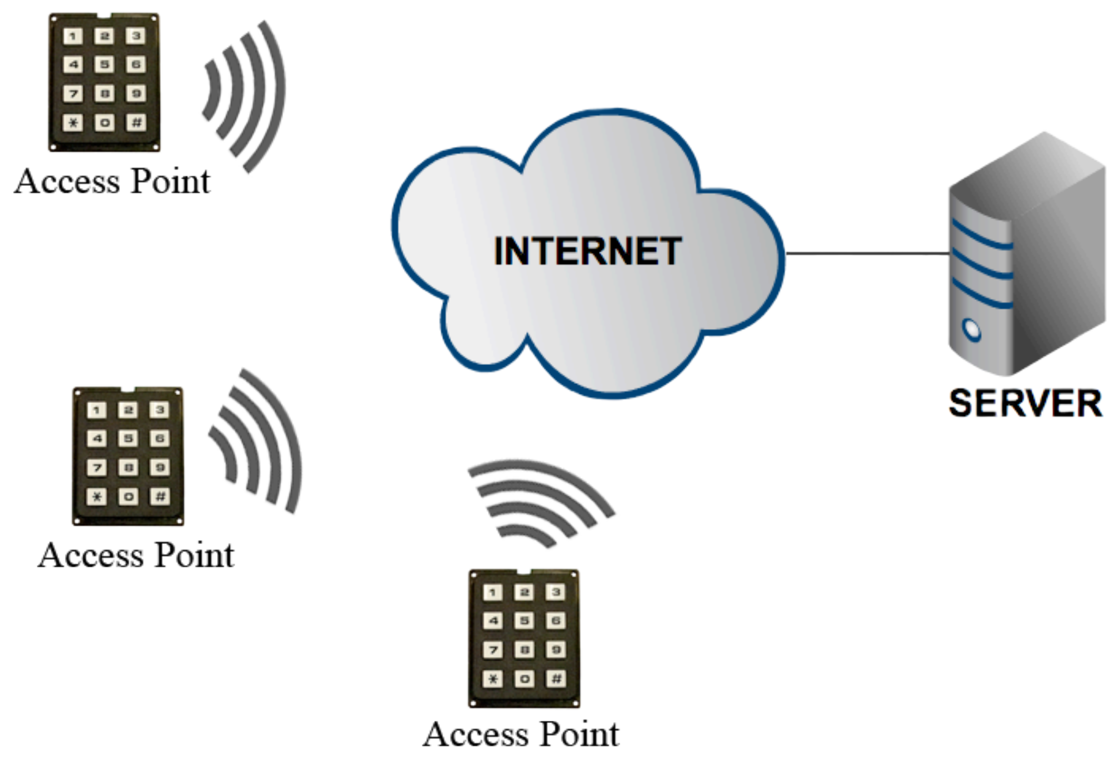
CHALMERS



Development of a Centralized Electronic Lock System Based on 3G Broadband Modules

Master of Science Thesis in Networks and Distributed Systems
Master of Science Thesis in Secure and Dependable Computer Systems

ABDI ABATE
PATRIK RUTGERSSON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, March 2012

Development of a Centralized Electronic Lock System Based on 3G Broadband Modules

ABDI ABATE
PATRIK RUTGERSSON

Examiner: TOMAS OLOVSSON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover: An overview of the developed system.

Department of Computer Science and Engineering
Göteborg, Sweden March 2012

Abstract

Ericsson's 3G Broadband modules are primarly intended for laptops, but they are perfect in terms of size for being used in other electronic devices. Therefore, the purpose of this master's thesis is to develop a demo system for Ericsson's 3G Broadband modules. We have developed an electronic lock system consisting of a large number of access points talking to a centralized server. The server handles the access control and is contacted every time a user wants to get access to an electronic lock. The electronic locks, called access points, need 3G modules to be able to communicate with the server over the Internet. The communication within the system is encrypted with SSH and pin codes are used to identify different users. The developed system became a well-functional electronic lock system.

Acknowledgements

We would like to thank the employees at Ericsson for the help they have given us. A special thank to our supervisors, Erika Asp and Thorbjörn Larsson, for all feedback and suggestions throughout the project. We would also like to thank our examiner at Chalmers University of Technology, Tomas Olovsson, for his valuable feedback on the report.

Table of Contents

List of Figures

Chapter 1

Introduction

In 1992 the first mobile network, called GSM, which transmitted the voice digitally was introduced. Nine years later, year 2001, a new type of mobile network was introduced, which would be called third generation mobile telecommunications or 3G. One major difference between these two was that GSM focused on voice transmission, while 3G also had a requirement to provide other digital services [1]. After the introduction of 3G, the development of the bandwidth and coverage have continued and been improved. For instance, Telia's 3G mobile network covers 99% of the Swedish population with a maximum downlink speed of 10-32 Mbit/s [2, 3]. With such a development of the mobile networks, it opens up for new products except for just mobile phones. Nowadays 3G Broadband modules exist, with the main purpose of providing Internet connections through the mobile network to laptops. But in terms of size, those modules are perfect for being used in other electronic devices where an Internet connection is necessary. An example of such an area is monitoring or steering devices.

An advantage with using 3G is that an Internet connection is most often available regardless of where the device is located, as a result of the mobile network's good coverage. In many cases where a cable has been a must before, it can now be replaced with a communication link over 3G. Such an example is an electronic lock that is controlled over the Internet.

1.1 Related Work

The use of mobile communication no longer focus on just the mobile phone market, but is nowadays used in other areas as well. For instance, 3G is built-in into one of Apple's iPad versions [4]. There are many possibilities and in [5], the authors have written about how they have developed a system that utilizes 3G to send patients' physiological parameters, like blood pressure and ECG, to a doctor's work station. In that way the doctor's efficiency is improved and at the same time the parameters of the patients are monitored such that they can get treatment in time if it is needed.

Using mobile communication for monitoring is fairly common and Cai et al. [6] have developed a system to supervise sensors over the mobile network from a centralized place.

An electronic lock system is something that preferably is centrally controlled. Daradimos et al. [7] have developed a system where electronic locks, called access points, talk to a central server over an already existing wired network.

However, something that does not seems to exist is a centralized electronic lock system that utilizes 3G to communicate. Wireless solutions exists, but they are based on other wireless technologies. For instance, the communication between the devices in the system developed by Hwang et al. [9] is based on ZigBee. But a limitation of ZigBee is that the distance between two adjacent devices cannot be longer than 100 meters [10]. The lack of an electronic lock system based on 3G makes this master's thesis relevant.

1.2 Problem Description

The purpose of the master's thesis is to develop a demo system for Ericsson's 3G Broadband Modules. The system is an electronic lock system; where remote access points communicate with a centralized server asking for unlock permissions. The requirements on the system are the following:

- The system must be scalable and be able to handle a large number of access points.
- The system needs to handle all access points in a smart way and to prevent them from having an established connection to the server at all times.
- The server will keep track of the configuration of each access point.
- New firmware updates are regularly released to the modules. These cannot be manually installed on each module, but needs to be remotely distributed.
- Each user that will use the system needs to be recorded in some way and given proper access rights.
- The user management of the system must be implemented such that it is easy to maintain when a lot of users use the system.
- All data that will be sent within the system needs to be encrypted to prevent it from being eavesdropped.

1.3 Limitations

The scope of this thesis is to develop a lock system that uses 3G Broadband modules to communicate with a centralized server. We mainly focus on the software and not as much on the hardware. Therefore, the access point will be based on a development board to allocate more time to the software implementation.

What is special about this system is that it communicates with the server over 3G, and instead of controlling a real lock from the access point, two LEDs will be used to illustrate whether it is locked or unlocked.

A keypad will be connected to the access point and used to get user input and the users will interact with the access point by typing their user ids and PIN codes. Already existing lock systems use for instance RFID tags or magnetic cards to avoid people to get access by just knowing the code. Since we mainly focus on the software, no extra security like that will be added to our hardware. But right from the moment when the user credentials are entered, they will be handled with respect to security.

The server application will be written to handle many access points. But due to hardware limitations, only one single access point was developed.

1.4 Outline of the thesis

In chapter 2, an overview of the implemented system is given. Chapter 3, gives a description of hardware and software related technologies used in this master's thesis. Chapter 4 discusses the techical choices that were made during the project. A hardware description of the access point is found in chapter 5. In chapter 6, the configuration of development board is explained and chapter 7 describes the implementation of the software. In chapter 8, 9 and 10 the summary, discussion and conclusions and future work are discussed.

Chapter 2

System Overview

This section gives an overview of how the different parts in the system work together. The lock system consists of a server and a large number of access points, which is illustrated in figure 2.1. Each access point talks to the server over the Internet to exchange information.

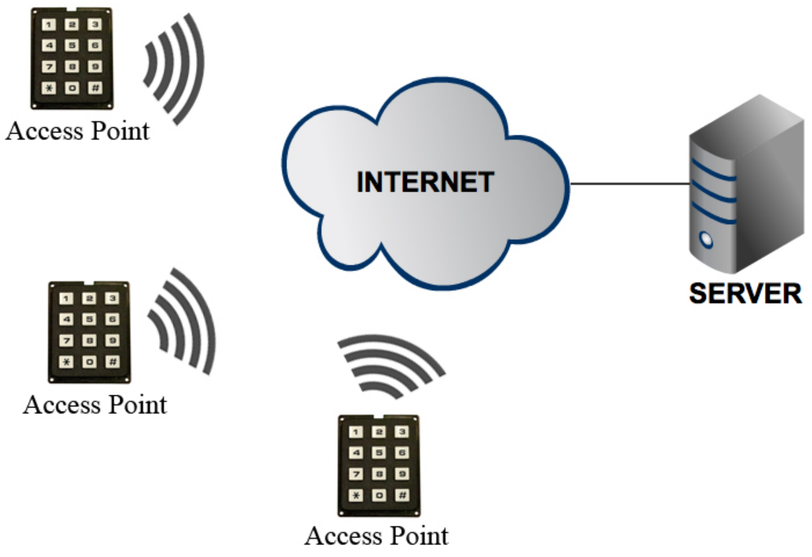


Figure 2.1: An overview of the system.

2.1 Access Point

The implemented access point (AP) is a device that illustrates the functionality of an electronic lock. An access point is needed at each location, e.g. a door that will be included in and controlled by the lock system. The overall functionality of the access point is to collect ids and pin codes from users, ask the server if that user has access to that specific location and if so let the user in.

The AP, illustrated in figure 2.2, is based on a development board called BeagleBoard. The reasons why that board is used are explained in section 4.1. The access point is equipped with a 3G Broadband module to be able to connect to the Internet and talk to the lock system server. The module is connected to the access point via one of the BeagleBoard's USB ports. In addition to the module, an external keypad is used to get input from the users. In section 5.2, an explanation of the keypad construction is given.

The functionality of the access point is implemented in a program written in C. The program handles everything from establishing an Internet connection to recognize individual key presses on the keypad. The implementation of this program is explained in section 7.2.

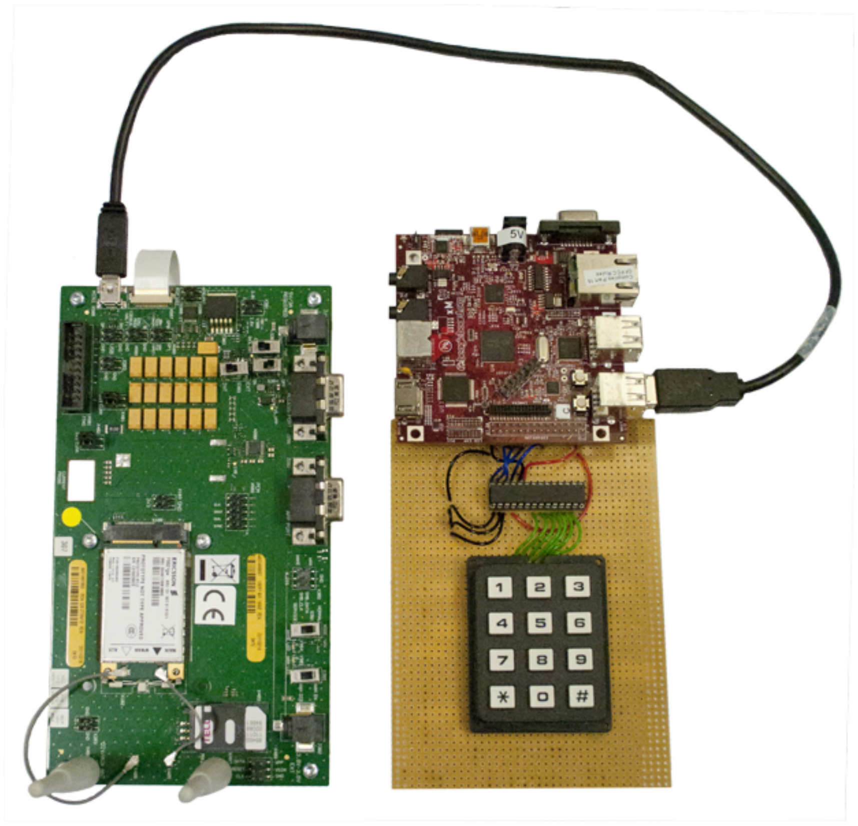


Figure 2.2: The access point that was developed.

2.2 Server

In the system a centralized server is used to handle the connections from all access points. The server controls the accesses of the users and based on their permissions, it accepts or rejects the unlock requests that are sent by the access points. For platform independency, this server application is written in Java and equipped with a graphical user interface to easily manage the system. Each individual access point is managed from the server and its configuration is stored on the server.

Since new firmware is released to the 3G Broadband modules, the system remotely handles these updates. When an update is released, it is manually added and controlled from the graphical user interface. If the server discovers that an access point needs a firmware update, that access point is informed about it and at what time the update will run. At that time, the file including the update is distributed from the server.

In section 7.3, the implementation of the server application is explained.

Chapter 3

Technical Description

This chapter gives a description of different technologies that have been used in this project. Sections from this chapter is recommended to be read when they are referenced from other chapters in this report.

3.1 Hardware Related

3.1.1 Matrix Keypad

Matrix keypads are used in systems for human interaction. The architecture of a matrix keypad is very simple and it is easy to use together with a microcontroller. The buttons are arranged in a matrix, which can be seen in the figure below. The only components are the buttons that are connected together in rows and columns [11].

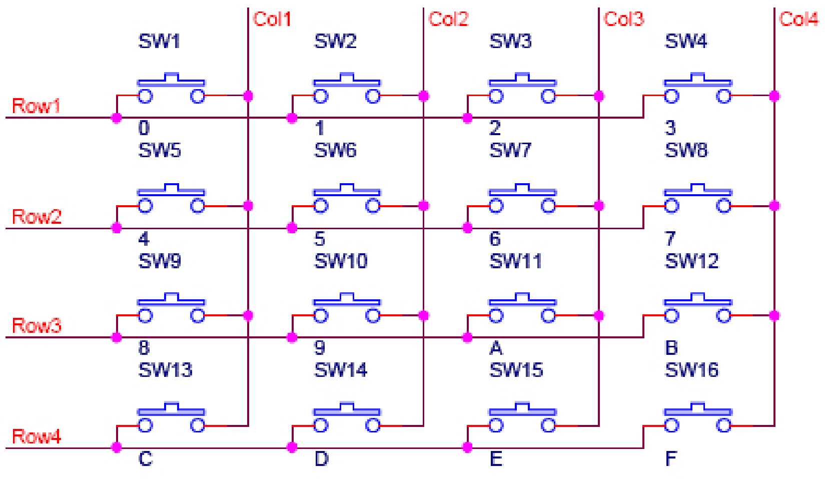


Figure 3.1: A schematic of a keypad. (Figure from [11])

Since the keypad only contains buttons, which are simple switches, the direction of the signals does not matter. Either the columns can be connected as output and the rows as input to the I/O signals on the processor or vice versa. Suppose rows are inputs and columns are outputs: when a button is pressed, the column pin of the button has the same signal value, thus '0' or '1', as its row pin. For instance, if SW1 in figure 3.1 is pressed, then Col1 has the same signal value as Row1. The value of a column pin is undefined if no button is pressed within that column. Therefore pull-up (or pull-down) resistors are usually used on the output pins to get a value in those situations.

3.1.2 Serial Peripheral Interface (SPI)

SPI is an interface for serial communication defined by Motorola. It is a 4-wire synchronous interface that is used for connecting external devices [12]. The signals that are used to communicate with a device are:

- SCLK - Serial Clock
- MISO - Master-In Slave-Out
- MOSI - Master-Out Slave-In
- SS - Slave Select

The devices that are communicating are referred to as "master" and "slave". The master controls the speed of the communication by driving the serial clock [12]. It uses data signals in both directions, which means that it is a full-duplex protocol. Several slave devices can share the same SCLK, MISO and MOSI signals by using an individual slave select signal to all external devices, see figure 3.2. The "master" chooses the device to talk to by enabling its slave select signal.

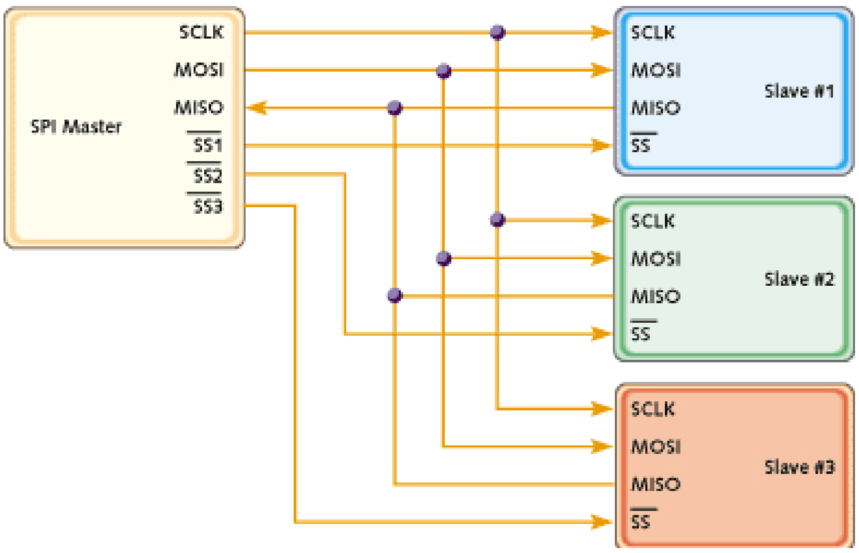


Figure 3.2: One master is talking to several slave devices over SPI. (Figure from [12])

3.2 Software Related

3.2.1 AT Commands

The AT commands were first used in modems developed by a company called Hayes in 1977. Earlier modems had used two different channels: one for sending and receiving data and another channel for sending control commands to the modem. However, this modem only used one channel for both data and control commands. To distinguish between commands, each command sent to the control channel started with the characters AT, which stands for attention. Because of these two characters, this set of commands became known as AT commands. Other modem manufacturers started to use this set of commands because of simplicity [13].

The syntax of each AT command is made up of a prefix, body and termination character [14]. The prefix can either be the characters "AT" or "at" or to repeat the command that was previously entered the characters "A/" or "a/". The body of the command consists of the command, parameters and values if it is applicable. The different syntaxes of commands can be seen in listing 3.1.

```
Basic syntax
AT<command>[=][<parameter>]<CR>

Extended syntax
AT+<command>=[<parameter>]<CR>
AT*<command>=[<parameter>]<CR>

Syntax for reading parameter values
AT<command>?<CR>
AT*<command>?<CR>
AT+<command>?<CR>
```

Listing 3.1: The syntax of the AT commands [14].

Here are examples of two different AT commands:

- AT+CPIN? - Ask modem if SIM is PIN code protected.
- AT+CPIN="1234" - Input pin code 1234 to the modem.

3.2.2 BuildRoot

BuildRoot is used to easily generate an embedded Linux system. BuildRoot is a set of Makefiles and patches that automates the building of cross-compiling toolchain, generation of filesystem and compilation of bootloader and kernel image [15]. Its focus is on small or embedded systems.

3.2.3 Cross-Compiling

In the compiling process, the terms host and target are used to differentiate between where the program is compiled and where the program will be executed, respectively. The compiler is called native if the host and the target are the same, but it is called a cross-compiler if the host is different from the target machine. Cross-compilation is needed when the target machine cannot run its own compiler, for example if it lacks resources like CPU or RAM capacity [16]. A cross-compiler is also needed to get Linux to run on a new hardware platform.

3.2.4 Model-View-Controller (MVC)

The design pattern Model-View-Controller is often used when graphical user interfaces are built. The design pattern consists of three different objects: model, view and controller. The model implements the functionality of the application, the view is the visual feedback and the controller takes care of the user input. By following MVC, the code is better structured, which increases flexibility and the ability to reuse the code [17]. Separating the functionality of the program from the user interaction makes it easier to create several different views to the same program or use the same view in another program.

3.2.5 NMEA 0183

The standard NMEA 0183 is a definition of a communication protocol between marine instrumentation. The data is transmitted in so-called sentences, which are sent between talkers and listeners [18]. The format of a sentence can be seen in listing 3.2 below. Each sentence starts with a '$' character followed by two characters representing the device's type; for instance "GP" for a GPS receiver. The next three characters specifies the sentence identifier. The sentence ends with a number of data fields separated by commas, a checksum and carriage return/line feed.

```
$ttsss ,d1 ,d2 ,.... <CR><LF>
```

Listing 3.2: The structure of an NMEA 0183 sentence.

3.2.6 Protocol Buffers

Protocol Buffers are developed and also used by Google to encode structured data. The encoding is done in a platform neutral way and can be compared with XML, but it is faster and simpler. The structure of the data is defined in ".proto" file as messages. Based on that file, a source code is generated to handle data structured in that way. At the moment, Protocol Buffers support three different languages: C++, Java and Python [19]. It can be used to store data or as a communication protocol.

Here is an example of defining a movie in Protocol Buffers. The movie needs a title and a genre, but the release year is optional and the number of actors is variable.

```
message Movie {
        required string title = 1;
        optional int32 release_year = 2;

        enum Genre {
                ACTION = 0;
                ADVENTURE = 1;
                COMEDY = 2;
                DRAMA = 3;
                HORROR = 4
                THRILLER = 5;
        }

        required Genre genres = 3;

        message Actor {
                required string name = 1;
        }

        repeated Actor actors = 4;
}
```

Listing 3.3: A representation of movie made in Protocol Buffers.

3.2.7 Role-Based Access Control (RBAC)

Role-Based Access Control became an alternative to Mandatory Access Control (MAC) and Discretionary Access Control (DAC), because of their inability of handing access control in larger organizations. In MAC, all users and objects are attached with a label or security classification. The accesses are based on those classifications and are granted if the security level of the object is equal or less than the level of user. In DAC, the accesses are based on the identity of the user or group that is accessing the object and the owner of an object can control its permissions. A difference between these and RBAC is that another level is added between the users and the permissions, which is called a role. A role is a collection of users, permissions and other inherited roles [29].

RBAC can be split into four different models: $RBAC_0$, $RBAC_1$, $RBAC_2$ and $RBAC_3$. $RBAC_0$ is the model that contains the minimum requirement for a RBAC system. $RBAC_1$ and $RBAC_2$ are two models with different extensions to the base model and $RBAC_3$ is a model containing both of those extensions. The base model contains four entities: users, roles, permissions and sessions. A user is a user of the system and for instance a human being. A role represents a job function within the company or organization and has the purpose of giving those permissions that members of that role should have. A permission is the same as granted access to an object. Sessions are used to control the permission level and to just activate those roles that are needed for a specific task. The second model, called RBAC$_1$, introduces role inheritance into the system, which allows creation of hierarchies of permissions. By letting roles inherit other roles, more powerful roles can be built upon less powerful roles. The next model, called RBAC$_2$, defines constraints [30]. By having constraints in the system, roles can for instance be specified to be mutually exclusive, which means that a user can only be assigned one of those roles.

An advantage with using RBAC compared to other access controls is that permissions that are assigned to a role tend to be more stable than permissions that are assigned directly to users [30]. Users' permissions need to be changed more often due to job function changes etc.

3.2.8 Sysfs

Sysfs is a filesystem and provides a way to manipulate kernel objects from user-space. Users and applications can easily access and control the kernel-space data structures, since the information is strictly organized and most often formatted in ASCII. In the Linux kernel, the kernel objects are represented as directories, object attributes as regular files and the relation between objects with symbolic links. Sysfs is mounted on the directory */sys*. Under this directory a number of subdirectories are created to represent the different subsystems that are registered with sysfs. These directories are block, bus, class, device, firmware, module and power. The *block directory* contains all devices that are block devices that have been discovered. The *bus directory* contains a directory for each different type of physical bus available in the system and supported by the kernel. A directory for every device class registered with the kernel is added to the *class directory*. In the *device directory*, a subdirectory is added for each device that has been discovered on any bus registered with the kernel. Platform-specific code, like x86 BIOS, can be manipulated or viewed through the interface in the *firmware directory*. All kernel modules are added as a directory into the *module directory*. The last directory, called *power*, is an interface for controlling states of processes and controlling the method that should be used by the system to suspend to disk [20].

Here is an example of how to control GPIO pins from sysfs. GPIO is a type of device class, so if GPIO is registered with the kernel, a subdirectory with the same name is located under the class directory (*/sys/class/gpio*). This directory contains subdirectories to represent GPIO chips and pins found on or connected to the hardware which runs the Linux kernel. A GPIO chip is a group of GPIO pins and each GPIO pin belongs to a chip. To control an individual pin, the pin needs to be exported from the chip using the file called export. By exporting a pin, a new subdirectory is created under GPIO, which contains files for controlling the value and direction of the pin. By writing "in" or "out" to the direction file, the pin is either set to input or output, respectively. The value file is read if the direction of the pin is set as input or "1" or "0" is written to control the output. In listing 3.4, an example of how the directories and files could be structured in the filesystem is shown.

```
/sys/class/gpio
        export
        gpio243
                device
                direction
                power
                subsystem
                uevent
                value
        gpiochip240
                base
                device
                label
                ngpio
                power
                subsystem
                uevent
        unexport
```
Listing 3.4: Gpio pin 243 is exported and controlled through sysfs.

Chapter 4

Technical Choices

4.1 Development Board

Right from the start, we decided that the access point will be based on a development board that runs a Linux kernel. The development board that we used is BeagleBoard-xM Rev C. Using a development board based on an ARM processor is preferable because it is cheap and well supported by the Linux kernel. The reason for using the BeagleBoard is first of all its support for SPI and USB. Secondly, its good specifications: 1 GHz processor and 512 MB of RAM. Thirdly, many others use it and good forums exist. Yet another advantage is that it has an Ethernet port, which is great to have during the development process. The type of the BeagleBoard used is shown in figure 4.1.

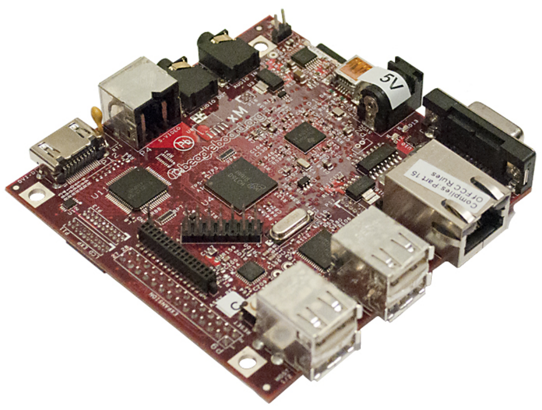


Figure 4.1: BeagleBoard-xM Rev. C.

4.2 Handling Users

The lock system that is developed is designed to handle a large number of access points and a lot of users. To properly assign access rights to the user, it is important that the management is made easy. Daradimos et al. [7] have created a system handling users and access points, but it does not support a way of grouping users and grouping access points and therefore it is not well suited for larger systems. However, their rules are based on weekly access rights and is well suited for our system. Weekly access rights mean that the rules are assigned what days and times users and access points should be active during a week; which are repeated on a weekly basis. Together with the weekly access rights, the lock system is based on Role-Based Access Control (RBAC); see section 3.2.7. Popa et al. [8] suggest an access control system based on smart cards which uses RBAC for access control. In RBAC, roles are created based on job functions or tasks performed within a company or organization. That is a general way and can therefore be adapted to basically all places where this lock system can be used.

4.3 Programming Languages

The program for the access point is written in C/C++. The combination of the languages is used because Protocol Buffers generate C++ code, but the rest of the code is written in C. By using C/C++, the development of the program for the access point could start on a Linux machine before the development board was received.

The server program is written in Java to be able to move the program between different operating system. From the beginning it was not known on what operating system the application would run and therefore using Java made the program platform independent. In addition, it is easy to create graphical user interfaces and we know how to write Java programs. Using a database together with a Java program is also trivial and Java is one of the languages that Protocol Buffers support.

4.4 Communication Between Server and Access Points

The data exchanged between the server and the access points is structured in Protocol Buffer messages; see section 3.2.6. Coming up with a new way of exchanging different types of data takes time, therefore it is better to use a protocol that is tested and used by others. Protocol Buffers is used because it is well documented, it is simple to use and it works with both Java and C++ [19]. An alternative can be Apache Thrift [21], originally developed by Facebook, which is similar to Google's Protocol Buffers. However, it is not well documented and has extra functionality that is not needed in this master's thesis.

There are also other data serialization formats like XML and JSON. Protocol Buffers however serializes structured data up to 100 times faster than XML [19] and neither XML nor JSON have elements of raw byte data.

4.5 SSH vs IPSec

To prevent PIN codes from being eavesdropped, all data sent within the system is encrypted. The encryption can either be embedded into the programs or existing third party implementations like SSH and IPsec can be used in combination with these programs. Implementing the encryption in the programs opens up for adding extra bugs and also that the encryption is not secure enough. Therefore, it is better to use existing SSH or IPsec implementations that are well established and tested. The advantage of using IPsec between the server and all access point is that it is a part of the operating system, which means that once it is configured all traffic will automatically be encrypted without considering it in the programs. However, the configuration of IPsec is harder than setting up an SSH tunnel. Since the access points are only talking to the server over one single port, it is enough to set up one tunnel from each access point to the server to get all data encrypted. Additionally, adding SSH to the Linux installation on the development board is easy; just enabling it in the compilation. Therefore, we decided that the lock system should use SSH tunnels to encrypt the traffic.

4.6 Keypad Construction

The keypad is constructed to get input from the user. It is connected to the BeagleBoard via a SPI to GPIO expander circuit. An alternative way is to use a USB keypad, which is easily supported by Linux. But a USB keypad cannot be used if the extension to this master thesis should be possible. The extension is to port the program written for the access point to the module and to let it execute directly on its processor. As a result, the BeagleBoard is not needed, but just the 3G module with a keypad connected to it. In that case, the keypad is preferably connected to the GPIO pins on the module. Unfortunately, the number of available GPIO pins on the module is limited and not enough to connect a keypad. An alternative is to use a SPI to GPIO expander to get more available GPIO pins. The SPI to GPIO expander circuit used is MCP23S17. It supports 1.8 Volts, which is the voltage of the pins of both the BeagleBoard and the 3G Broadband module, and there is also a driver supporting the circuit in Linux.

Chapter 5

Hardware

This chapter gives a description of the hardware of the access point. In section 1, the 3G Broadband module used is described. Section 2 gives a detailed description of how the keypad is connected to the development board.

5.1 Ericsson's 3G Broadband Module

A 3G Broadband Module is small 3G modem intended to be used inside a laptop computer. Its purpose is to provide Internet access to a computer through a mobile network. The module used in this project is the Ericsson's F5521gw module illustrated in figure 5.1. That model supports a bandwidth up to 21 Mbps downlink and 5.76 Mbps uplink. Along with the modem functionality, the module is also equipped with a network assisted GPS receiver. As seen in the figure, the form factor of the module is of type PCI Express Mini card. The f5521gw is supported by Windows 7 and Linux and has support for IPv6.

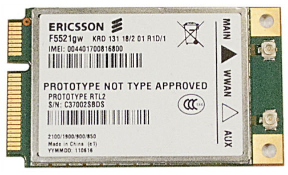


Figure 5.1: An Ericsson F5521gw module.

In the project a so-called cradle is used to connect the module to the host. The cradle has a Mini PCI Express slot to place the module in which can be connected to the host with just a USB cable. The cradle is equipped with a SIM card holder and antennas, which are necessary to use the module. A cradle is shown in figure 5.2.

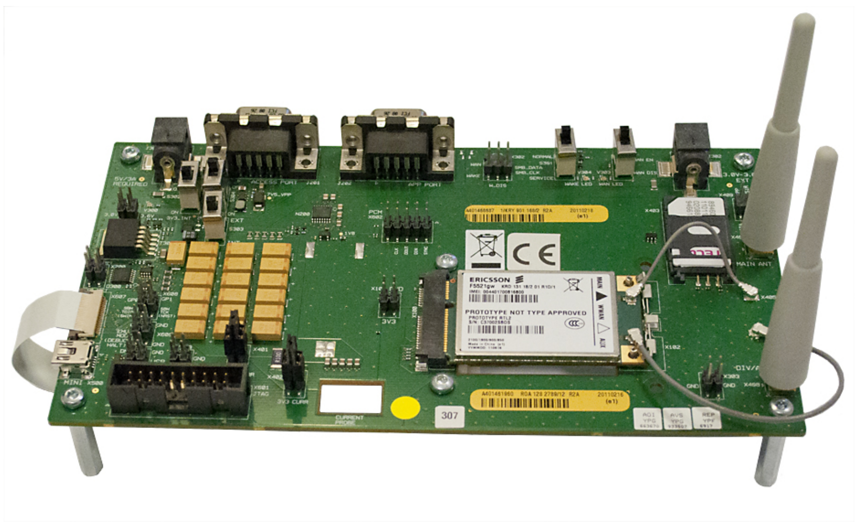


Figure 5.2: A cradle that the 3G Broadband module is placed in.

5.1.1 AT Commands

The module is managed by sending AT commands from the host. See section 3.2.1 for a more detailed description of AT commands. The AT commands are sent to the module to one of its virtual serial ports that are created when the module is connected. The modules handle a large set of different AT commands which can change all of its settings including to enable it when it is about to be used.

The AT commands used in this master's thesis are only a subset of those that are available. The AT commands that are needed are used to turn on the module, input PIN code to the SIM card, establish an Internet connection, read GPS coordinates etc. A complete list of all AT commands and their functionality is given in Appendix A.

5.1.2 Firmware Updates

Firmware updates for Ericsson's 3G Broadband modules are released to correct bugs and improve the functionality of the modules. The updates are developed by Ericsson, but are usually released together with updates from the laptop manufacturer that the module is used together with. The updates are installed by running the updater on the host, which the module is connected to. The installation of the update takes about 5 minutes and directly after its completion the module is ready to be used again.

5.2 Keypad Connection

A keypad is used to read input from the users and it is connected to the BeagleBoard's extension connector. A schematic diagram over the connected components is illustrated in figure 5.3.

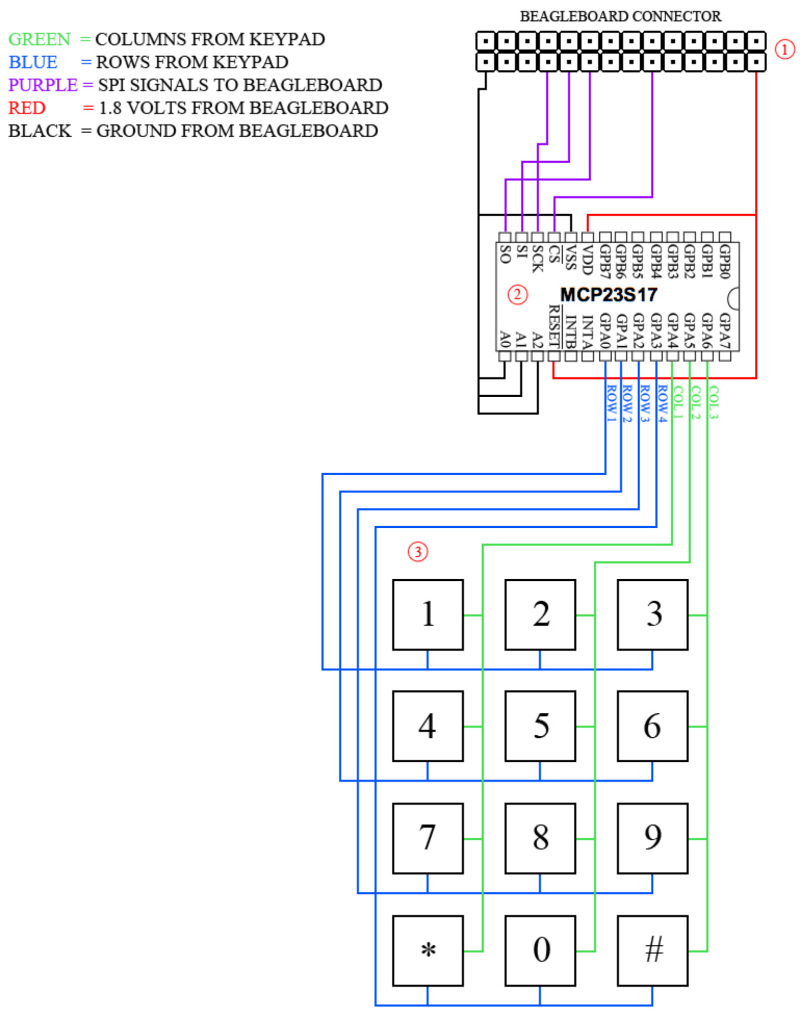


Figure 5.3: Connection of keypad to the BeagleBoard through a SPI to GPIO expander.

The component marked with '1' in the schematic is a connector that enables a plug in of the BeagleBoard's extension connector. The extension connector is a 28-pin connector used to externally connect peripherals to add more functionality to the board. In the connector, I/O pins and different communication interfaces, like SPI and I2C, are available; but also pins connected to ground, 1.8 and 5.0 volts. All pins that are connected to communication interfaces or I/O use 1.8 volts [22].

The circuit, called MCP23S17, marked with '2' connects the keypad to the BeagleBoard. It is a SPI to GPIO expander, which means that the BeagleBoard is equipped with 16 extra GPIO pins by just talking to the circuit through SPI. Several similar I/O expanders are available, but this circuit is used because it works with 1.8 volts and it is available in a DIP package. The pins on the I/O expander circuit marked with A0, A1 and A2 are address pins. Up to eight MCP23S17 circuits can be used on the same SPI bus, with the same chip select signal, and with help of these signals, the hardware address of the circuit is specified [23]. Only one circuit is needed and the address of that one is set to 0x0 (all A-pins connected to ground). See section 6.1.1 to see how the address is configured in the Linux kernel.

The keypad, marked with '3', is connected with seven signals to the MCP23S17 circuit. Four of the signals (ROW1, ROW2, ROW3 and ROW4), routed to the rows of the keypad, are connected as output pins from the expander circuit. The other three signals (COL1, COL2 and COL3), routed to the columns of the keypad, are connected as input to the circuit.

Chapter 6

Configuring the Development Board

This chapter describes how to configure and build a customized Linux kernel for BeagleBoard-xM. The kernel is patched to support SPI and the MCP23S17 circuit. A description is also given for how the SD card is prepared before it can be used together with the BeagleBoard.

6.1 Building a Customized Linux Kernel

Building a Linux kernel is not always needed. Pre-built kernels are available on the Internet and there are also webpages that can generate kernels on request. The need of building a Linux kernel arises when it has to be customized for a special purpose or for a new hardware. BeagleBoard is a type of development board which is commonly used, so there is no problem getting a kernel to work with it. However, using a pre-built kernel does not enable all functionality that is available on the board which is the reason for building a customized kernel.

Due to hardware limitations of the BeagleBoard, the new kernel cannot be compiled on the board itself, but has to be cross-compiled (see section 3.2.3) with a cross-compilation toolchain. There are different ways of cross-compiling a kernel. One is to download a pre-built cross-compilation toolchain and use that for compiling the kernel. But a more suitable way, for this project, is to use BuildRoot, see section 3.2.2 for more information. In the configuration of BuildRoot, an external cross-compiling toolchain[1], Linux kernel 3.1.4 and a root filesystem is enabled. The BeagleBoard is already shipped with a bootloader and therefore BuildRoot does not need to compile another one. To customize the kernel, a kernel patch and a custom kernel configuration file are also enabled in the BuildRoot configuration.

[1]The cross-compilation toolchain that BuildRoot compiles uses uClibc, which is a reduced C library compared to GNU C Library. The tool for updating the firmware of the 3G Broadband Module is written for the latter and does not support uClibc. Therefore BuildRoot is configured to use an external toolchain.

6.1.1 Kernel Patch

By using the default configuration for an omap2plus processor, a Linux kernel (version 3.1.4) will work with BeagleBoard-xM without making any modifications to it. However, the BeagleBoard is equipped with an extension connector, which allows the user to connect external peripherals. This extension connector uses software pin multiplexing, which means that most of these pins can be configured in software for different functionality. If these settings need to be configured differently compared to the default values, the kernel needs to be patched before it is compiled. Two SPI interfaces are located in the expansion connector and since one of these is used to connect the MCP23S17 circuit, the kernel has to be patched to enable those SPI pins. The patch that is used to get SPI to work is based on the patch from [24], but it is modified to get the MCP23S17 circuit to work with its Linux driver. The modified patch is shown in listing 6.1.

```
--- a/arch/arm/mach-omap2/board-omap3beagle.c
+++ b/arch/arm/mach-omap2/board-omap3beagle.c
@@ -30,6 +30,8 @@
 #include <linux/mtd/nand.h>
 #include <linux/mmc/host.h>

+#include <linux/spi/mcp23s08.h>
+#include <linux/spi/spi.h>
 #include <linux/regulator/machine.h>
 #include <linux/i2c/twl.h>

@@ -522,6 +524,62 @@
        return;
 }

+static void __init omap3_beagle_config_mcspi3_mux(void)
+{
+
+        // NOTE: Clock pins need to be in input mode
+        omap_mux_init_signal("sdmmc2_clk.mcspi3_clk",
                OMAP_PIN_INPUT);
+        omap_mux_init_signal("sdmmc2_dat3.mcspi3_cs0",
                OMAP_PIN_OUTPUT);
+        omap_mux_init_signal("sdmmc2_dat2.mcspi3_cs1",
                OMAP_PIN_OUTPUT);
+        omap_mux_init_signal("sdmmc2_cmd.mcspi3_simo",
                OMAP_PIN_OUTPUT);
+        omap_mux_init_signal("sdmmc2_dat0.mcspi3_somi",
                OMAP_PIN_INPUT_PULLUP);
+}
+
+static void __init omap3_beagle_config_mcspi4_mux(void)
+{
```

```
+         // NOTE: Clock pins need to be in input mode
+        omap_mux_init_signal("mcbsp1_clkr.mcspi4_clk",
                 OMAP_PIN_INPUT);
+        omap_mux_init_signal("mcbsp1_fsx.mcspi4_cs0",
                 OMAP_PIN_OUTPUT);
+        omap_mux_init_signal("mcbsp1_dx.mcspi4_simo",
                 OMAP_PIN_OUTPUT);
+        omap_mux_init_signal("mcbsp1_dr.mcspi4_somi",
                 OMAP_PIN_INPUT_PULLUP);
+}
+
+static struct mcp23s08_platform_data mcp23s17_data = {
+        .chip[0].is_present = true,
+        .chip[0].pullups = 0x0070,
+        .base = 240,
+};
+
+static struct spi_board_info beagle_mcspi_board_info[] = {
+        // spi 3.0
+        {
+                .modalias        = "mcp23s17",
+                .platform_data   = &mcp23s17_data,
+                .max_speed_hz    = 5000000, //5 Mbps
+                .bus_num         = 3,
+                .chip_select     = 0,
+                .mode = SPI_MODE_0,
+        },
+
+        // spi 3.1
+        {
+                .modalias        = "spidev",
+                .max_speed_hz    = 48000000, //48 Mbps
+                .bus_num         = 3,
+                .chip_select     = 1,
+                .mode = SPI_MODE_1,
+        },
+
+        // spi 4.0
+        {
+                .modalias        = "spidev",
+                .max_speed_hz    = 48000000, //48 Mbps
+                .bus_num         = 4,
+                .chip_select     = 0,
+                .mode = SPI_MODE_1,
+        },
+};
```

```
+
 static void __init omap3_beagle_init(void)
 {
         omap3_mux_init(board_mux, OMAP_PACKAGE_CBB);
@@ -534,6 +592,11 @@
                         ARRAY_SIZE(omap3_beagle_devices));
         omap_display_init(&beagle_dss_data);
         omap_serial_init();
+
+        omap3_beagle_config_mcspi3_mux();
+        omap3_beagle_config_mcspi4_mux();
+        spi_register_board_info(beagle_mcspi_board_info,
+                ARRAY_SIZE(beagle_mcspi_board_info));

         omap_mux_init_gpio(170, OMAP_PIN_INPUT);
```
Listing 6.1: Kernel Patch.

The patch contains functions and data structures to enable the SPI on the BeagleBoard. The two functions that are added are executed by the kernel to mux the pins to be used as SPI interfaces. The structure called “beagle_mcspi_board_info” contains the settings for the SPI interfaces. Which can be seen in listing 6.1, SPI interface 3.0 is different from interface 3.1 and 4.0. The 3.0 interface is modified to use the driver for mcp23s17 circuit instead of spidev. That driver requires platform data about the circuit, which is given in the structure called “mcp23s17_data”. By default, all chips in the data structure is disabled, so at the first row the MCP23S17 chip with address 0x0 is enabled, see the description of the addresses in section 5.2. The pullup resistors that the MCP23S17 circuit is equipped with are enabled for three of the signals from the keypad. The output value from the keypad is undefined if no button is pressed, so pull-up resistors are required on the output pins from the keypad. The last row in the structure specifies the base of the circuit. The base tells the driver what number the first GPIO pin will be assigned in sysfs (see section 3.2.8). In this case, there are 16 GPIO pins available so the pins will be assigned the numbers 240-255 and grouped together as a gpiochip in */sys/class/gpio*.

6.1.2 Kernel Configuration

In the kernel source tree, there is a default configuration, called “omap2plus_defconfig”, that works with BeagleBoard-xM. However, this configuration does not enable everything on the board that is needed and the drivers for the module and the MCP23S17 circuit are not included. Here are those configs that are enabled in the kernel configuration:

- Enable the BeagleBoard’s USB ports:
  - CONFIG_USB_EHCI_HCD
  - CONFIG_USB_EHCI_HCD_OMAP
  - CONFIG_USB_OHCI_HCD

  - CONFIG_USB_OHCI_HCD_OMAP3
- Enable the BeagleBoard's Ethernet port:
  - CONFIG_USB_NET_SMSC95XX
- The 3G Broadband module needs the ACM driver to work with Linux:
  - CONFIG_USB_ACM
- Enable SPI driver in the kernel:
  - CONFIG_SPI
  - CONFIG_SPI_MASTER
  - CONFIG_SPI_SPIDEV
- Add sysfs interface for GPIO:
  - CONFIG_GPIO_SYSFS
- Add driver for MCP23S17 circuit:
  - CONFIG_GPIO_MCP23S08[2]
- Let kernel maintain /dev tmpfs:
  - CONFIG_DEVTMPFS

6.2 Setting Up an SD Card

BeagleBoard-xM does not have a NAND flash memory, but is equipped with a microSD card reader. Not having a NAND flash means that all data that is needed to boot a kernel is saved on the memory card [22]. The SD card's file system and geometry of the partition table used in this kind of boot process is important to be compatible with BeagleBoard-xM's boot ROM. A FAT file system is required for the boot partition and the partition table geometry of the SD card needs to be: 255 heads, 63 sectors and 512 bytes per sector [25]. The number of cylinders is dependent on these settings and the size of the card. This formula is used to calculate number of cylinders:

$$cylinders = \lfloor sd\_size/heads/sectors/sector\_size \rfloor = \lfloor 3904897024/255/63/512 \rfloor = 474$$

The fdisk command is used to apply these settings to the memory card.

6.2.1 Moving files to the SD Card

The order in which files are moved to an SD card is usually not important, but for the boot partition the order matters. The reason is that boot ROM is only looking for the x-loader

[2]This driver handles both circuit MCP23S08 and MCP23S17.

at the first few entries of the root directory [27]. Therefore, the files to the boot partition have to be moved in the following order:

- MLO (x-loader) - *Shipped with the BeagleBoard.*
- U-boot.bin - *Shipped with the BeagleBoard.*
- uImage (kernel)
- uEnv.tx

The second partition, the root filesystem partition, does not have any ordering requirements. The filesystem is generated by BuildRoot as a compressed tar-file and is extracted with these commands:

```
#sudo mount /dev/mmcblk0p2 /media/ROOTFS
#sudo tar -xv -C /media/ROOTFS -f rootfs.tar
#sudo umount /media/ROOTFS
```

Listing 6.2: Command for mounting and extracting the file system.

6.3 Booting the Kernel Using U-Boot

In the booting process, the first code that is executed is fixed code in the Boot Rom. This code locates the X-loader and passes the control to it. X-loader scans the root directory of the memory card to find U-boot. If U-boot is found, X-loader loads it into the memory and passes the control to it. U-boot is the bootloader and what it does is that it loads the Linux kernel into memory and passes necessary information to it; this includes filesystem location and console port and its baud rate [26].

The version of U-boot used has a set of hardcoded environment variables. These variables are used by U-boot itself, but also sent to the kernel. Among these variables, there are declarations of where the kernel image is located and which console port the kernel will use. These variables are unfortunately defined incorrectly. By placing a file on the boot partition of the memory card with the name “uEnv.txt”, these variables can be overridden.

```
console=ttyO2,115200n8
loaduimage=fatload mmc ${mmcdev} ${loadaddr} uImage
```

Listing 6.3: The “uEnv.txt” file placed on the memory card.

The first line of listing 6.3 redefines the console variable. This variable is passed to the kernel and tells it to what serial port the debug information from the kernel should be outputted. The second line tells U-boot to load the kernel image from the memory card.

Chapter 7

Software

In this chapter, the implementation of the programs are described. In the first section, the communication between the server and the access point is explained. The following two sections describe the software for the access point and the server, respectively.

7.1 Communication between Server and Access Point

The communication between the server and the access points is based on a client-server model, which means that the AP acts as a client and always the one that starts the communication with the server. The server provides the service, in this case accepts or rejects unlock requests that the APs send. In addition, the server also provides the service of sending firmware updates to the APs.

7.1.1 Limit the Traffic in the System

An important aspect in a lock system is the time it takes from when user credentials has been entered until the access point unlocks itself. It is hard to affect the time it takes for the data to pass through the Internet, but by handling the data as soon as it arrives at the server end, the response time can be limited. To be able to have an acceptable unlock time and at the same time support a large number of locks, it is important that each access point is just connected to the server when unlock requests are sent and firmware updates are downloaded, in order to not overload the server. A clear disadvantage with just having an established connection at those points in time is that an AP can be stuck in a state where it never connects to the server. This will happen if the AP does not have a scheduled firmware update and no users are unlocking it. The solution to this problem is the call-ins. The call-in is not a special message, but just a time specified by the server when the AP is required to connect to the server if nothing has happened since the last connection. If a user tries to unlock the AP or if a new firmware update is downloaded to the AP, the server will send a new time for the next call-in to the access point, which replaces the old one. The good thing about the call-ins is that they make all APs regularly connect to the server, including the APs which are not often unlocked by users. By knowing that all APs regularly connect allows the server to centrally store each access point's settings and make changes to them.

7.1.2 Protocol Buffers Messages

The data that is exchanged between the server and the access points is structured in Protocol Buffers messages, see section 3.2.6 for more information. Using this flexible protocol makes it easy to exchange information and at the same time make changes during the development phase if it is needed. The information to be exchanged is divided into several messages; one message is used for each stage in the communication. A message is sent by first serializing it and then outputting it to the stream. At the destination, the data received from the input stream is first stored into a buffer and then parsed back in a message with help of Protocol Buffers' function. That function needs to know what message it is parsing, because that cannot be distinguished from the data itself. Therefore, a message is used as an envelope to carry the rest of the messages. That message is called "LockMessage" and consists of a message type and several different nested messages. The receiver always parses the received data in this message and by looking at the type, it knows whether an optional message is included or not. In listing 7.1, all messages used in the communication are shown. Here follows an explanation of each message and what it is used for:

- **LockMessage** - This message is sent every time data is sent between the server and the access point. The functionality can be likened with an envelope, thus used for holding the messages that are sent. For the receiver to know what is included inside this message, a type is always required. Depending on the type, one or zero optional messages can be included. Some of the types do not need to add an optional message; for example the type "DISCONNECT" which is used to inform the server that the access point is about to disconnect and no more information is needed.
- **LockInfo** - The access point uses this message when opening a new connection to the server. This message has fields for specifying the lock id, name etc., which is used by the server to identify the access point.
- **ServerInfo** - This message is sent as a response to the "LockInfo" message. It includes some information about the server and also a "LockInfoUpdate" message to the access point.
- **LockInfoUpdate** - This message includes updates to the AP. It specifies at what time the lock is required to call-in next time, whether the lock is activated or inactivated, if the server needs new 3G module information and an option describing the name of the access point. There are also options that tell if firmware updates are available, and if so, when it will occur.
- **ModuleInfo** - This message is used to update the server about the 3G Broadband Module that the access point is using. This information can be requested by the server via the "LockInfoUpdate" message or sent by the AP after a firmware update.

- **UnlockRequest** - The access point sends this message when a user tries to unlock it. It includes the user's id and pin code together with a timestamp.
- **UnlockResponse** - The server responds with this type of message, when it receives an "UnlockRequest" message from an access point. This message includes the user id, whether the request is accepted or rejected and a timestamp.
- **FwFileRequest** - This message is used by access points to request firmware updates from the server.
- **FwFile** - This type of message is sent from the server to the AP as a response to the "FwFileRequest" message. Actually, the server will respond with a lot of FwFile messages to each firmware request, where each "FwFile" message contains a small part of the file.

```
package lockprotocol;

option java_package = "com.lock.lockprotocol";
option java_outer_classname = "LockProtos";

message LockMessage {
        enum MessageType {
                SERVER_INFO = 0;
                LOCK_INFO = 1;
                UNLOCK_REQUEST = 2;
                UNLOCK_RESPONSE = 3;
                DISCONNECT = 4;
                FW_FILE_REQUEST = 5;
                FW_FILE = 6;
                MODULE_INFO = 7;
        }
        required MessageType type = 1;
        optional ServerInfo server_info_msg = 2;
        optional LockInfo lock_info_msg = 3;
        optional UnlockRequest unlock_request_msg = 4;
        optional UnlockResponse unlock_response_msg = 5;
        optional FWFileRequest fw_file_request_msg = 6;
        optional FWFile fw_file_msg = 7;
        optional ModuleInfo module_info_msg = 8;
}
message ServerInfo {
        required int32 id = 1;
        required int32 major_version = 2;
        required int32 minor_version = 3;
        required int64 current_time = 4;
        required LockInfoUpdate lock_info_update_msg = 5;
}
message LockInfoUpdate {
        required int64 next_connect = 1;
        required bool activated = 2;
        required bool send_module_info = 3;
        optional string name = 4;
        optional bool need_fw_update = 5;
        optional int64 fw_update_time = 6;
}
message LockInfo {
        required int32 id = 1;
        required int32 major_version = 2;
        required int32 minor_version = 3;
        required string name = 4;
        required string ip_address = 5;
        required bool position_available = 6;
        optional double position_longitude = 7;
        optional double position_latitude = 8;
}
message ModuleInfo {
        required string model = 1;
        ...
        required string config_set_product = 27;
        required string config_set_revision = 28;
        ...
        required string upgrade_state = 42;
        ...
}
message UnlockRequest {
        required string user_id = 1;
        required string pin_code = 2;
        required int64 timestamp = 3;
}
message UnlockResponse {
        required string user_id = 1;
        required bool accepted = 2;
        required int64 timestamp = 3;
}
message FWFileRequest {
        required string version = 1;
}
message FWFile {
        required string filename = 1;
        required bool last_packet = 2;
        required bytes data = 3;
}
```

Listing 7.1: The Protocol Buffers file that defined the messages that were used.

7.1.3 The Order of the Messages

As described above, the “LockMessage” is always sent between the server and the APs and can be considered as an “envelope” to the other messages. The messages described here are the messages included inside the “envelope”.

The access point starts the communication by sending the “LockInfo” message to the server. This message contains enough information about the AP for the server to know whom it is talking to. Based on that information, the server makes the decision whether to continue the communication or not. If the server accepts the connection, the server responds with a “ServerInfo” message to the AP. If the server is requesting 3G module information or if the firmware has been updated or access point restarted since the last connection, the AP now sends a “ModuleInfo” message to the server. The AP checks if user data is available and in that case sends an “UnlockRequest” message to the server. The server responds to the AP with an “UnlockResponse”, where the decision of the request is included. If the server has indicated, in an earlier message, that a firmware update is available and the time when this should happen is fulfilled, the access point will now send “FwFileRequest” to request the file from the server. The server sends the file, which is divided into several “FwFile” messages. Regardless of what has been done during the connection, the AP will end the communication by sending a message of type “DISCONNECT”.

7.1.4 Sending Subsequent Messages

Protocol Buffers does not have the functionality to know when a message ends. Since messages are frequently sent between the server and the access points, situations where several messages are stored in the same buffer at the same time occurs. In those situations, the functions for parsing the messages will parse all messages in the buffer as one message and then report an error. In the suggested solution by [28] and in the way it is handled, the size of the upcoming message is first sent to the stream and then the message itself. This is illustrated in figure 7.1. At the receiving end, the size is first read and given as an argument to the Protocol Buffers functions to parse the message.

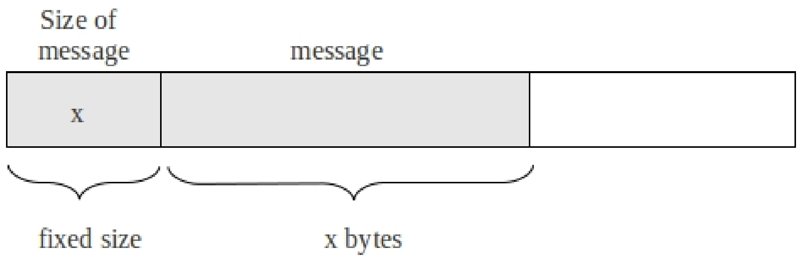


Figure 7.1: Illustration of a message and its size in a buffer.

7.1.5 Encrypted Communication

In a lock system like this, it is important that the communication is encrypted to prevent sensitive information from being eavesdropped and unlock requests from being inserted into communication. If information would leak or if unlock requests would be inserted, it can have devastating consequences and let unauthorized persons into rooms or building that they usually do not have access to. To prevent this from happening all information sent between the server and the access points is encrypted with SSH. The lock system server runs an SSH server and each access point sets up and pipes all traffic through an SSH tunnel. The server does not accept access points that do not encrypt the traffic.

7.1.6 Sending Firmware Updates

The firmware updates are sent from the server to the access points to be able to remotely update each 3G Broadband Module. The file used to update the module is a rather big file, about 20 MB. Therefore it is required to split it up, because Protocol Buffers cannot handle messages that big. The message used to send each piece is the “FwFile” message as described above. No messages are used to acknowledge each piece, because that would slow down the transfer. Instead the server sends all pieces directly after each other without waiting for a response from the access point. TCP is used in the communication and since it guarantees data delivery, acknowledgements are not actually needed.

7.2 Access Point Software

This section describes how the program written for the access point is implemented. The program reads user input from the keypad, sets up an Internet connection through the 3G Broadband module and encrypts the traffic over an SSH tunnel etc.

7.2.1 Program Flow

The functionality of the access point program is implemented in a while-loop. The program connects to the server if user data is available, if it is time for a call-in or a firmware update is scheduled. A more detailed description of the program flow is given in figure 7.2.

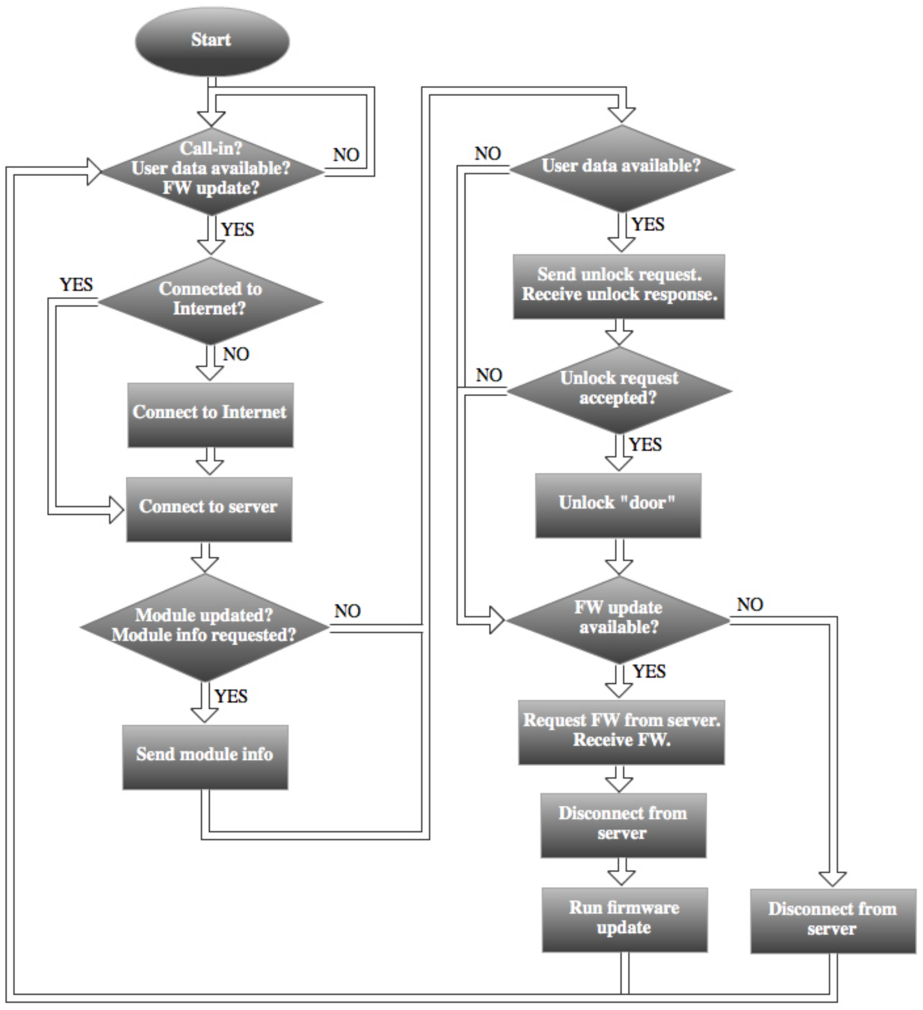


Figure 7.2: Flowchart explaining the program flow of the access point.

7.2.2 Communicating with 3G Broadband Module

When the module is connected to the development board, the ACM driver enabled in the customized kernel is loaded. Three different virtual serial ports, “ttyACM0”, “ttyACM1” and “ttyACM2”, are created under “/dev” in the file system. The interaction with the module is done through AT commands, which are sent to those serial ports. To send the AT commands and receive results from the module, the ports are opened as regular files, commands are written and results are read as in usual Linux file I/O.

7.2.3 Connecting to the Internet

To connect to the Internet through the module, a subset of AT commands are sent in the order explained in figure 7.3. Instead of giving the name of the AT commands, the functionality of them is explained. All AT commands are given in Appendix A.

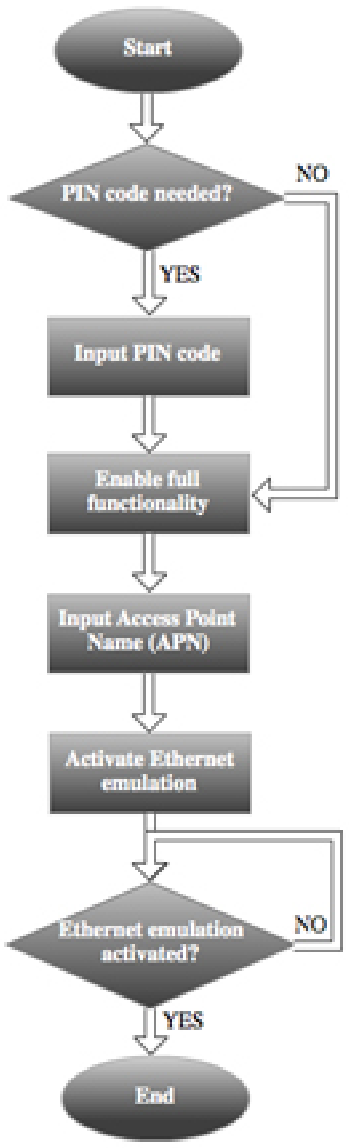

Start
PIN code needed?
NO
YES
Input PIN code
Enable full functionality
Input Access Point Name (APN)
Activate Ethernet emulation
Ethernet emulation activated?
NO
YES
End

Figure 7.3: Flowchart explaining how an Internet connection is set up.

After the emulated Ethernet interface becomes available, a DHCP client is executed to get the IP address and DNS servers.

7.2.4 SSH Tunnel

To encrypt the traffic between the access point and the server, an SSH tunnel is used. The program sets up the tunnel every time the 3G Broadband module is required to connect to the Internet. To create the encrypted tunnel, a child process is forked to be used for calling the system command, given in listing 7.2, that starts an SSH tunnel. The tunnel itself is not closed when the Internet connection is lost, so by saving the process id of the child, the SSH tunnel can at any time be killed and replaced with a new one. If the SSH tunnel is not killed when the Internet connection is closed, the local port will be reserved until the development board has been restarted. Replacing the tunnel is a requirement, because the module always gets a new IP address when it reconnects.

```
ssh p1@p2 -L p3:localhost:p4 -N
        p1 : username of a user on the remote machine
        p2 : IP address or DNS name of the remote machine
        p3 : port on the local machine
        p4 : port to which traffic is tunnelled
```

Listing 7.2: SSH Command.

7.2.5 Reading User Input From Keypad

The reading of user input is divided into two different threads. One thread for reading individual key presses and the other thread for analysing the input and accept it if it is in the expected format. The expected format of the user input is the user id followed by the '*' character and then the pin code followed by the '#' character, as shown in listing 7.3. As soon as input is found in that format, that thread will make it available to the main thread, which forms an unlocks request as soon as possible and sends it to the server.

```
1234*0000#
        -1234 : user id
        -0000 : pin code
```

Listing 7.3: An example of correctly entered user data.

The keypad is connected to seven pins of the MCP23S17 circuit, which in turn is connected to the BeagleBoard as described in section 5.2. The Linux kernel is configured to use the driver available for that circuit, which means that all GPIO pins available will be exported via the virtual file system, sysfs, to the directory */sys/class/gpio*. Reading or changing the value of such a pin is just as simple as reading or writing to a file, respectively. The structure of the keypad is formed as a matrix, as described in section 3.1.1, and finding out which keys are pressed is done one key at a time. The scanning of the keys is done in a loop in the first thread described above, which is running constantly to discover key presses as soon as they appear. Four of the seven pins on the keypad are connected to the rows and used as inputs to the keypad while the remaining three are connected to the columns and used as outputs from the keypad. If no keys are pressed, the output signals are by default '1' due to pull-up resistors. Pressing a key gives the output signal the value of the input signal for the specific key. To find out if and which keys are pressed, all input pins to the keypad are initially set to '1'. Changing the input pins, one at a time, to the value '0' and at the same time checking the output pins, all keys are covered and key presses can be discovered. In figure 7.4, it is illustrated how the program distinguished if a key is pressed or not.

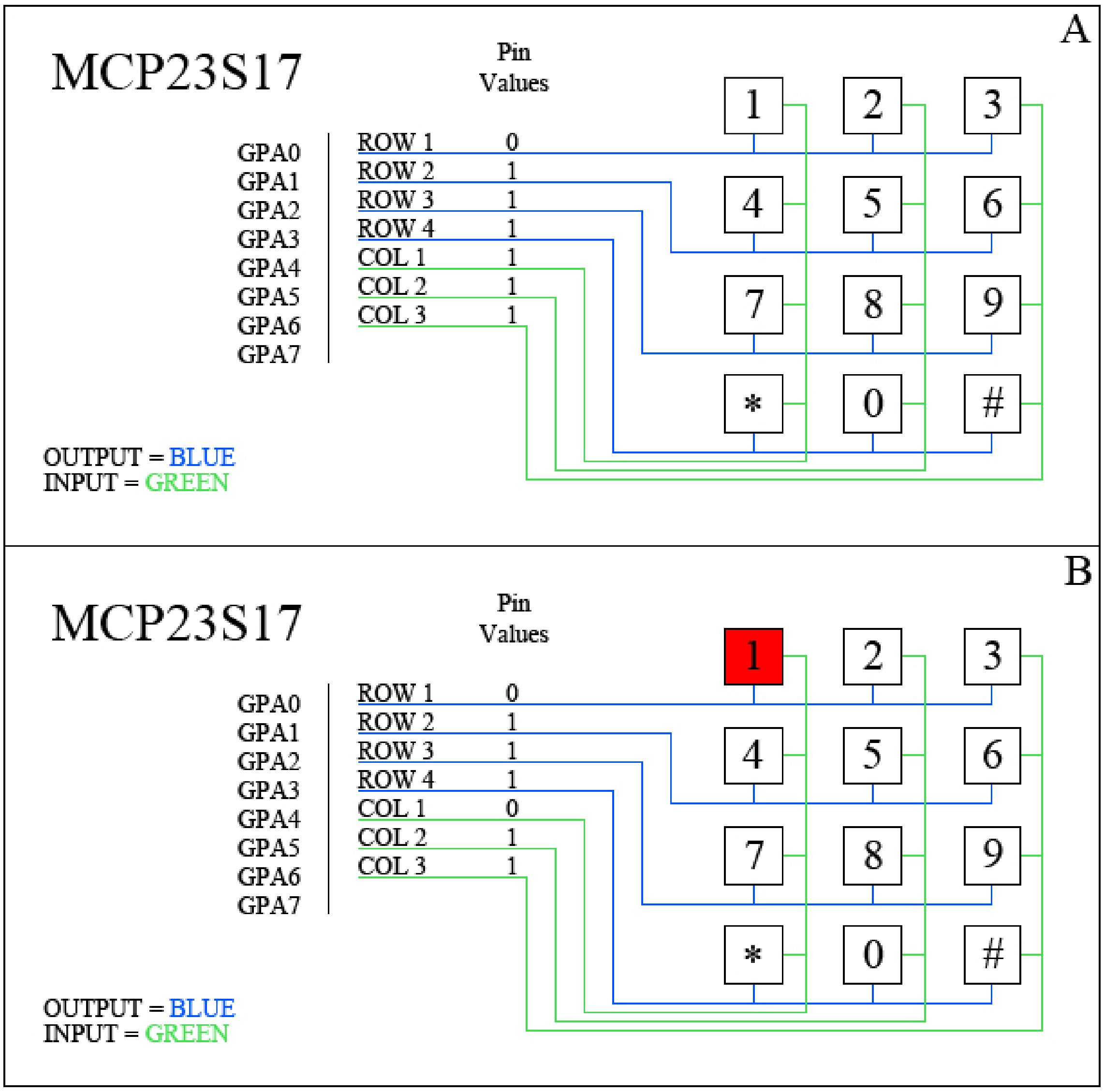


Figure 7.4: How key presses are recognized. In A, no keys are pressed and all column values are ’1’. In B, key ’1’ is pressed and therefore col 1 gets the value of row 1.

The second thread recognizes user input patterns from the key presses that the first thread gets from the keypad. Every time a new character is returned, it loops through all characters that are available in the buffer to see if the input has the expected format. If the input does not follow the rules of the input, for instance if the ’#’ character arrives in the input before a ’*’ character has arrived, all data up to the ’#’ character is removed. All key presses are kept in the buffer until a decision can be made whether the input is valid or not, but for maximum of 10 seconds. If 10 seconds is reached since the last key press, the buffer is cleared. To be able to send the unlock request to the server quickly, a function is available that returns a Boolean value saying whether there are key presses in the buffer or not. By regularly calling this function, the main thread can directly establish a connection to the server when a user starts typing. At the time when the user completes the typing, the access point has established a connection to the server and the unlock request can immediately be sent. The 10 seconds limit explained above is also preventing the AP from being stuck in a state with an established connection to the server and just waiting for user input.

7.2.6 Access Point Locked or Unlocked

As mentioned in the limitation part, section 1.3, the access point is not connected to a real lock, but instead two LEDs are used to indicate whether it is locked or unlocked. The LEDs are located on the development board and can be seen in figure 7.5. To also inform the user that the access point really recognizes the key presses, these two LEDs starts to blink simultaneously as soon as a user presses the first key. They will keep doing that until the unlock request is accepted/rejected or the input is considered invalid.

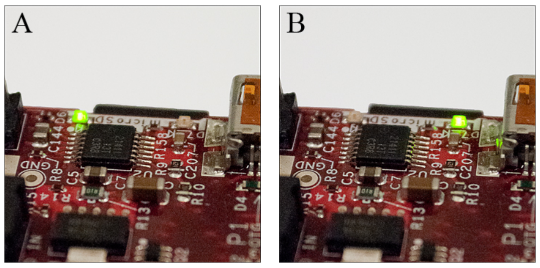


Figure 7.5: In A, the access point is locked and in B unlocked.

7.2.7 Inactivation of the Access Point

The access points can be inactivated from the server. When an access point is inactivated, it stops accepting user input and sending unlock requests. Since no unlock requests are sent, the load of the server is reduced compared with if the access points would function normally and just let the server reject all unlock requests for inactivated access points. Considering the flowchart in figure 7.2; when an access point is inactivated everything dealing with user data is skipped. If an access point is set to be inactivated, it will apply the inactivation next time it connects. If a user tries to unlock an access point in the time period between when the access point has been inactivated at the server side and the access point has received the inactivation, an unlock request will still not be sent. Since when the access point sets up a connection to the server, the server will send the new configuration before it is time to send the unlock request and therefore the unlock request will be skipped.

7.2.8 Getting and Formatting the GPS Data

The 3G Broadband module used in the access point is equipped with a GPS receiver, which is used to let the server know where each access point is located. To enable the GPS, two different AT commands are sent to the module; one command for changing the settings and one to start receiving the data. Those commands are added and explained in Appendix A. The GPS data from this receiver is formatted in NMEA 0183 application layer protocol, which is explained in section 3.2.5. Each fix received from the GPS receiver contains the position together with several other values. Those different values are:

- $GPGSV - GPS Satellites in View
- $GPGLL - Geographical Position, Latitude/Longitude
- $GPGGA - Global Positioning System Fix Data
- $GPGSA - GPS DOP and Active Satellites
- $GPGST - GPS Pseudorange Noise Statistics
- $GPRMC - Recommended Minimum Specific GPS/TRANSIT Data

What the server is interested in is the geographic position, which is a part of the $GPGLL row above. An example of a $GPGLL row is given in listing 7.4.

```
$GPGLL,5742.3197,N,1156.5081,E,125613,A*1F
```

Listing 7.4: Latitude/Longitude position given in NMEA format.

The row in listing 7.4 contains the following: $GPGLL, latitude position, north or south, longitude position, east or west, time when fix is taken, data validity and a checksum. However, that row contains the position in NMEA format and not in decimal. Therefore, a conversion is required. To convert from NMEA to decimal the following formula is used:

$$toDecimal(x) = \lfloor x/100 \rfloor + ((x - (\lfloor x/100 \rfloor * 100))/60)$$

Here is an example when the NMEA data above is converted into decimal:

	NMEA	**Decimal**
Latitude	5742.3197 N	57 + (42.3197 / 60) N = 57.705328 N = 57.705328
Longitude	1156.5081 E	11 + (56.5081 / 60) E = 11.941801 E = 11.941801

In the example above, the latitude position is North and longitude is East. Those directions do not affect the decimal conversion. But if latitude would be South or longitude would be West, the decimal representation is negative.

7.2.9 Updating the Firmware of a Module

When a firmware update is added to the system, the server decides when each access point will be updated by sending the time to it. At the given time, the access point sends a request to the server and the server starts to send the file to the access point. After a completed file transmission, the access point first starts with closing the connection to the server, closes the SSH tunnel and disconnects from the Internet. Updating the module means that the connection to the Internet is lost, so at least closing the SSH tunnel is important to be able to start it again after the update. Having everything closed, the update of the module is started and will be completed within 5 minutes.

7.3 Server Software

The server plays a crucial role in the system. It handles user authentication, firmware updates of the 3G Broadband Modules and also gives the administrator of the system a good overview of all access points. The server application is written in Java with a graphical user interface. This section gives a description of how the server application is implemented.

7.3.1 Application implemented in MVC

Since the server application is developed with a graphical user interface, it follows the model-view-controller design pattern, see section 3.2.4, to get well-structured code. The functionality of the server is implemented in the model part, thus handling the connection with the access points, user authentication and the firmware updates. The windows are part of the view and the controller handles the key presses and the mouse clicks made by the user.

The communication between the model and the view is implemented with the Observer/Observable interface available in the Java framework. When information is changed in the model, the view is informed through the interface and can request new information from the model.

The controller is implemented as an action listener and is informed about the user interaction that is made in the view. When an event occurs, the controller calls the corresponding function in the model that will handle the required changes for that particular event.

The model is split into several classes to separate functionality; the view is split into several classes to separate different parts of the view; the controller is split into several classes to just handle parts of the view.

7.3.2 Using a Database to Save Information

The database is used to save all the information that the server collects from the access points, together with users' access rights and information about firmware updates uploaded in the system. Having everything stored in a database makes it easy to take backups and recovers from a system crash or a system reboot.

The connection to the database is implemented in a Java class. That class contains functions for inserting, updating and selecting data out of the database. Most of the data passed to and from this class is passed in objects. For instance, selecting information about an access point from the database is returned as an access point object. To update the information about a user in the database, a user object is passed to a function in the database class.

Since the system requires identification of the user, the pin codes that the users use have to be stored in the database. However, saving the pin codes in clear text is risky if the database in some way would leak out to the public. Not just that the pin codes for this system would leak, but users may use the same pin codes in other situations. Therefore, the pin codes are first salted and then the hash is stored. The salt for each pin code is generated every time a pin code is changed and is based on the current time. The hash is calculated with a MD5 hash function.

The structure of the database is split into nine different tables. The entity-relation diagram of the database is given in figure 7.6.

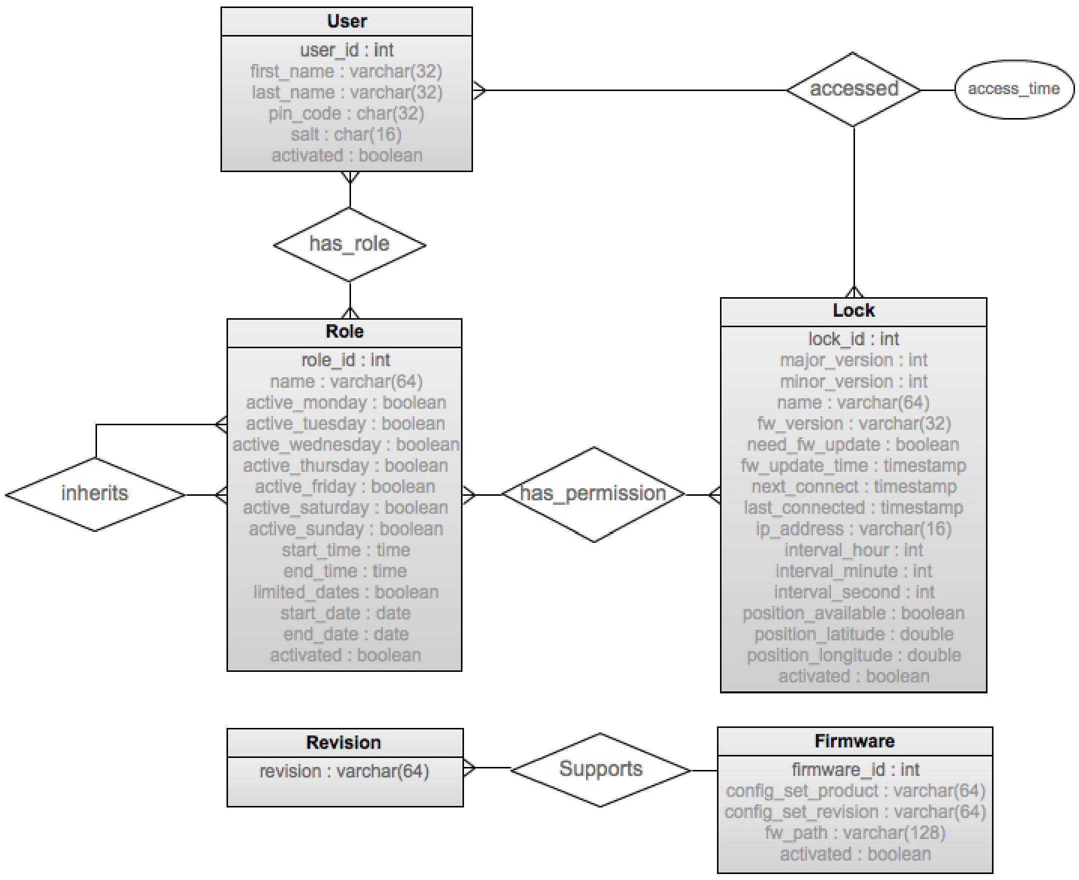


Figure 7.6: Entity-Relation diagram over the database used in the lock system.

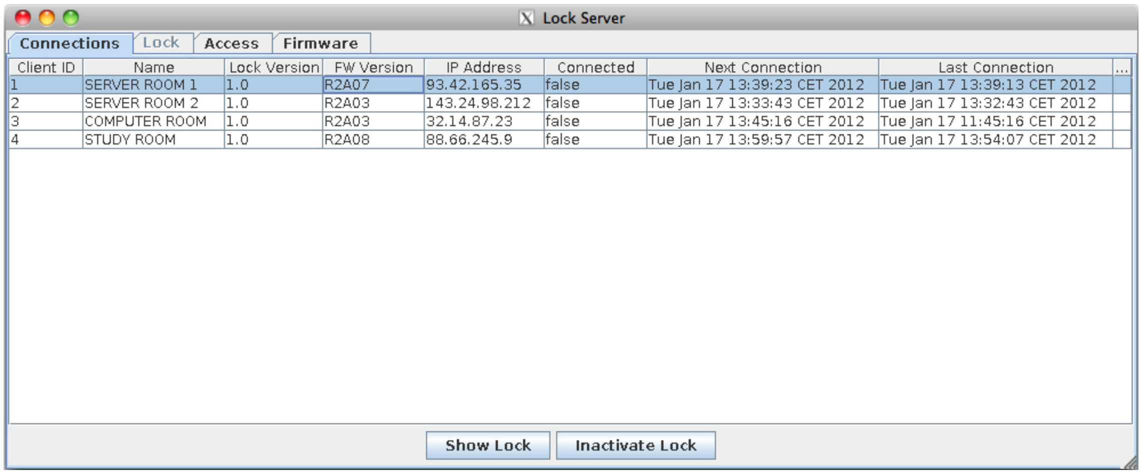
7.3.3 Handling Connections From Access Points

A system requirement was to support a large number of access points. Therefore, the server application is written in a general way to have the ability of expanding the system and use it in a larger scale. With this server software architecture, the only limitation is the hardware. As written in section 7.1.1, each access point does not have an established connection to the server at all times. The APs are only connected to the server in case of a call-in, an unlock request or a firmware update. It is a conscious thought to support more APs than the server could handle if the access points would always be connected.

The connection to the server is always established from the access point, as it usually is in a server-client model. With an established connection, the server first checks that the traffic is encrypted; otherwise the connection is directly closed. That check is based on the IP address. When traffic is sent through SSH-tunnels, the traffic always seems to be sent from the local machine, which in this case is the server itself. Checking that the source IP address is equal to “127.0.0.1”, only encrypted traffic generated from outside of the server is accepted. When an AP discovers that the server accepts the communication, the first message that includes the access point id is sent. The server checks that there is an entry in the database for that specific AP and then a thread is created to handle the AP for the rest of the communication. This thread runs until the AP sends a message to indicate the end of the communication.

The system uses call-ins to ensure that each access point regularly connects to the server. Every time an access point establishes a connection, the server sends an absolute time when the AP is required to connect next time. The reason for using absolute time is consistency. Absolute time is needed for timestamps in the unlock requests and therefore also used here. A requirement to get absolute time to work is that all clocks within the system are synchronized. To ensure that synchronization, the current time of the server is sent at every new connection. The absolute time sent to the access point is based on an interval given to the AP and calculated by adding the interval to the current time. At the server side, each AP is given a default value for the call-in interval which manually can be changed for each AP. The server supports values from 1 second up to 24 hours. The value of the interval needs to be adapted to each system such that the server is not overloaded. A system with a small number of access points can let them connect more often than a system with a larger number of access points. A short interval is recommended for newly added access points to let it fast apply changes in the configuration.

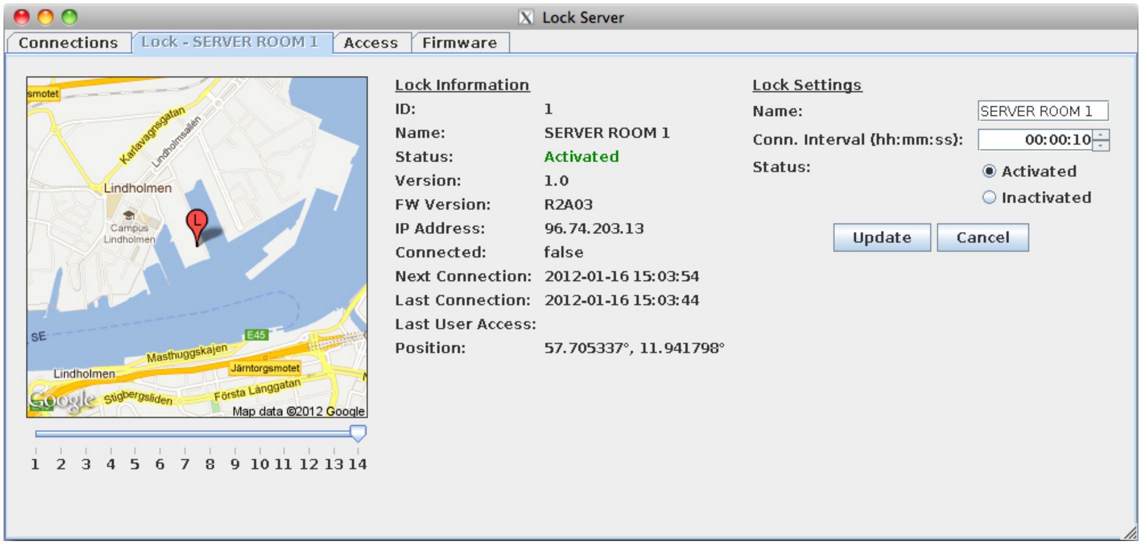
In figure 7.7, the GUI of the connection part of the application is shown. In that view, all access points added into the system are listed. For each access point, information about the name, firmware version, IP address and when it was last connected to the server etc. is shown.



Client ID	Name	Lock Version	FW Version	IP Address	Connected	Next Connection	Last Connection
1	SERVER ROOM 1	1.0	R2A07	93.42.165.35	false	Tue Jan 17 13:39:23 CET 2012	Tue Jan 17 13:39:13 CET 2012
2	SERVER ROOM 2	1.0	R2A03	143.24.98.212	false	Tue Jan 17 13:33:43 CET 2012	Tue Jan 17 13:32:43 CET 2012
3	COMPUTER ROOM	1.0	R2A03	32.14.87.23	false	Tue Jan 17 13:45:16 CET 2012	Tue Jan 17 11:45:16 CET 2012
4	STUDY ROOM	1.0	R2A08	88.66.245.9	false	Tue Jan 17 13:59:57 CET 2012	Tue Jan 17 13:54:07 CET 2012

Figure 7.7: The GUI that lists all access points within the system.

By pressing the button named “Show Lock” in figure 7.7, the view is changed to the view in figure 7.8. This view shows more detailed information about an access point together with different settings. Starting from the left, a map is included to show where the access point is located. Beneath the map, there is a slider to zoom in or out in the map. The next column contains basically the same information as in the connection view except for the status of the access point and coordinates of the position. The last column contains the settings that can be made: changing the name, the interval between each connection and if the access point should be activated/inactivated.



Lock Information

ID:	1
Name:	SERVER ROOM 1
Status:	Activated
Version:	1.0
FW Version:	R2A03
IP Address:	96.74.203.13
Connected:	false
Next Connection:	2012-01-16 15:03:54
Last Connection:	2012-01-16 15:03:44
Last User Access:	
Position:	57.705337°, 11.941798°

Lock Settings

Name:	SERVER ROOM 1
Conn. Interval (hh:mm:ss):	00:00:10
Status:	[✓ Activated] Inactivated

Figure 7.8: The GUI for showing more information about a single access point.

7.3.4 Handling Users and Their Permissions

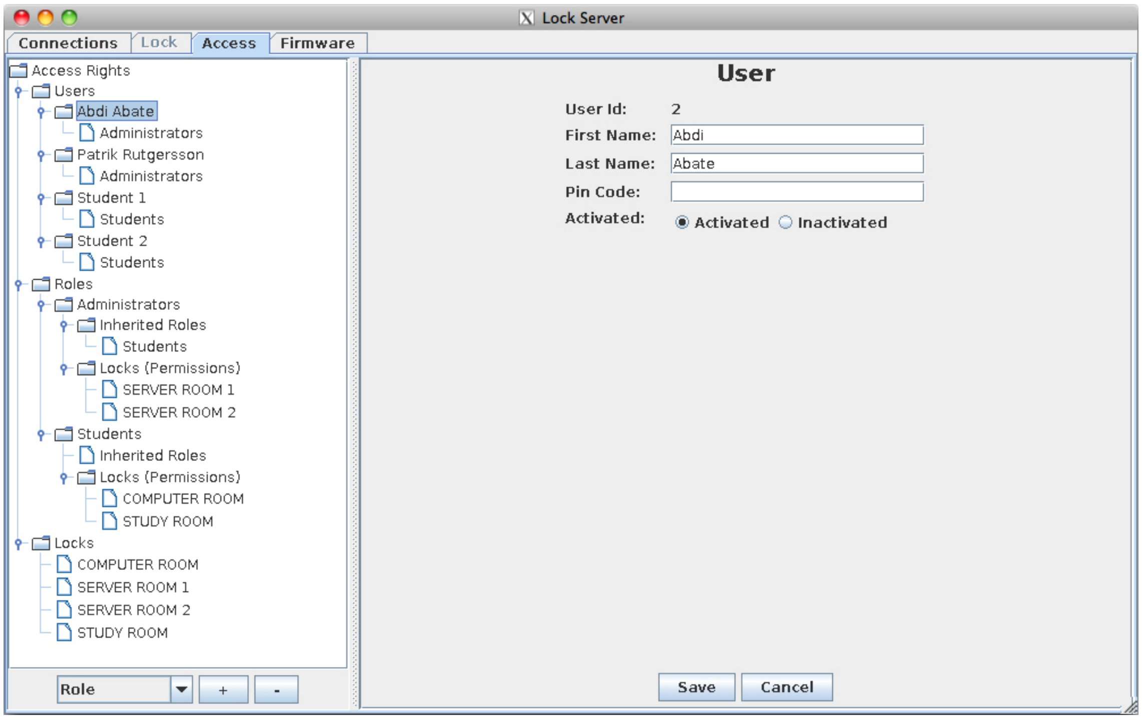
Users and APs is handled according to the solution that the authors of [7] have in a combination with a type of Role-Based Access Control (RBAC) solution. The authors suggest a solution where rules are created for each access point. Each rule is based on weekly access rights, which means that the days and the times when the rule is active during one week is specified. One single access point can only be assigned per rule. Each user is assigned a number of rules to get access to the access points he/she should have access to. A drawback with this solution, which the authors have listed as future work, is that the management of rules and users in the system get complex when the system grows. In their future work they suggest a solution where users can be grouped together and the rules are instead assigned to groups. A better solution, which is used in this master thesis, is a combination of their weekly access rights together with an RBAC model.

The Role-Based Access Control model implemented in this system implements the advanced model called RBAC1 that is discussed in [30], except for the sessions. Sessions are actually a part of the minimum requirement for an RBAC solution, but is not applicable in this system. Sessions are needed when a more advanced RBAC solution is implemented that includes tasks activating several roles at the same time. In this system, it is enough to use one role at a time containing the permission to open the AP the user is trying to unlock. Implementing RBAC1 model means the RBAC solution needs to be able to handle role-role relations. Hence, one role can inherit another role.

Each AP within this system is considered a permission. Each role can be assigned all or a subset of these permissions, but also assigned other roles. At what days and during what time the role is active can also be specified. To enable the users to access the APs, the users are assigned one or several of these roles.

Each time a user wants to unlock an access point, the AP sends an unlock request to the server. As soon as the server gets the unlock request, the server first checks that the user id and the pin code match. With matching user credentials, the server extracts the roles the user is assigned to and tries to find the AP where the unlock request arrived from in at least one of these roles. If the server finds such a role, the unlock request is accepted assuming the user account, the AP and the role are activated.

The access control in the system is managed through the server application's GUI. In the interface users, roles and access points can be added, removed or edited. Users can be assigned roles and roles can be assigned permissions (access points) and other roles for inheritance. In figure 7.9, the view where a user is edited is shown. The list, in the left part of the figure, contains the access tree. Higlighting a row in the list makes it possible to manage the selected item with the dropdown box, '+' and '-' buttons beneath the list. The dropdown box contains possible things that can be added to the selected item. In the figure a user is selected: pressing the '-' button will remove the user and pressing the '+' button will add a role.



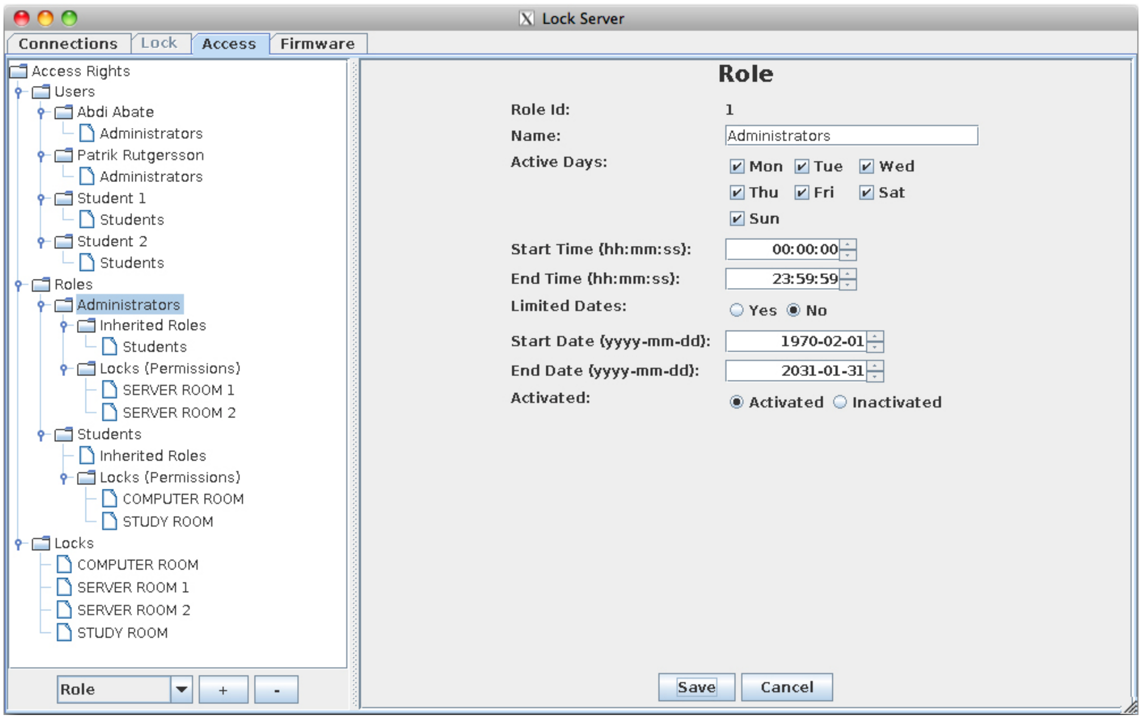
- Access Rights
  - Users
    - Abdi Abate
      - Administrators
    - Patrik Rutgersson
      - Administrators
    - Student 1
      - Students
    - Student 2
      - Students
  - Roles
    - Administrators
      - Inherited Roles
        - Students
      - Locks (Permissions)
        - SERVER ROOM 1
        - SERVER ROOM 2
    - Students
      - Inherited Roles
      - Locks (Permissions)
        - COMPUTER ROOM
        - STUDY ROOM
  - Locks
    - COMPUTER ROOM
    - SERVER ROOM 1
    - SERVER ROOM 2
    - STUDY ROOM

User

User Id:	2
First Name:	Abdi
Last Name:	Abate
Pin Code:	
Activated:	[✓ Activated] Inactivated

Figure 7.9: The GUI for editing a user.

By highlighting a role in the access tree the view is changed for editing a role as shown in figure 7.10. In this view the name of the role and during what times it will be active can be changed.



- Access Rights
  - Users
    - Abdi Abate
      - Administrators
    - Patrik Rutgersson
      - Administrators
    - Student 1
      - Students
    - Student 2
      - Students
  - Roles
    - Administrators
      - Inherited Roles
        - Students
      - Locks (Permissions)
        - SERVER ROOM 1
        - SERVER ROOM 2
    - Students
      - Inherited Roles
      - Locks (Permissions)
        - COMPUTER ROOM
        - STUDY ROOM
  - Locks
    - COMPUTER ROOM
    - SERVER ROOM 1
    - SERVER ROOM 2
    - STUDY ROOM

Role

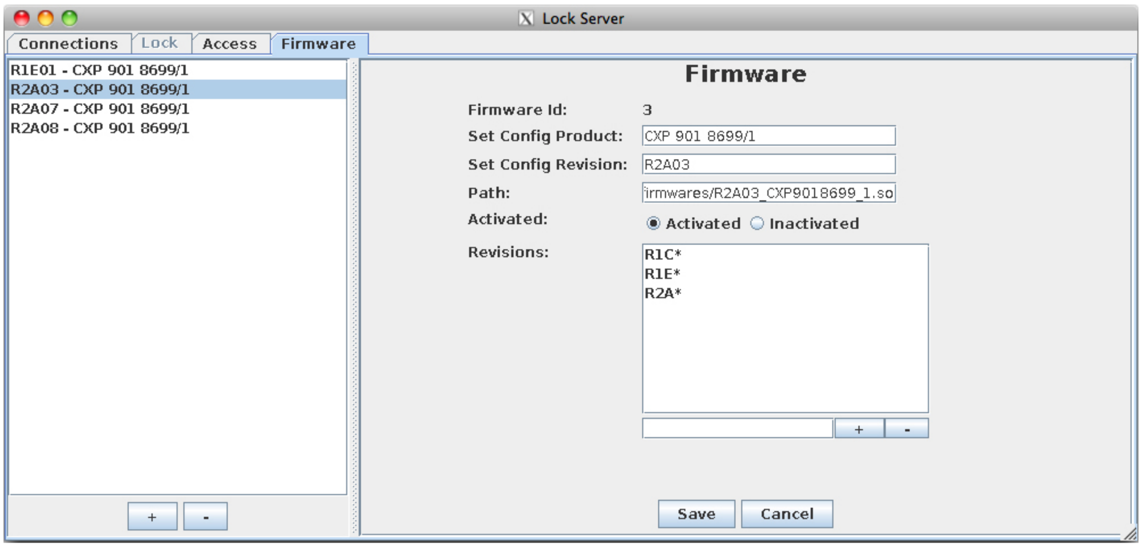
Role Id:	1
Name:	Administrators
Active Days:	✓ Mon ✓ Tue ✓ Wed ✓ Thu ✓ Fri ✓ Sat ✓ Sun
Start Time (hh:mm:ss):	00:00:00
End Time (hh:mm:ss):	23:59:59
Limited Dates:	Yes [✓ No]
Start Date (yyyy-mm-dd):	1970-02-01
End Date (yyyy-mm-dd):	2031-01-31
Activated:	[✓ Activated] Inactivated

Figure 7.10: Changing information about a role is made via this GUI.

7.3.5 Firmware Updates

The implemented solution lets the administrator of the system add firmware updates and then the server handles the distribution. There are different models of the 3G modules and all of them do not use the same firmware updates. To be able to send the right firmware update, the server needs to know which version of the module each specific AP uses. Since an AP can at any time change which module type it is using, for instance to get the latest version with better bandwidth, this information is never saved into the database. Instead, the server requests this information or the AP sends it when the module has been updated or the access point has been restarted, and the server chooses the latest firmware update available for that model. All updates does not support to update from all earlier firmware revisions, but some just support a subset of the revisions. This information is also added to all firmware updates in the server, to let it chose the right update for each AP. The GUI for handling the firmware updates can be seen in figure 7.11.

Every time a new connection is established from an access point to the server, the server checks if there is a new firmware update available for the AP's 3G module. If there is, the server will indicate that in the message together with the time when the update should occur. The access point software supports to update the firmware at any time, but the server is implemented in such a way that it always sets the time of the update to the same time as the next call-in.



R1E01 - CXP 901 8699/1
R2A03 - CXP 901 8699/1
R2A07 - CXP 901 8699/1
R2A08 - CXP 901 8699/1

Firmware

Firmware Id:	3
Set Config Product:	CXP 901 8699/1
Set Config Revision:	R2A03
Path:	irmwares/R2A03_CXP9018699_1.so
Activated:	[✓ Activated] Inactivated
Revisions:	R1C* R1E* R2A*

Figure 7.11: From this graphical interface firmware update can be added/removed or edited.

7.4 Software-Based Access Point

Even if only one real access point is developed, the server still has to handle several connections at the same time. To be able to test this, a software-based access point was also implemented entirely in Java. It neither uses a 3G module nor a keypad. The purpose of it is to just test simultaneous connections.

The software-based access point was developed early in the project which turned out to be good. It was advantageous to use the software-based access point in situations when the development of the server and the access point programs were independent, for instance the user management for the server and getting the module to work with the access point program. The development of these programs were made in parallel and therefore it was not always possible to test the functionality of the server with the physical access point, since required functionality was not implemented yet. A graphical user interface was also developed for the software-based access point to easily demonstrate the system. The GUI can be seen in figure 7.12.

However, the software-based access point does not implement the full functionality that the physical access point does. Since the software-based access point does not use a 3G module, there was no reason of implementing this functionality in the program.

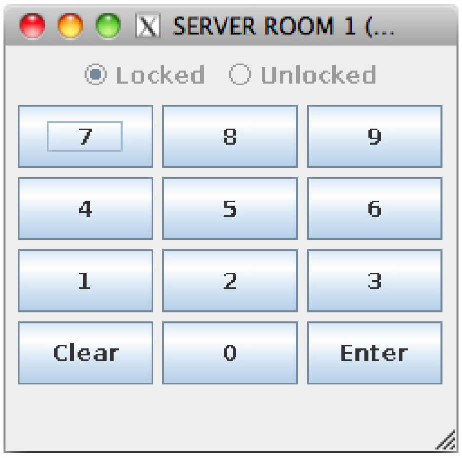


Figure 7.12: The graphical interface for the software-based access point.

Chapter 8

Summary

The server application is programmed according to the Model-View-Controller design pattern to separate the view from the code that handles the server's internal functionality. From the graphical user interface the whole functionality of the server can be seen and handled; getting an overview of all access points and add new firmware and new users to the system. To be able to handle a larger set of users, the management of the users and their access rights is based on Role-Based Access Control. Each user can be given a name, a pin code and whether the user is activated or inactivated. To assign permissions to the users, each user is also given different roles. For each role, times and days are specified to configure when it will be active along with the permissions (access points) that are included. A role can inherit the permissions of other roles as well. A functionality that handles the firmware updates is added to be able to update the 3G Broadband modules and it is implemented to support future 3G Broadband modules. The server has the firmware updates and which one it chooses to send is based on the information that each access point sends to the server about its module. The server makes the decision if and when an access point will be updated and in such a situation the file is distributed in pieces from the server to the access point.

The access point is based on a development board called BeagleBoard-xM that runs a Linux kernel. It gets its Internet connection from a 3G Broadband module connected via USB and uses a keypad, which is connected via SPI, to get user input. The access point establishes a connection to the server only when information is exchanged; i.e. when unlock requests are sent, a firmware update is needed or it is time for a call-in. A call-in is a time given by the server that defines when the access point is required to connect to the server if nothing has happened since the last connection. The access point uses two LEDs to indicate whether it is unlocked or locked, instead of using a real lock. When the access point receives a response from an unlock request, it indicates the result by lightning the LED that represents the outcome of the unlock request.

When the user starts to type on the keypad, the access point establishes an encrypted connection to the server. This is done to minimize the time to get back the result from an unlock request. The average time it takes from the point when the user has finished typing until the access point shows the result of the unlock request is about one second.

The implemented system is a well functional electronic lock system based on 3G Broadband modules. The central server is able to handle many concurrent access points. The server has the information of where and when users have access to the different locations, so at every unlock request the server is contacted to make the decision whether to unlock the access point or not. The system uses 3G to get an Internet connection and that works very well. The connection does not have any negative effects on the functionality and works at least as good as a wired connection.

Chapter 9

Discussion and Conclusions

In this chapter we discuss the results of this master thesis and how some parts of implementation could have been done differently.

As mentioned in the summary chapter, the average time for an unlock request was about one second. Depending on situation, one second can be considered too long and also annoying. In this solution, the user id and pin code is sent to the server at the same time. When the server has received the user credentials, the server checks if it is correct and if the user has access to that access point. To minimize the time, the user id can instead be sent as soon as the user has entered it. The server prepares the unlock response by looking at the access rights of the user to decide whether to accept or reject it and then waits for the pin code. If the received pin code matches the one stored in the system, the server will respond with the prepared result, otherwise the unlock request is rejected. When the server decides whether an unlock request is accepted or rejected, it loops through the user's roles and tries to find a role including the access point. Hence, this operation is heavier for users with a large number of roles than users with a small number. In this alternative solution, this operation is done when the user id is received instead of doing it after when the pin code is received, which means that the biggest response time reduction would be seen for users with a large number of roles.

Testing of the physical access point was only done at the office, where full 3G coverage was available, which means that no testing was done when the access point was connected to the Internet with just EDGE or GPRS coverage. However, the relevance of testing the time it would take with EDGE or GPRS coverage can be discussed since the coverage is improved all the time.

The testing of the server was done with just one single real access point. A software-based access point was implemented to be able to test the most crucial part of the server with several access points; not including the firmware update functionality. If 3G module information would have been simulated in the software-based access point, even tests of the sending the firmware updates could have been done more thoroughly. The system worked well for updating one access point with new firmware, but the system should also be tested with simultaneous transfers.

The encryption between the access points and the server was based on SSH tunnels. There was one single drawback with using SSH and that was when access points were added. When using SSH, host keys are preferably used to connect to the remote machine. Adding a new access point means that a key also has to be installed on the server side. This process is currently not automated, since there is no good way of knowing if the key belongs to the right access point. At the moment, there is also no connection between the ID of an access point and the key installed at the server side. But it is hard to tell if access points should be automatically added to the system or not? Maybe it is good enough to manually add access points to be sure they are legitimate.

Overall, the system works well and the issues discussed above are actually small problems that do not really affect the result of the demo system. Those things would have been fixed if more time were available or if a real product would be produced.

Chapter 10

Future Work

A further extension to this master thesis and to the access point used in this system will be to port the application to the 3G Broadband module. That means to eliminate the development board and run the access point program directly on the module's processor and to decrease the physical size and reduce costs and power consumption of the access point. As it is now, the firmware of the 3G Broadband module is updated externally from the development board to the module via USB and the update is only installed by the module if it is signed by Ericsson. But executing a program on the same processor as the module's firmware introduces a code separation problem; the ported application cannot interfere with the execution of the firmware.

Considering the hardware, the access point should in the future be equipped with an RFID- or magnetic card reader to improve the security. At the moment, knowing the user id and pin code is enough to get access. If the user is also given a physical badge or a card, it is not enough to just steal someone's code to get access. Having a physical object also makes it easier to discover a loss than to discover that someone knows the pin code. Preferably, the access point can also be equipped with an LCD to easily inform the user about the access point status.

The access points are only connected to the server when data is exchanged to easen the burden on the server and at the same time support more access points. As it is implemented, the time when the access point should connect next time is sent from the server. Every time an access point connects, it gets a new time that is only based on current time plus the interval given to that access point. To get a better distribution over time, the server should be able to schedule every call-in the access points do instead of just using an interval. Additionally, the server should implement a separate scheduling for the firmware updates. As it is now, the server only gives the access point an absolute time, which is the same time as the next call-in. The scheduling of the updates should be done in such a way that they do not occur at the same time and the time of the update could be adjusted such that it occurs when the access point is used the least.

Bibliography

[1] Nationalencyklopedi. mobiltelefoni. Swedish. Available at URL:
www.ne.se/lang/mobiltelefoni
Accessed January 7, 2012.

[2] Telia.se - Telias och Tele2s gemensamma turbo-3G nät bäst i test. Swedish.
Available at URL:
http://nyheter.telia.se/2176/telias-och-tele2s-gemensamma-turbo-3g-nat-bast-i-test/
Accessed January 7, 2012

[3] Telia.se - Täckningskarta. Swedish. Available at URL:
http://www.telia.se/privat/mobilt-bredband/merom/tackningskarta/tackningskarta.page
Accessed January 7, 2012

[4] Apple.com - iPad Technical Specifications. Available at URL:
http://www.apple.com/ipad/specs/
Accessed January 7, 2012

[5] Li Ping, Liao Jingsheng, Hu Chao, Yin Yong, Meng Max Q.-H. Design of Medical Remote Monitoring System Base on Embedded Linux. Proceeding of the IEEE International Conference on Information and Automation. June 2011; Shenzhen, China; pp. 590-594.

[6] Cai Li, Zhang Wenya, Li En, Liang Zize, Hou Zeng-Guang, Tan Min. Design and Implementation of a CDMA-Based Remote Monitoring and Controlling System. SICE Annual Conference 2007. Sept. 17-20, 2007; Kagawa, Japan; pp. 2445-2450.

[7] Daradimos Ilias, Papadopoulos Konstantinos, Stavrakas Ilias, Kaitsa Maria, Kontogiannis Theophanis, Triantis Dimos. A Physical Access Control System that utilizes existing networking and computer infrastructure. EUROCON 2007: The International Conference on "Computer as a Tool". Sept. 9-12, 2007; Warsaw, Poland. pp. 501-504.

[8] Popa M., Popa S., Marcu M.. A Distributed Smart Card Based Access Control System. SISY 2010: IEEE 8th International Symposium on Intelligent Systems and Informatics. Sept. 10-11, 2010. Subotica, Serbia. pp. 341-346.

[9] Hwang Il-Kyu, Baek Jin-Wook. Wireless Access Monitoring and Control System based on Digital Door Lock. IEEE Transactions on Consumer Electronics, Vol. 53, No. 4. Nov, 2007; pp. 1724-1730.

[10] ZigBee - Specifications. Available at URL:
http://www.zigbee.org/Specifications/ZigBee/FAQ.aspx
Accessed January 7, 2012.

[11] Bhargav Ajay. Martix Keypad Interfacing with Microcontrollers: Introduction. Available at URL:
http://www.8051projects.net/keypad-interfacing/introduction.php
Accessed January 14, 2012.

[12] Kalinsky David, Kalinsky Roee. Introduction to Serial Peripheral Interface. Available at URL:
http://www.eetimes.com/discussion/beginner-s-corner/4023908/Introduction-to-Serial-Peripheral-Interface
Accessed January 14, 2012.

[13] Bies, Lammert. Hayes modem AT command set. Available at URL:
http://www.lammertbies.nl/comm/info/hayes-at-commands.html
Accessed January 22, 2012

[14] AT Command Manual. *Ericsson*. 2009.

[15] BuildRoot - About. Available at URL:
http://buildroot.uclibc.org/about.html
Accessed January 14, 2012.

[16] Introduction to cross-compiling for Linux. Available at URL:
http://landley.net/writing/docs/cross-compiling.html
Accessed January 14, 2012.

[17] Model/View Programming. Available at URL:
http://developer.qt.nokia.com/doc/qt-4.8/model-view-programming.html
Accessed January 21, 2012.

[18] Betke, Klaus. The NMEA 0183 Protocol. May 2000. Available at URL:
www.tronico.fi/OH6NT/docs/NMEA0183.pdf
Accessed January 21, 2012.

[19] Protocol Buffers - Developers Guide. Available at URL:
http://code.google.com/intl/sv-SE/apis/protocolbuffers/docs/overview.html
Accessed October, 2011.

[20] Mochel Patrick, The sysfs Filesystem. Available at URL:
www.kernel.org/pub/linux/kernel/people/mochel/doc/papers/ols-2005/mochel.pdf
Accessed January 21, 2012

[21] Apache.org - Thrift. Available at URL: http://thrift.apache.org
Accessed October, 2011.

[22] BeagleBoard-xM System Rev C Reference Manual Revision 1.0. *beagleboard.org*. 2010.

[23] Datasheet for MCP23S17. *Microship Technology Inc*. 2007.

[24] eLinux.org - BeagleBoard/SPI/Patch-2.6.32. Available at URL:
http://elinux.org/BeagleBoard/SPI/Patch-2.6.32
Accessed November 7, 2011.

[25] SD/MMC format for OMAP3 boot. Available at URL:
http://processors.wiki.ti.com/index.php?title=MMC_Boot_Format
Accessed October 25, 2011.

[26] Boot Sequence. Available at URL:
http://processors.wiki.ti.com/index.php/Boot_Sequence
Accessed October 25, 2011.

[27] TI AM/OMAP Booting and Flash Recovery. Available at URL:
http://community.qnx.com/sf/wiki/do/viewPage/projects.bsp/wiki/AM_OMAP_boot_resources
Accessed October 25, 2011.

[28] Protocol Buffers - Techniques. Available at URL:
http://code.google.com/intl/sv-SE/apis/protocolbuffers/docs/techniques.html
Accessed October, 2011.

[29] Belokosztolszki, András. Role-based access control policy administration. 2004.

[30] Sandhu Ravi S., Coyne Edward J., Feinstein Hal L.. Role-Based Access Control Models. Computer, Vol. 29 Issue 2. Feb, 1996; pp. 38-47.

Appendix A

AT Commands

- **AT+CPIN?** - This command checks whether the SIM card is protected by a PIN code or not. The return value says READY if it is not protected and otherwise what type of code that needs to be input; PIN, PUK etc.
- **AT+CPIN** - This command is used to unlock the SIM card by sending the PIN or PUK code to it.
- **AT+CFUN** - This command is used to turn on the module. There are different modes for the module: fully functional mode, energy-saving mode etc. The mode is given as a parameter to the command.
- **AT+CGDCONT** - Used to configure the Access Point Name (APN) of the module.
- **AT*ENAP** - The USB Ethernet Emulation is enabled or disabled with this command.
- **AT*ENAP?** - This command is used to check the status of the connection. Which is disconnected, connected or connection in progress.
- **AT*E2GPSCTL** - Command used for enabling or disabling the GPS receiver. As a parameter the command is given an interval for how often gps positions will be returned.
- **AT*E2GPSNPS** - With this command a virtual USB COM port is activated for GPS (NMEA) output.
- **AT*EEVINFO** - A call to this command returns information about the module.
- **AT*CGPADDR** - This command returns the IP address of the module.