adding a new RAM; or shorting the XML that is being managed at each moment in the embedded device. Definitely, the first solution would tap better this issue.
3. The GUI on the mobile application is limited to some screen components and absented of 2D or 3D graphics. This would be a positive point to have an elegant GUI and definitely would attract more users to use the *SafeHouse* system. Even though, the main goal of the mobile application was achieved.
4. The embedded devices are battery driven, what means that if someone plugs them out from the socket they stop working immediately. The best solution to tap this problem would be to have power generators or an UPS. This would give extra credit to such home automation system.

There is also the limitation of using Bluetooth as medium of communication with the Mobile Application, but since the user can use 3G either indoors or outdoors, this is not considered as a main limitation.

Regarding the four limitations exposed, let us remember that this system was made as a proof of concept and a commercial implementation of such system with more available resources than the ones that I had during this semester would tap or find other solutions with other technologies. Besides, the system is consistent and stable, what is undoubtedly the strongest point of the *SafeHouse* project.

VII. Appendixes

Appendix A

As we have stated in section 3.1 of Analysis chapter (page 28), we have went through some solutions to solve the problem of linearization with the distance sensor Sharp GP2D12. The first approach we took in account is represented in the following picture. We have used a 1st grade linear regression in the output signal. With this the signal becomes linear, but pruned to huge deviations as we can see. Only around 1V and 3,75V the distance respects the real value.

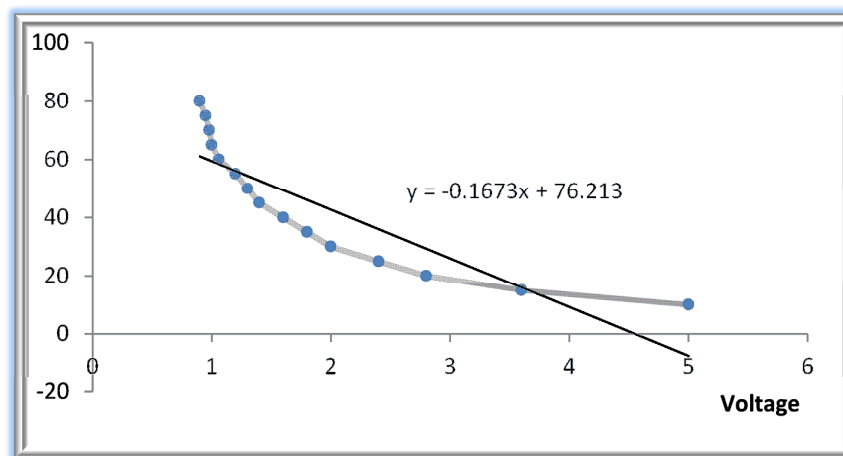


Figure 71 - Sharp GP2D12 non-linear and linear graph.

As referred in that section, we could have used a 2nd grade regression and with this the problem will become solved, or at least, the output would be really approximated to the real distance value. However, for a microcontroller as the pic30f3013 this would have become a heavy solution for a light CPU.

Thus, we had two options: either we store the values on the EPPROM of the microcontroller or we make sub-linearization. The first solution lacks of completeness: it is almost impracticable to make a table with every single input value. With this, we have divided our output into three different intervals and applied a 1st linear regression (as stated in that section). With this we achieve quite approximate values, as we can see in the following table.

Without having the necessity of being 100% precise, we have decided to use this last option.

Volts (x100)	Distance (in cm)	Linear Distance (in cm)	Absolute Error (in cm)
90	80	75,482	4,518
95	75	72,216	2,784
98	70	70,2564	0,2564
100	65	68,95	3,95
106	60	65,0308	5,0308
120	55	55,886	0,886
130	50	49,354	0,646
140	45	43,919	1,081
160	40	39,865	0,135
180	35	35,811	0,811
200	30	31,757	1,757
240	25	23,222	1,778
280	20	21,046	1,046
360	15	16,694	1,694
500	10	9,078	0,922
			1,81968

Figure 72 - Comparison between different output data in the Sharp GP2D12. The green cell is the average of the absolute error.

Appendix B

In section 4 of Analysis (page 32) was referred that we have used libraries developed previously during Sérgio Pedro's internship at PDM Technology (Pedro, 2010). In this appendix we are going through which libraries we are using: their intent and why we have used in this project.

1. XML Library

The main propose of this library is to make the XML easy to be parsed and easy to transform into an object (XMLObject). Then, the object can be acceded in a simple way and the developer can perform different actions over it: create, update and delete. Let us say that we receive a string representing the XML called **received**. Thus, we can parse that string by simply doing:

```
Parser parser = new Parser(received);  
XMLObject xmlObject = parser.getXMLObject();
```

Of course, the string has to respect the XML format, i.e. the XML has to be valid otherwise an InvalidXMLException will be thrown. At this point, we have our XML object loaded. Let us say that we receive another string, we parse it into a new XML object called **xmlObjectUpdate** and now we want to update our XML object in memory. To do that we simply make:

```
XMLNode rootNode = xmlObject.getRoot(); //to get the root of the XML  
XMLNode rootNodeUpdate = xmlObjectUpdate.getRoot();  
rootNode.updateNode(rootNodeUpdate);
```

Other actions can be taken like creation or deletion of elements in the xml. Since the *SafeHouse* system uses XML as format of inter connection between components, we have decided to integrate this library into our project.

2. BSCOM Library

Different frameworks implement in different ways the communication methods. Some frameworks, like J2ME make a quite standard way to use connectivity, but others not that much, like Android SDK. Following-up the idea of unifying different communication standards, the intent of this library was to make a flexible and easy way to create either a client or a server. With this, the developer can use either Internet (3G/Wi-Fi) or Bluetooth without taking care of implementation details. In the following code we explain how to open different kind of

connections. For instance, for a client to open a connection and send/receive data the procedure is really simple:

```
BCom bsCom = new BCom(); //object containing the main methods
BConnection conn = bsCom.Connect(ip,port,name); //opens connection
conn.WriteData("Hello");
String received = conn.ReadData();
```

To use Internet or Bluetooth the procedure is simple:

- If the developer wants to use Internet, it has to give the ip and port of the server and leave name null.
- If the developer wants to use Bluetooth, it has to give the mac address (ip), port and the name of service.

For a server the procedure is similar, except that instead of using Connect, it has to use the method Accept.

Since we have used during this project Bluetooth, 3G and Wi-Fi, it made all the sense to incorporate this library into our application.

Appendix C

In the chapter 3.2.1 it was referred the mechanism of encryption used in the *SafeHouse* system. Indeed, there are two steps to be described: the encryption and the decryption. The Vigenere algorithm is a symmetric cypher, meaning that the decryption is the symmetric operation of the encryption.

The message can contain any ASCII character, except for the 32 first ones, that in general are special and not used by means of information. Thus, there are 96 characters remaining of the ASCII table to be encrypted. In first place, both sides of the communication channel have to agree in a key to encrypt and decrypt the information. In order to be easily understandable, each character is an integer from 0 to 95, both for the key or for the message. In the following equations is the representation of message, key and encrypted message:

$$M (\text{message}) = \{m_0, m_1, \dots, m_N\}$$

$$K (\text{key}) = \{k_0, k_1, \dots, k_M\}$$

$$C (\text{encrypted message}) = \{c_0, c_1, \dots, c_N\}$$

The first step of the encryption and the decryption process is to assure that the key length is the same as the message length. Thus, if $N \leq M$, the key is going to be the first N characters of the original key. If $N > M$, the key is going to be repeated till it reaches the length of the message. In either of the cases the key is length N in the end. After this step, it is produced the encrypted message:

$$C_i = (M_i + K_i) \text{ mod } 96, \forall i \in (0, N)$$

Every character of the message is shifted according to the character of the key and applied modulo 96, to ensure that every value of the cypher message is between 0 and 95. The symmetric process is repeated in the decryption process:

$$M_i = (C_i - K_i) \text{ mod } 96, \forall i \in (0, N)$$

As an example, let us consider that we want to encrypt the following the message "Hello I am Sergio and I study in AAU!" with the key "safehouseSystem". The message has 29 characters, what means that the key will be expanded to "safehouseSystemsafefhouseSyste". Applying the formula above, the encrypted message will be:

- " ;FRQWo>sF@yFYWT\PfFVSu<eFmhX^m\Of&)Dv"

Using the same key the decryption is made in the symmetric way and in the other side finally the message can be read and interpreted.

Appendix D

1. Room/Component Structure

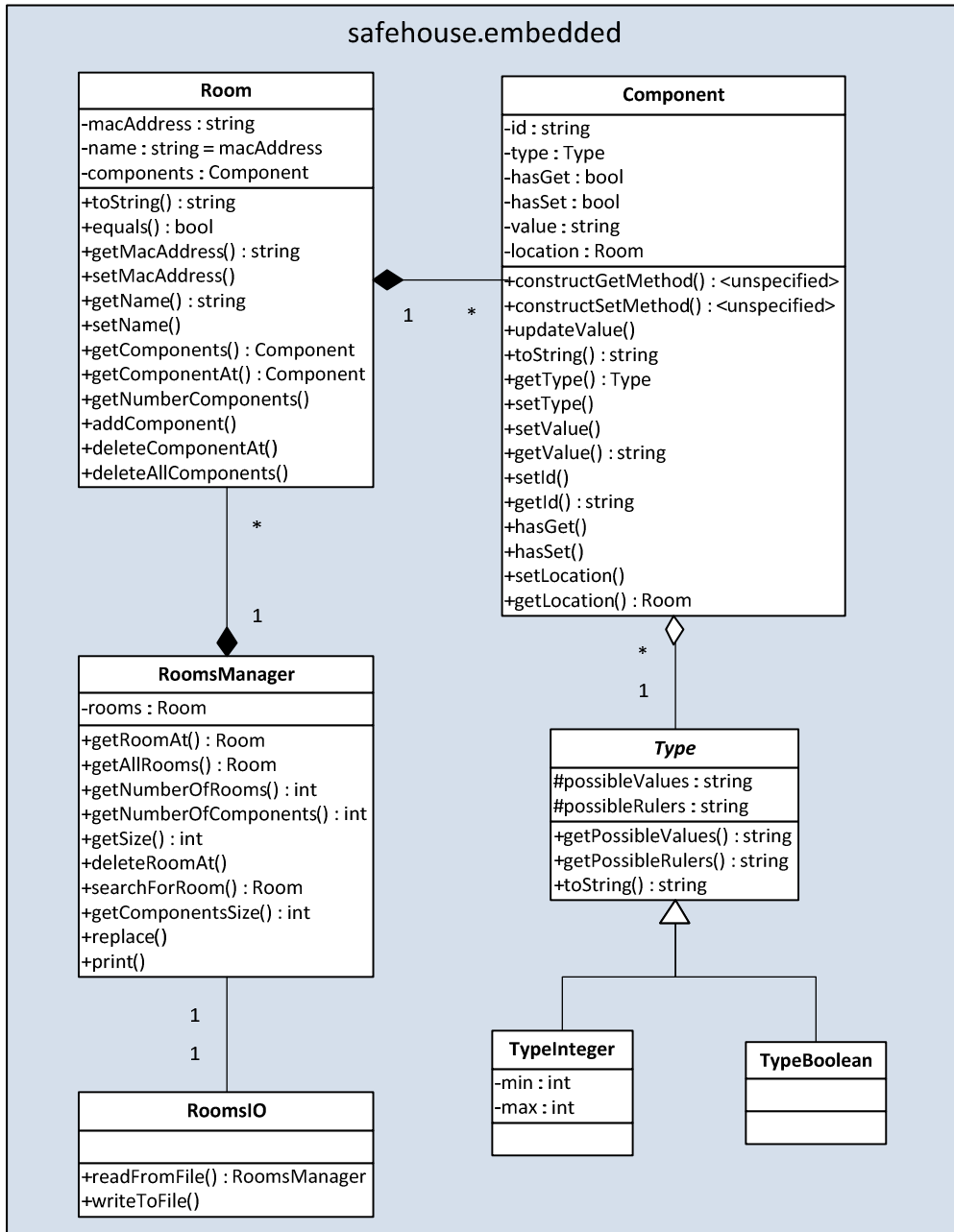


Figure 73 - Class Diagram for the Rooms/Components Structure.

2. Users Structure

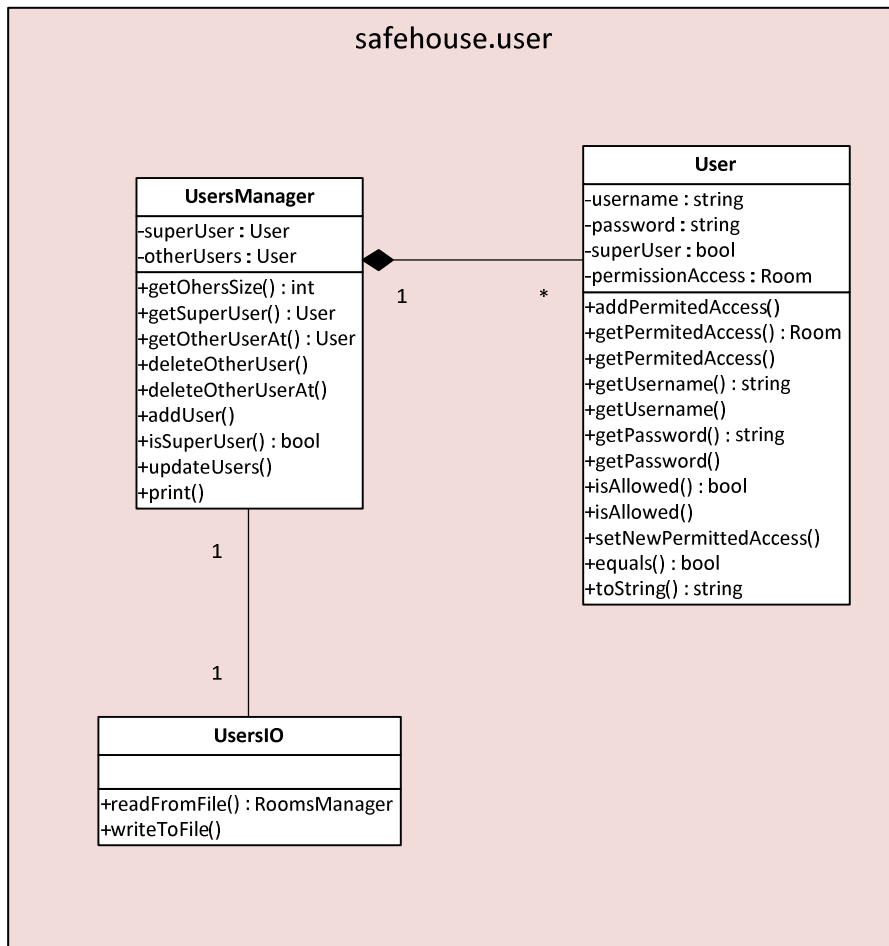


Figure 74 - Class Diagram for the Users structure.

3. Notifications Structure

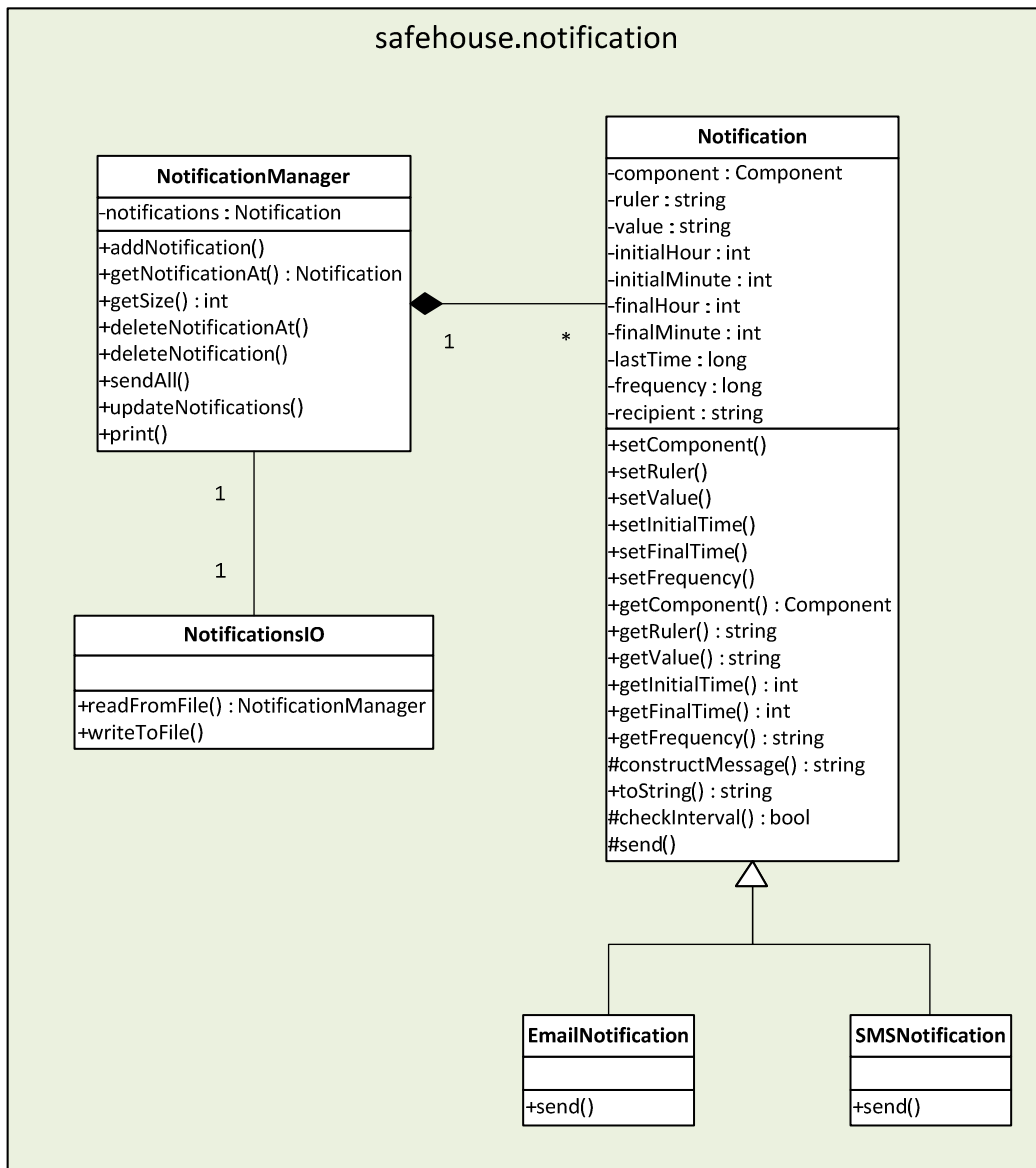


Figure 75 - Class Diagram for the Notifications Structure.

4. Rules Structure

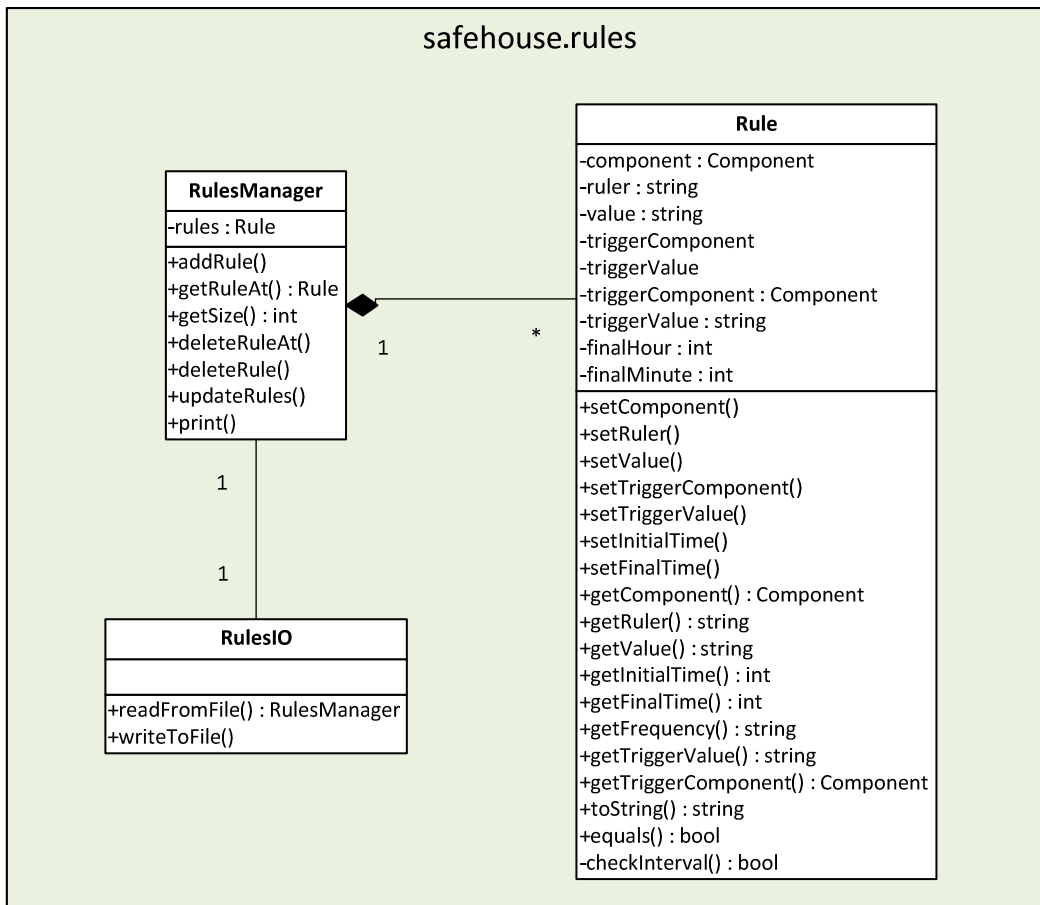


Figure 76 - Class Diagram for the Rules Structure.

Appendix E

1. Message Class

Message
-rightMessage : bool
-message : string
+isRight() : bool
+getMessage() : string

Figure 77 - Message class. The attribute `rightMessage` indicates if the operation was succesful or not; the attribute `message` is used to display some error information or new information to be put on the GUI.

2. GUIInterface Class

```
public static Message authenticateUser(String username, String password){ }
```

This method is used when the Desktop Application (DA) or the Mobile Application (MA) needs to check if the username and password inserted are valid, i.e. if the user exists in the system or not.

```
public static Message createRule(String id, String ruler, String value,
String triggerId, String triggerValue, String[] initialTime, String[]
finalTime){}
```

This method is used when the DA, after user input on the Create Rule screen, wants to create a new rule in the system. For that, all the parameters are passed to the method in order to create this rule. A Message object is returned indicating if the operation was successful and containing a message to be used on the DA.

```
public static Message deleteRule(String rule){}
```

This method deletes a rule according to the rule passed as argument to this method. The rule is going to be searched on the system and if it exists, it proceeds to the deletion of that rule.

```
public static Message createNotification(String componentId, String ruler,
String value, String type, String recipient, String[] initialTime, String[]
finalTime, String frequency){}
```

This method has the same logic as in the *createRule* method.

```
public static Message deleteNotification(String notification){}
```

This method has the same logic as in the *deleteRule* method.

```
public static Message createUser(String username, String password, String
repeatPassword, String[] permissions){}
```

This method has the same logic as in the *createRule* and *createNotification* methods.

```
public static Message deleteUser(String username){}
```

This method has the same logic as in the *deleteRule* and *deleteNotification* methods.

```
public static Message setNewRoomNames(String[] roomNames){}
```

This method is used when the user has customized the names of the rooms. Thus, it goes through all the rooms in the system and modifies the names of all of them. A Message object is returned in the end.

```
public static String[] getLogs(){}
```

This method is used in the main menu to get the logs of the system.

```
public static String[] getUsernames(){}
```

This method is used in the User screen in order to display the usernames of all the users in the system.

```
public static String[] getRules(){}
```

This method is used in the Rule screen in order to display all the rules in the system in a string representation.

```
public static String[] getNotifications(){}
```

This method is used in the Notification screen in order to display all the notifications in the system in a string representation.

```
public static String[] getComponents(){}
```

This method is used in the Create Rule and Create Notification screens in order to obtain the possible components for the condition in the rule and in the notification, respectively.

```
public static String[] getTriggerComponents(){}
```

This method is used in the Create Rule screen to obtain the possible components where is going to be triggered an action. Only components with the **set** method active can be returned by this method.

```
public static String[] getRulers(String component){}
```

This method is used in the Create Rule and Create Notification screens in order to obtain the possible rulers (is, is more than, is less than) for the condition in the rule and in the notification, respectively.

```
public static String[] getValues(String component){}
```

This method is used in the Create Rule and Create Notification screens in order to obtain the possible values for the component given as argument in the rule and in the notification, respectively.

```
public static String[] getPossibleTypesOfNotification(){}
```

This method is used in the Create Notification screen to get which types of notification can be used to create the notification. In this master thesis were considered two: e-mail and SMS.

```
public static String[] getPossibleHours(){}
```

This method is used to get the possible hours to be used in the Create Rule and Create Notification screens in order to set the initial or final hour of the interval. Typically, it returns a list containing values between 01 and 24.

```
public static String[] getPossibleMinutes(){}
```

This method is used to get the possible minutes to be used in the Create Rule and Create Notification screens in order to set the initial or final minute of the interval. Typically, it returns a list containing values between 00 and 59.

```
public static String[] getRooms(){}
```

This method is used in the Customization screen to get the current names of the rooms. It is also used in the Create User screen to set up the permissions for the new user.

VIII. Bibliography

- Acroname, I. (2010). *Sharp IR Rangers Information*. Retrieved February 2010, from Acroname Robotics: <http://www.acroname.com/robotics/info/articles/sharp/sharp.html>
- Adão, H., Antunes, R., & Grilo, F. (2008). Web-Based Control & Notification for Home Automation Alarm Systems. *World Academy of Science, Engineering and Technology*(37), 152-156.
- Alkar, A. Z., & Buhur, U. (2005). An Internet Based Wireless Home Automation. *IEEE Consumer Electronics*, 51(4), 1169-1175.
- Bieberstein, N. (. (2008). *Executing SOA: a practical guide for the service-oriented architecture*. Upper Saddle River, N.J.: IBM Press / Pearson.
- Burroughs, J. (2010). *X-10® Home Automation Using the PIC16F877A*. Microchip Technology Inc.
- Corporation, N. S. (2010). *LM35 - Precision Centigrade Temperature Sensor*. Retrieved February 2010, from National Semiconductor: <http://www.national.com/mpf/LM/LM35.html#Overview>
- Department of Electronic Systems, A. U. (2009). *Mobile Devices: opensensor*. Retrieved February 2010, from <http://mobiledevices.kom.aau.dk/projects/00/>
- Forum Nokia. (2010). *Getting Started with the Nokia Qt SDK*. Nokia Corporation.
- Group, J. E. (2002). *Mobile Information Device Profile for Java 2 Micro Edition*. Sun Microsystems Inc. and Motorola Inc.
- Huhns, M. P., & Singh, M. N. (2005, January). Service-oriented computing: key concepts and principles. *IEEE Internet Comput.*, 75-81.
- Indeed. (2010, May). *Bluetooth, Zigbee Job Trends*. Retrieved May 2010, from Indeed - one search. all jobs.: <http://www.indeed.com/jobtrends?q=Bluetooth,+Zigbee&l=>
- Indeed. (2010). *j2me, pys60, Symbian C++, Qt Job Trends | Indeed.com*. Retrieved March 2010, from Indeed.com: view-source:<http://www.indeed.com/jobtrends?q=j2me,+pys60,+Symbian+C%2B%2B,+Qt&l=>
- Intel Research. (2006, December 18). *Bluecove*. Retrieved March 2010, from Sourceforge: <http://sourceforge.net/projects/bluecove/>
- Josuttis, N. M. (2007). *SOA in practice*. Beijing: O'Reilly.
- Khiyal, M. (. (2009). SMS Based Wireless Home Appliance Control System. *Issues in Informing Science and Information Technology*, 6, 887-894.

- Klingsheim, A. N. (2004). *J2ME Bluetooth Programming*. Bergen: University of Bergen.
- McLean, P. (2009, November). *AppleInsider | Canals Q3 2009: iPhone, RIM taking over smartphone market*. Retrieved February 2010, from AppleInsider:
http://www.appleinsider.com/articles/09/11/03/canals_q3_2009_iphone_rim_taking_over_smartphone_market.html
- Meyer, S., & Schulze, E. (2006). Smart Homes and the Aging User, Trends and Analysis of Consumer Behavior. *Symposium Domotics and Networking*. Miami: Berlin Institute for Social Research.
- Nokia Corporation. (2008). *PyS60 Library Reference*. Nokia Corporation.
- Nokia Corporation. (2008, October). S60 5th Edition: What's New for Developers. Nokia.
- Osipov, M. (2008). Home Automation with ZigBee. *The 8th International Conference on Next Generation Teletraffic and Wired/Wireless Advanced Networking (NEW2AN)* (pp. 263-270). St. Petersburg: Springer-Verlag.
- Papazoglou, M. P. (2003). Service-Oriented Computing: Concepts, Characteristics and Directions. *Fourth International Conference on Web Information Systems Engineering* (p. 3). Rome: IEEE Computer Society.
- Pedro, S. (2010). *Orion Mobile*. Aalborg: Aalborg University/PDM Technology.
- Pérez, F. (. (2007). Symbian C++ Application Programming Overview. ITACA Institute, Polytechnic University of Valencia.
- Pering, T. (. (2006). CoolSpots: Reducing the Power Consumption of Wireless Mobile Devices with Multiple Radio Interfaces. *4th Int. Conf. Mobile Systems, Applications and Services (MobiSys)*, (pp. 220-232).
- Regnier, T. (n.d.). *MIDP telephones benchmark*. Retrieved March 2010, from Le Club des utilisateurs Java: http://www.club-java.com/TastePhone/J2ME/MIDP_Benchmark.jsp
- Rijmenants, D. (2010). Retrieved April 2010, from Cipher Machines and Cryptology:
<http://users.telenet.be/d.rijmenants/pics/vigenere.JPG>
- Schmidt, M. -T. (2005). The Enterprise Service Bus: Making service-oriented architecture real. *IBM Systems Journal*, 44(4), 781–798.
- Stinson, D. R. (2006). *Cryptography*. Boca Raton: Chapman & Hall/CRC.
- Tanenbaum, A. S. (2003). *Computer Networks*. New Jersey: Pearson Education, Inc.
- Warmer, C. (. (2009). Web services for integration of smart houses in the smart grid. *Grid-Interop*, (pp. 207-214). Denver.
- Weis, S. A. (2003). *Security and Privacy in Radio-Frequency Identification Devices*. Massachusetts: Massachusetts Institute of Technology.

Xuguang, H. (2009, November). An Introduction to Android. Database Lab. Inha University.