Institutionen för datavetenskap
Department of Computer and Information Science

Final Thesis

Lightweight M2M Solution on Android Platform

by

Magnus Gustafsson

LIU-IDA/LITH-EX-A--11/031--SE

2011-09-14
Final thesis

Lightweight M2M Solution on Android Platform

by

Magnus Gustafsson

LIU-IDA/LITH-EX-A–11/031–SE

2011-09-14

Supervisor: Mikael Asplund (IDA)
Daniel Pettersson (Attentec AB)

Examiner: Simin Nadjm-Tehrani
Abstract

Machine-to-machine communication (M2M) is a generic term for technologies dealing with autonomous communication between machines. For the last 10 years a wide range of business areas utilize a variety of different M2M solutions for remote management of equipment. Common for almost all of those solutions is that they are expensive and require the infrastructure to be adapted to them. They are also usually built out of several different systems working together and thus there are several systems that require maintenance.

This thesis investigates the possibility to develop a lightweight alternative to existing M2M solutions using only common devices and protocols. Lightweight here means that the system should be flexible, have a low cost for set-up and operation and that both ends should be mobile. By developing a lightweight M2M architecture the technology may become available in new business areas and new types of services may arise.

In the thesis a prototype is implemented. The purpose of the prototype is to practically verify whether a lightweight M2M solution is possible to develop in this manner. The solution uses the Android platform for backend and user interface and a Cinterion TC65T as slave device to which the sensors can be connected. The implemented system is limited in terms of security and performance but still acts as a proof of concept for this kind of M2M solution.
Acknowledgements

This master thesis is performed at Attentec AB in Linköping during March 2011 to September 2011. It has been examined by Institutionen för Datavetenskap (IDA), Linköpings Universitet.

I would like to thank my supervisor at Attentec AB, Daniel Pettersson, for his many ideas and Lars Lundberg for his commitment and aid in my work. I would also like to thank my examiner, Simin Nadjm-Tehrani, and my supervisor at IDA, Mikael Asplund, for their help with reviewing and improving this report.
# Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM</td>
<td>Advanced RISC Machine.</td>
</tr>
<tr>
<td>ARM7</td>
<td>A generation of the ARM processor design.</td>
</tr>
<tr>
<td>AT Commands</td>
<td>Attention Commands. Command set often used by modems and phones.</td>
</tr>
<tr>
<td>DIN rail</td>
<td>Metal rail used to mount industrial equipment.</td>
</tr>
<tr>
<td>GPIO</td>
<td>General Purpose Input/Output. May be Analog/Digital Converters, I²C, SPI, etc.</td>
</tr>
<tr>
<td>GPRS</td>
<td>General Packet Radio Service. Set of standards allowing data packets to be sent over GSM. GPRS is also known as 2.5G.</td>
</tr>
<tr>
<td>GSM</td>
<td>Global System for Mobile Communications. Set of standards for the second generation mobile network. GSM is also known as 2G.</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface.</td>
</tr>
<tr>
<td>HTTP</td>
<td>HyperText Transfer Protocol. Communication protocol used mainly to transfer web pages.</td>
</tr>
<tr>
<td>HTTPS</td>
<td>HyperText Transfer Protocol Secure. A encrypted version of HTTP.</td>
</tr>
<tr>
<td>I²C</td>
<td>Inter-Integrated Circuit. A communication interface.</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronics Engineers.</td>
</tr>
<tr>
<td>IMEI</td>
<td>International Mobile Equipment Identity. Unique number for each mobile device.</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol.</td>
</tr>
<tr>
<td><strong>Glossary</strong></td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td></td>
</tr>
<tr>
<td><strong>LAN</strong></td>
<td>Local Area Network.</td>
</tr>
<tr>
<td><strong>M2M</strong></td>
<td>Machine-to-machine communication. Autonomic communication between machines. See Section 2.1 for details.</td>
</tr>
<tr>
<td><strong>MAN</strong></td>
<td>Metropolitan Area Network</td>
</tr>
<tr>
<td><strong>Master</strong></td>
<td>Server and user interface for the system. The Android tablet in the implementation, see Section 3.4.</td>
</tr>
<tr>
<td><strong>RISC</strong></td>
<td>Reduced Instruction Set Computer.</td>
</tr>
<tr>
<td><strong>RS232</strong></td>
<td>Standard communications port. Also known as COM-port.</td>
</tr>
<tr>
<td><strong>SDK</strong></td>
<td>Software Development Kit.</td>
</tr>
<tr>
<td><strong>Slave</strong></td>
<td>Device physically connected to the monitored sensors. The Cinterion TC65T device in the implementation, see Section 3.5.</td>
</tr>
<tr>
<td><strong>SPI</strong></td>
<td>Serial Peripheral Interface. A communication interface.</td>
</tr>
<tr>
<td><strong>SVN</strong></td>
<td>Subversion. Software for revision control and backup.</td>
</tr>
<tr>
<td><strong>TCP</strong></td>
<td>Transmission Control Protocol.</td>
</tr>
<tr>
<td><strong>UI</strong></td>
<td>User Interface.</td>
</tr>
<tr>
<td><strong>Update session</strong></td>
<td>Event where the slave sends all sensor values accumulated since the last update session to the master (local concept only applicable for this report).</td>
</tr>
<tr>
<td><strong>WLAN</strong></td>
<td>Wireless Local Area Network.</td>
</tr>
<tr>
<td><strong>Zig-Bee</strong></td>
<td>Low power, low cost wireless network standard.</td>
</tr>
</tbody>
</table>
Contents

1 Introduction ................................................. 1
  1.1 Background ............................................. 1
  1.2 Purpose and Objective ................................ 2
  1.3 Limitations ............................................. 3
  1.4 Intended Audience ..................................... 4
  1.5 Thesis Overview ....................................... 4

2 Technologies .................................................. 5
  2.1 Machine-to-machine Communication ...................... 5
    2.1.1 Definition ......................................... 5
    2.1.2 Background ........................................ 6
    2.1.3 Current M2M Applications ......................... 7
  2.2 Communication Technologies .................. 7
    2.2.1 Network Standards ................................. 8
    2.2.2 Communication Protocol ......................... 8
  2.3 Cinterion TC65T ........................................ 8
    2.3.1 Cinterion Wireless Modules GmbH ................ 9
    2.3.2 Device Information ................................ 9
    2.3.3 Internal Structure ................................. 9
  2.4 Java Micro Edition ................................... 10
  2.5 Android .................................................. 11
    2.5.1 History ............................................. 11
    2.5.2 Android for the Application Programmer .......... 12

3 Android-based M2M platform .................. 15
  3.1 Requirements ......................................... 15
    3.1.1 Structure of Requirements ....................... 15
    3.1.2 Functional Requirements .......................... 16
    3.1.3 Non-functional Requirements ...................... 17
  3.2 Major Design Choices ................................. 17
    3.2.1 Android Platform ................................ 17
    3.2.2 Terminal Module .................................. 17
    3.2.3 Communication Interface ......................... 18
  3.3 System Overview ....................................... 19
3.3.1 Communication ........................................ 19
3.4 Master Application ....................................... 20
  3.4.1 User Interface Activities .......................... 22
  3.4.2 Connection Service ................................. 25
  3.4.3 SQLite Database .................................. 25
  3.4.4 Using the Master Application ...................... 26
3.5 Slave Application ....................................... 27
  3.5.1 Structure of the Slave Application .............. 32
  3.5.2 Using the Slave Application ...................... 33

4 Evaluation ................................................. 35
  4.1 Requirement Verification ............................. 35
  4.2 System Viability ...................................... 38
  4.3 Reliability ............................................ 39
  4.4 Security ............................................... 39
  4.5 Performance .......................................... 39
    4.5.1 Performance Tests ............................... 40
  4.6 Time-outs ............................................. 43

5 Summary and Conclusions ................................ 45
  5.1 Future Work .......................................... 46
List of Tables

3.1 Requirements priority. ................................. 15
3.2 Functional requirements. ............................ 16
3.3 Non-functional requirements. ........................ 17
3.4 Requirements affecting the communication interface. 18
3.5 Fields in the database table for slave devices. 25
3.6 Fields in the database table for sensor values. 25
3.7 Value types for sensor values. ........................ 26

4.1 Functional requirements verification, part 1 of 2. 36
4.2 Functional requirements verification, part 2 of 2. 37
4.3 Non-functional requirements verification. .......... 38
4.4 Theoretical number of devices the system can handle. 42
List of Figures

1.1 Original M2M solution ........................................ 2
1.2 Overview of the system ...................................... 3

2.1 M2M Hierarchy .................................................. 6
2.2 Cinterion TC65T .................................................. 10
2.3 Internal structure of the Cinterion TC65T ................. 11

3.1 Architecture of the implemented system .................... 20
3.2 Communication between user, master and slave ............ 21
3.3 Hierarchy of the master application ......................... 22
3.4 Screenshots of M2MMasterActivity and ViewSlaveActivity ........................................ 23
3.5 Screenshots of ViewSensorActivity and CreateSensorActivity ........................................ 24
3.6 Screenshot of the notification icon ......................... 26
3.7 Notifications in extended mode ............................... 27
3.8 Using LightweightM2M, steps 1 to 4 of 7 .................... 28
3.9 Using LightweightM2M, steps 5 to 7 of 7 .................... 29
3.10 Timeline for a typical use scenario ....................... 30
3.11 Flow chart of the slave application ....................... 31

4.1 Update session time, multiple sensors ........................ 41
4.2 Update session time, one sensor ............................ 41
4.3 Update session time, 40-value sensors ........................ 42
Chapter 1

Introduction

This chapter is the introduction to a master thesis project (30 credit points) final report examined at the Department of Computer and Information Science (IDA) at Linköping University. The thesis is the final part of a 5 year degree leading to a Master of Computer Science and Engineering.

The thesis work has been performed at Attentec AB in Linköping. Attentec AB is a software consultancy company with current focus on Internet, mobility and web. For further information about Attentec AB visit www.attentec.se.

1.1 Background

Machine-to-machine communication (M2M) is a generic term for technologies regarding autonomous communication between machines. It is used to monitor and maintain remote equipment.

Commercial drivers are that remote management:

- Is more cost-efficient than management on location.
- Enables better availability of the equipment.
- Enables introduction of new services and functionality.

The M2M solution that can be seen in Figure 1.1 was a proposal for a customer for Attentec AB. From this structure an idea emerged. The idea was that it may be possible to remove the web server and communicate directly between the mobile device and the GSM module. This would allow for a more simple solution that has a lower hardware and installation cost. From the idea this master thesis was formulated to verify whether such a solution was feasible.

Currently there are a lot of different M2M solutions on the market. Most of them are very expensive to acquire due to specialized hardware and customer specific software. The installation process is also often complex and
CHAPTER 1. INTRODUCTION

Figure 1.1: Original M2M solution. A PLC (Programmable Logic Controller) controls a set of valves and collects data from a set of sensors. The PLC communicates with a web server via a GSM Module. The web server then communicates with both a web interface as well as a mobile device for representing the values of the sensors and controlling the valves.

requires the infrastructure to be adopted to the system. This reduces the target group to large companies, and companies which heavily rely on M2M to operate. By building more cost-efficient M2M architectures and implementations new types of services may become available for new tiers of equipment in new types of businesses. A further investigation of the concept of M2M can be found later in the report (in Section 2.1).

1.2 Purpose and Objective

By making M2M solutions more accessible with lower costs the amount of feasible applications using the technology will increase. Reducing the cost also expands the target group for M2M solutions since it will be less of an investment for the company. This will make more business areas able to utilize M2M and thus, in time, increase the need for the technology.

This thesis examines the possibility to develop a lightweight alternative to existing M2M solutions using only common hardware devices and protocols. Here lightweight means that:

- It should have low cost for set-up and operation.
1.3. LIMITATIONS

- It should be flexible so that the same core system may be used in several different application areas, with only minor adjustments.

- Both master and slave devices could potentially be mobile.

To accomplish this description of a lightweight system an Android tablet will be used as both the back-end and user interface. The hardware for the slave device will be a Cinterion TC65T (see Section 2.3 for details). This will satisfy all three criteria for a lightweight system. The choice of using these hardware devices was made before the thesis begun. An overview of the system can be seen in Figure 1.2.

![Figure 1.2: Overview of the system. The Cinterion TC65T devices (slaves) collects data from the connected sensors and forwards it to the Android tablet (master). This allows the user to manage the sensors remotely through the tablet.](image)

1.3 Limitations

Since this thesis does not focus on a specific use for M2M, application specific design choices such as user interface, parameters and sensor-specific configuration is not considered. Energy consumption will also be of minor importance for the implementation in this thesis. Performance and security are aspects that have been considered to some extent but not as a prime concern in the design and development stages.
1.4 Intended Audience

This report is intended for an audience familiar with software development using an object oriented language. Readers with a limited experience in software development may have trouble understanding the more complex parts of Chapter 2 (Technologies) and Chapter 3 (Android-based M2M platform). Anyone not familiar with software development will still be able to understand the main concepts of the thesis.

1.5 Thesis Overview

There are a total of five chapters in this thesis.

Chapter 2 introduces the reader to the different technologies used in this thesis. First off is M2M (Machine-to-machine communication), then the communication technologies used in the thesis. After that the used hardware platform (Cinterion TC65T), then Java ME (Micro Edition), and at last the Android platform.

Chapter 3 deals with the Android-based M2M platform implemented in the thesis. First the system requirements are listed. After that the major design choices of the design is discussed followed by an overview of the system. This chapter also describes the architecture of the implemented system and how both applications are used.

Chapter 4 contains the evaluation of the system described in Chapter 3. The evaluation is based on several aspects such as reliability, security and performance. An explanation of how the requirements are tested and which of them that are fulfilled is also included.

Chapter 5 summarizes the thesis. It also discusses this thesis as well as the future for this kind of M2M solution.
Chapter 2

Technologies

2.1 Machine-to-machine Communication

This section aims to clarify the concept of machine-to-machine communication (M2M).

2.1.1 Definition

M2M does not yet have a general definition, and therefore companies (and people) make up their own definitions which can be seen more as descriptions until one standard definition is in place. The following description is, in my opinion, the most accurate. It is formulated by a company named Numerex [1]:

"M2M communications consists of using a device (e.g., sensor, meter, etc.) to capture an "event" (e.g., temperature, inventory level, location, environment status, etc.), relayed through a network (e.g., wireless, wired or hybrid) to an application (software program), translating the captured event into meaningful information (e.g., there is a breach, corrosion requires attention, items need to be restocked, an accident has occurred, etc.)."

Institute of Electrical and Electronics Engineers (IEEE) 802 LAN/MAN Standards Committee has initiated a project for creating a standard for M2M [2]. This standard also includes definitions for commonly used concepts in the area. The latest revision (2011-06-30) for the M2M definition is as follows:

"Machine-to-Machine communication: This is information exchange between user devices through a base station, or between a device and a server in the core network through a Base Station that may be carried out without any human interaction."
The European Telecommunications Standards Institute (ETSI) has also
started a Technical Committee in 2008 to develop standards for M2M [3, 4].

Even though the definitions differ they are all referring to the same core
structure where information is passed between a simple device such as a
sensor and a more competent back-end system over a network. Commonly
the information is collected in two or more layers to reduce the cost of the
simple devices. A general structure can be seen in Figure 2.1. The back-end
system usually consists of a database, a user interface (like a web server)
and a module for logic, control and data gathering. The data collectors
collect data from connected sensors and forward it to the back-end system.
The sensors are usually connected to the data collectors using either some
short range wireless technology (WLAN, Bluetooth, Zig-Bee, etc.) or wired
connection (LAN, I²C, analog, etc.). Communication between the back-
end system and the data collectors are usually some long range wireless
connection like GPRS, 3G or GSM.

Another interesting use of the M2M technology is to connect several
different devices used in every-day life in a context-aware system. An
generic example of this has been implemented at a university in Zagreb, Croatia
[5]. Their implementation uses the Android operating system for the client
devices, but with the aim to extend support for other devices as well.

2.1.2 Background

Remote management of equipment has been around for a long time but the
concept of M2M was founded in late 1990s with the idea of utilizing base
stations instead of telephone lines for the communication. At that time
hardware components were too expensive and neither the technology nor
the infrastructure were stable enough for such use. Today the needed tech-
nologies are a lot more developed and costs for both hardware and mobile
subscriptions are at reasonable levels [4].

Mobile operators have a great interest in the development of M2M. The
market for phone subscriptions are getting saturated and machine communication is a new opportunity for mobile operators [4]. For example, in 2008 Telenor established a subsidiary called Telenor Connexion completely focusing on M2M [6].

2.1.3 Current M2M Applications

M2M is used in a lot of business areas to increase productivity and availability. There are essentially three common uses of the technology; production, monitoring and remote updates.

Production

Using M2M in production lets the manufacturers know when the construction machines needs maintenance and/or a refill of parts [7]. The requests may be directed to either a person or a robot for refill and usually to a person for maintenance. For this application M2M can also be used to increase or decrease the speed at which the machines are working. Most production areas use M2M for said purposes. This increases efficiency and productivity of the production and the quality of the product.

Monitoring

Remote monitoring reduces the cost of maintenance and management a lot since no person has to manually read the values from the devices or check if they are working [7]. The monitored devices are also often spread out in a large area and/or in places difficult to reach. Thus utilizing M2M for monitoring is highly desirable since it removes the travelling time needed to check the devices. In 2009 a law was passed in Sweden saying that the electricity meters in all housing has to be read on a monthly basis [8]. To make this feasible for electricity provider companies they use different M2M technologies to acquire the data from the meters.

Updates

The use of electrical billboards has increased a lot for advertising purposes [9]. This is desirable since they can be updated to show different advertisements based on time of day and/or day of the week. It also allows for instant updates of daily prices and offers.

2.2 Communication Technologies

This section gives a brief introduction in the communication protocols and standards used in this thesis.
2.2.1 Network Standards

The network standards describe the physical means for transferring messages. In this thesis both GPRS and 3G are used for communication. GPRS is used by the slave device (Cinterion TC65T) to access the internet. 3G is used in the master device for the same purpose. The reason why GPRS is used instead of 3G in the slave device is that Cinterion TC65T does not support 3G. In future versions of M2M-devices 3G will probably be used to increase performance and decrease latency.

GPRS

General Packet Radio Service (GPRS) is an extension to Global System for Mobile Communications (GSM) adding functionality for packet-switched data transfers [10]. GPRS is often called 2.5G.

3G

The third generation mobile telecommunications (3G) is a set of standards used for telecommunications. Compared to GPRS it offers significantly higher bandwidth.

2.2.2 Communication Protocol

For the implemented system described in Chapter 3 the TCP/IP model is used for communication.

TCP/IP

The Transmission Control Protocol (TCP) and the Internet Protocol (IP) are usually used together along with a set of other protocols for communication, forming a model. This model is often referred to as the TCP/IP model (or just TCP/IP) since these two protocols are the main protocols used. The model is based on four layers: Link Layer, Internet Layer, Transport Layer and Application Layer. Each layer will perform some actions on each data packet to send. For example the Internet Layer handles addressing, so it adds an IP header to each packet containing the IP address of the receiver among other things. Most platforms supporting network communication has built in support for the TCP/IP model which lets the application programmer focus on other aspects of the application.

2.3 Cinterion TC65T

This section describes the GSM module Cinterion TC65T used in the implementation.
2.3. CINTERION TC65T

2.3.1 Cinterion Wireless Modules GmbH

Cinterion Wireless Modules GmbH is a supplier of M2M-capable devices with headquarters in Germany. Cinterion was the first company to provide a M2M data module utilizing GSM in 1995. Since then they have been one of the leading companies in the business of M2M devices with several important innovations in the area. They have been the global market share leader in cellular M2M devices every year since 2003. In 2009 they had a global market share of 26%, where the second largest had 23% [11].

2.3.2 Device Information

The TC65T module (Figure 2.2) is a GPRS capable M2M device. The actual model name is “TC65”, and the ending "T" means that it is mounted in a terminal. For the implementation part of this thesis the TC65T module is used as slave device.

The key features of the TC65: [12]

- Quad-band technology (850/900/1800/1900 MHz).
- High-speed data transfer using GPRS (class 12).
- Java™ support, IMP-NG (see Section 2.4 for a description).
- Range of standard interfaces (I²C bus, SPI bus, analogue-digital converter (ADC), serial, audio, 10 GPIOs and a SIM card interface).
- ARM7 processor.
- Integrated TCP/IP stack.

2.3.3 Internal Structure

The TC65T has mainly three modules; a Modem, a Java Machine and a Recordstore database. View Figure 2.3 for an overview.

Modem

The Modem module handles communication with the attached SIM card, GPIOs and the GSM network. The user may interact with the modem module with AT commands through the RS232 (COM) port.

Java Machine

The Java Machine in the TC65T supports the Java ME profile IMP-NG (Information Module Profile - Next Generation, see Section 2.4 for details). This allows for a relatively low development time for applications compared to other devices utilizing C or Assembler for developing applications. AT
commands may be used to communicate with the Modem module. The Java Machine implementation also has abstraction layers for common communication channels like the RS232 port, Internet and GPIOs (General Purpose Inputs/Outputs).

**Recordstore Database**

The Recordstore database is a very limited database used for its minimal space requirement. It has an abstraction layer in the Java Machine that only supports storing arrays of bytes. This means that the application programmer has to perform the conversion between the internal structures and byte arrays.

### 2.4 Java Micro Edition

Java Micro Edition (ME) is a stripped down version of Java Standard Edition (SE). It was originally created by Sun Microsystems (later acquired by Oracle) to allow developers to make applications for small devices with limited memory, display and power capacity [13]. Because of the wide spectrum of applications for Java ME it is divided into two base configurations and several profiles.

The two base configurations of Java ME are *Connected Limited Device Configuration* (CLDC) which is intended for small limited devices and *Connected Device Configuration* (CDC) which is more versatile. *Mobile Infor-
2.5 ANDROID

Figure 2.3: Internal structure of the Cinterion TC65T. The Modem handles the SIM card and GSM/GPRS functionality. The Java Machine can run Java ME code and communicates with the modem through AT Commands.

Information Device Profile (MIDP) is an extended version of the CDLC used for developing applications for mobile communication devices. MIDP is divided into three versions where 1.0 is CDLC extended with support for MIDlet programming, GUI, persistent data storage, some network capability and security. MIDP 2.0 extends MIDP 1.0 with the most important additions being support for HTTPS, digital certificates and a small XML parser. Some features added in the 3.0 version are IPv6, multiple network interfaces and auto-launch of MIDlets.

The profile used in Cinterion TC65T (Section 2.3) is Information Module Profile - Next Generation (IMP-NG) which is a strict subset of MIDP 2.0 where all GUI components has been removed. This profiled is intended for small embedded network devices without display.

2.5 Android

The purpose of this section is to give the reader some basic understanding of the Android operating system.

Android is an operating system for smartphones and tablets. It is distributed under an open license and is free to use. Some standard applications like SMS, mail, maps, contacts and calendar are shipped with the operating system. Some manufacturers also include their own software in their devices. Application developers have access to the same APIs used by the standard applications. Due to this, each application have the same priority seen from the operating systems point of view.

2.5.1 History

With the purpose to advance the open standards for hand held devices the Open Handset Alliance (OHA) was formed in November 2007 by 34 compa-
cies, led by Google. Along with the formation, OHA released Android, an operating system targeting hand held devices. Android has been available since October 2008.

2.5.2 Android for the Application Programmer

Applications for Android is developed using the Java programming language. Instead of using the Java Virtual Machine (JVM) OHA created the Dalvik Virtual Machine which works almost identical as the JVM. A plugin for Eclipse [14] exists containing a lot of features to make developing for Android easier and more efficient. There are four application components for Android: *activity*, *service*, *content provider* and *broadcast receiver*. The following descriptions for each of them are copied from the Android developer web page [15].

**Activity**

An *activity* represents a single screen with a user interface. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. Although the activities work together to form a cohesive user experience in the email application, each one is independent of the others. As such, a different application can start any one of these activities (if the email application allows it). For example, a camera application can start the activity in the email application that composes new mail, in order for the user to share a picture.

**Service**

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

**Content provider**

A *content provider* manages a shared set of application data. You can store the data in the file system, an SQLite database, on the web, or any other persistent storage location your application can access. Through the content provider, other applications can query or even modify the data (if the content provider
2.5. ANDROID

allows it). For example, the Android system provides a content provider that manages the user’s contact information. As such, any application with the proper permissions can query part of the content provider (such as ContactsContract.Data) to read and write information about a particular person.

Content providers are also useful for reading and writing data that is private to your application and not shared.

Broadcast receiver

A broadcast receiver is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Applications can also initiate broadcasts—for example, to let other applications know that some data has been downloaded to the device and is available for them to use. Although broadcast receivers don’t display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs. More commonly, though, a broadcast receiver is just a "gateway" to other components and is intended to do a very minimal amount of work. For instance, it might initiate a service to perform some work based on the event.
Chapter 3

Android-based M2M platform

This chapter describes the design and implementation of the M2M system developed during this thesis. The system consists of a master device (top two layers in Figure 2.1) and a slave device (third layer in Figure 2.1).

3.1 Requirements

In this section all requirements on the implementation are listed. Since the implementation is a "Proof of Concept" the requirements are there to ensure the right concepts are proven by the implemented system.

3.1.1 Structure of Requirements

Each requirement has a unique ID, an explanation and a priority. The priority will follow Table 3.1.

<table>
<thead>
<tr>
<th>Priority</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Main goal of the project. Has to be done.</td>
</tr>
<tr>
<td>2</td>
<td>In the scope of the project but can be ignored if there is a risk for them to interfere with fulfilling priority 1 requirements (e.g. for time reasons).</td>
</tr>
<tr>
<td>3</td>
<td>Not in the scope of the project and will be done after all priority 1 and 2 requirements are fulfilled.</td>
</tr>
</tbody>
</table>

Table 3.1: Requirements priority.
### 3.1.2 Functional Requirements

All functional requirements for the implementation are listed in Table 3.2.

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The system shall be implemented using a Master-Slave architecture.</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>The master can send messages to the slave when connection is established.</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>The slave can send messages to the master when connection is established.</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Messages sent when connection is lost should be saved and sent when connection is reestablished.</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>The system shall be able to handle at least two slaves.</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>Slaves shall not contain master-specific software (IP address et cetera).</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>System shall be compatible with Android 2.2 or higher.</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>New slave shall be possible to add to the system without updating master software.</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>Communication between master and slave shall use GPRS.</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>Communication interface shall use IP.</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>The slaves shall support multiple connected sensors.</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>Restarting the master application should not remove monitored slaves.</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>The master application should have a simple user interface.</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>Slaves may be able to be updated remotely.</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>Slave parameters may be updated over the air.</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>Identified security issues shall be solved.</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>Communication interface may be compatible with &gt;2.5G networks.</td>
<td>2</td>
</tr>
<tr>
<td>18</td>
<td>The android based master application shall be replaceable with a PC based web server.</td>
<td>2</td>
</tr>
<tr>
<td>19</td>
<td>Slave parameters (sensor names, ranges, etc.) may be configurable via the server UI.</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>The master may be able to retrieve sensor values from other sources (e.g. internet).</td>
<td>3</td>
</tr>
<tr>
<td>21</td>
<td>The master architecture shall provide a developer friendly platform for developing logic controlled events.</td>
<td>3</td>
</tr>
<tr>
<td>22</td>
<td>The slave may support sensors over I²Cbus.</td>
<td>3</td>
</tr>
<tr>
<td>23</td>
<td>The master application may be replaceable with a Google based cloud service.</td>
<td>3</td>
</tr>
<tr>
<td>24</td>
<td>The system shall be able to handle roaming networks.</td>
<td>3</td>
</tr>
</tbody>
</table>
3.2. MAJOR DESIGN CHOICES

3.1.3 Non-functional Requirements

All non-functional requirements for the implementation are listed in Table 3.3.

Table 3.3: Non-functional requirements.

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>The slave software shall be developed using Eclipse.</td>
<td>1</td>
</tr>
<tr>
<td>26</td>
<td>All software shall be commented thoroughly and complex design documented separately.</td>
<td>1</td>
</tr>
<tr>
<td>27</td>
<td>All software developed shall be under revision control using SVN, all releases shall be tagged and repository kept on Attentec AB’s servers.</td>
<td>1</td>
</tr>
<tr>
<td>28</td>
<td>Possible security issues identified during development shall be documented.</td>
<td>1</td>
</tr>
</tbody>
</table>

3.2 Major Design Choices

In this section the major design choices are described. The implementation aims to fulfil all priority 1 requirements while still being considered lightweight as described in Section 1.2. Some priority 2 and 3 requirements will be fulfilled if they do not risk the completion of more important tasks.

3.2.1 Android Platform

Even though the choice to use the Android platform was taken before the thesis started it is worth considering if this is indeed the most appropriate choice for the system. The thesis is focused on a lightweight M2M solution with Android only as one of the possible options. Other options include iOS on an iPad/iPhone and Windows/UNIX on a PC (with a web server for remote control).

Using a smartphone platform for the master device both ensures mobility and allows the back-end and the user interface to be combined in one device. This greatly reduces the cost compared to using a stationary server with a web interface. It also requires less physical space and has less power consumption. The reason why Android was chosen over iOS is that it is more accessible for developers as it requires no developer license.

Since the programming language for Android is Java, the code for the server part of the master is easy to port to a PC if needed.

3.2.2 Terminal Module

To acquire a system fulfilling our description of lightweight, the terminal module need to be mobile and it needs to be as generic as possible in terms
of possible applications. It should also preferably not be expensive and it should be easy to develop to since that also lowers the cost. The Cinterion TC65T module (see Section 2.3) fulfills all of these criteria and since it is also one of the most popular platforms for developing M2M solutions it has been chosen as slave device in this thesis. Another reason for this choice is that the company at which the thesis work was done (Attentec AB) has a collaboration with a distribution company for Cinterion called Acte Solutions AB [16]. This made both the device and support for it easily accessible.

### 3.2.3 Communication Interface

For this implementation the communication protocol should be able to detect if messages are transferred properly, it should be a common protocol and it should be available on both platforms used. TCP/IP fulfills all of these criteria and since it is one of the most common protocols for data communication no other protocol is considered.

Since the communication between master and slave is a TCP/IP connection, using Java’s `Socket` class for handling the protocol is convenient since it manages the protocol. This lets the application programmer focus on other aspects of the application. To establish connection between the two devices in the system one of them has to run as TCP/IP server. The TCP/IP server waits for incoming connection from the TCP/IP client. The client device always has to initiate the connection. Either device may be TCP/IP server, and there are advantages and drawbacks with both setups. Low-cost subscription usually provide dynamic IP address and put the user behind a firewall. This firewall blocks incoming connections which means that it cannot be the TCP/IP server. The dynamic IP address aspect is also not preferred for the TCP/IP server since that would make the IP address able to change and the TCP/IP client has to connect to the new address. Requirements affecting the choice for communication interface are listed in Table 3.4.

#### Table 3.4: Requirements affecting the communication interface.

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>The master can send messages to the slave when connection is established.</td>
</tr>
<tr>
<td>3</td>
<td>The slave can send messages to the master when connection is established.</td>
</tr>
<tr>
<td>5</td>
<td>The system shall be able to handle at least two slaves.</td>
</tr>
<tr>
<td>6</td>
<td>Slaves shall not contain master-specific software (IP address et cetera).</td>
</tr>
</tbody>
</table>
3.3. SYSTEM OVERVIEW

Advantages of Having the Master Device as TCP/IP Server
- Only the master device need to have a more expensive subscription without the operator’s firewall, and preferably with static IP.
- The slaves may send sensor updates at any time (assuming both devices are connecting to the network).
- No need for server to poll for sensor updates.

Advantages of Having the Slave Device as TCP/IP Server
- The master may send commands at any time (assuming both devices are connecting to the network).
- Initial connection is a lot more simple since the user can enter the IP address to the slave device in the user interface.

Chosen Design
The slave device is the one usually sending updates and the best option is therefore to let it be the TCP/IP client. Using the master device as TCP/IP server is also more beneficial for the user because only the master device needs a more expensive subscription and the system is designed to allow several slaves to connect to the same master. The major drawback of using this design is that the slave needs to get the IP to the master in some way since the slave does not know the IP address to the master for the first connection. Some additional communication channel where the master can address the slave is needed. The TC65T has several wired inputs but since they are wired they do not fit the purpose. It also supports SMS which fits perfectly since it is a long-range wireless technology where the master device can address the slave device.

3.3 System Overview

The system consists of one master application and one or more slave application(s). An overview with one slave can be seen in Figure 3.1. More slaves may be added and all slaves communicates with the same connection service in the master application (the system is only tested with a maximum of two slaves).

Figure 3.2 shows how the interaction between user, master and slave is designed to work.

3.3.1 Communication

As with all communication systems the communication has to be reliable. Messages should not be lost at any point. This is a bit tricky in this system since both devices are mobile and may lose connection at any time.
Figure 3.1: Architecture of the implemented system. Sensor values are captured by the Sensor drivers and saved in the Recordstore. At each update session all saved sensor values are read by the Communication and Control and sent to the Connection Service using GPRS/IP and then stored in the SQLite database. The AT-Command Listener is used for capturing SMS-events and the UI Activities are used to interact with the user.

If connection is lost while a message is being sent both devices need to see this and act accordingly to avoid losing messages. The system is designed to catch all losses of connection and save messages until the next time the device should send. The effect of this is that connections initialized when connection is absent will be skipped and all values saved. Losing connection during a message will have the same effect as not having connection at all for the current message.

3.4 Master Application

The master application consists of three separate modules; a SQLite database, a section of user interface activities and a connection service. Most of the communication between the user interface activities and the connection ser-
3.4. MASTER APPLICATION

Communication between user, master and slave. When both devices are switched on, the slave is triggered by sending a SMS containing the IP address to the master. The slave then tries to connect to that IP address. If the connection is successful the master will notify the user that a new slave has connected and the user may enter configuration information for it. The configuration information is sent to the slave which then enters the update session loop. In the loop the slave sends all sensor values to the master, then receives configuration for new sensors, and if transmission is successful it removes all sent sensor values.
vice uses the database as channel. A hierarchy of the master application may be seen in Figure 3.3.

The device used while developing is Samsung Galaxy Tab which is a tablet running Android version 2.2.

**UI Activities**
- M2MMasterActivity
- ViewSlaveActivity
- CreateSlaveActivity
- ViewSensorActivity
- CreateSensorActivity
- ConnectionService
- SQLite database

![Hierarchy of the master application](image)

Figure 3.3: Hierarchy of the master application.

### 3.4.1 User Interface Activities

Since the implementation is a proof-of-concept the user interface will only support the required functional features and does not provide a high level of usability.

**M2MMasterActivity**

M2MMasterActivity is the entrance activity and will therefore be started when the application is started. When started, this activity also starts the Connection Service (see Section 3.4.2) and has menu options for stopping and restarting the Connection Service. The activity shows the IP address to the device it is running on since the user need to send that IP address to the slave device as a SMS. A list with all connected slaves is presented in the activity as well. If the user clicks on a slave the ViewSlaveActivity will be started. A screenshot of M2MMasterActivity may be seen in Figure 3.4a.

**ViewSlaveActivity**

When ViewSlaveActivity is started it requests the database row id of a slave. The slave description is fetched and all sensors associated with it are listed. Clicking on one of the sensors starts the ViewSensorActivity with current slave’s IMEI (International Mobile Equipment Identity) and current sensors
3.4. MASTER APPLICATION

name as parameters. A screenshot of ViewSlaveActivity may be seen in Figure 3.4b. The slave device represented in this screenshot has the name \textit{Slave Device} and has two connected sensors named \textit{Sensor 1} and \textit{Sensor 2}. The sensor type and update interval of each sensor may be seen as well. Clicking the menu button on the device brings up one menu option: \textit{Insert} which starts the CreateSensorActivity.

![M2MMasterActivity and ViewSlaveActivity](image)

(a) M2MMasterActivity. The top field is the name of the application and below it is the IP to the device. The third field from the top is a list of all connected slaves. Each entry in the list has name and IMEI of the slave.

(b) ViewSlaveActivity. The top field is the name of the slave and below it is a list of all sensors connected to that slave. Each sensor has its name to the left and the type of sensor and the update interval to the right.

Figure 3.4: Screenshots of M2MMasterActivity and ViewSlaveActivity.

**ViewSensorActivity**

ViewSensorActivity lists all values for the sensor with the input slave IMEI and sensor name. Each row in the list contains one value and the timestamp at which time the value was sampled in the slave. A screenshot of ViewSensor-Activity may be seen in Figure 3.5a. The name of the sensor represented is \textit{Sensor 2} and it has logged 10 sensor values.
CHAPTER 3. ANDROID-BASED M2M PLATFORM

Figure 3.5: Screenshots of ViewSensorActivity and CreateSensorActivity.

CreateSensorActivity

When creating a new sensor the user has to input information about it. The fields are name, update interval and sensor type. These input parameters are saved as an entry in the slave’s table in the SQLite database. The new entry gets a value in the "Value type"-field for showing that it is the header-entry for a new value. A screenshot may be seen in Figure 3.5b.

CreateSlaveActivity

The purpose of the CreateSlaveActivity is to set parameters for a new slave connected to the master. The parameters to set are name and update interval. These input parameters are saved as an entry in the table for sensor values in the SQLite database. The activity also shows a 60 seconds timer which begins when a new slave connects. The purpose of this timer is to allow the connection to time-out after that time and the slave does not have
3.4. MASTER APPLICATION

3.4.2 Connection Service

The connection service is used for communicating with the slave devices. It is initiated, and can be stopped, by the M2MMasterActivity. While running, the connection service holds a port open for incoming connection requests. When a slave connects, it handles all communication and performs different tasks depending on the input from the slave and the state of the master.

3.4.3 SQLite Database

Communication with the SQLite database in Android is done by SQL-queries. To make database access easier and the code more readable an abstraction class is present as a layer between the database and the application. This class is called SensorDbAdapter in the implementation and has functions for storing and fetching data from the database. The database has two tables; one for slave devices, (Table 3.5) and one for sensor values (Table 3.6). There are three types of sensor values listed in Table 3.7.

Table 3.5: Fields in the database table for slave devices.

<table>
<thead>
<tr>
<th>Name</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row ID</td>
<td>integer</td>
<td>The row id in the database. Set by the database when a new slave device is added.</td>
</tr>
<tr>
<td>IMEI</td>
<td>integer</td>
<td>The unique IMEI of the slave device. Used as identifier for it.</td>
</tr>
<tr>
<td>Name</td>
<td>text</td>
<td>The name of the slave device.</td>
</tr>
<tr>
<td>Update interval</td>
<td>long</td>
<td>The time interval between two subsequent update sessions (how often the slave device sends sensor updates to the master).</td>
</tr>
</tbody>
</table>

Table 3.6: Fields in the database table for sensor values.

<table>
<thead>
<tr>
<th>Name</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row ID</td>
<td>integer</td>
<td>The row id in the database. Set by the database when a new values is added.</td>
</tr>
<tr>
<td>Name</td>
<td>text</td>
<td>The name of the sensor.</td>
</tr>
<tr>
<td>Value</td>
<td>long</td>
<td>The value to log.</td>
</tr>
<tr>
<td>Timestamp</td>
<td>long</td>
<td>The time when the value was sampled.</td>
</tr>
<tr>
<td>IMEI</td>
<td>integer</td>
<td>The unique IMEI of the slave device containing the sensor.</td>
</tr>
<tr>
<td>Value type</td>
<td>integer</td>
<td>The type of the value (see Table 3.7 for details).</td>
</tr>
</tbody>
</table>
Table 3.7: Value types for sensor values.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular value</td>
<td>0</td>
<td>Indicates this is a normal sensor value.</td>
</tr>
<tr>
<td>Sensor header</td>
<td>1</td>
<td>The header for the sensor. Does not contain sensor value nor timestamp, but instead has the sensor type in the value field and the update interval for the sensor in the timestamp field.</td>
</tr>
<tr>
<td>New sensor header</td>
<td>2</td>
<td>Same as Sensor header except it has been created after the last update session and has therefore not been sent to the slave. Will be sent in the next update session and then changed into a sensor header.</td>
</tr>
</tbody>
</table>

3.4.4 Using the Master Application

When the application is started the connection service will be started and is represented by a red icon in the notification bar (see Figure 3.6). Pressing the notification in extended mode (see Figure 3.7a) starts the M2MMasterActivity.

When a slave device connects to the master then notification changes to alert the user that a slave has connected (Figure 3.7b). If the notification is pressed in this changed state the CreateSlaveActivity is started instead.

Figure 3.6: The middle icon is the icon representing the connection service.

The application is set up by performing the following 7 steps:

1. Start the ConnectionService by starting the main application. Only the ConnectionService need to be running and not the main application.

2. (Figure 3.7b) Start the CreateSlaveActivity by clicking the extended notification that triggers when a slave device connects.

3. (Figure 3.8a) Add information about the slave in the fields of the CreateSlaveActivity and press Confirm.

4. (Figure 3.8b) Click the newly created slave to start the ViewSlaveActivity.
3.5 Slave Application

The tasks of the slave application is to collect and log values from all connected and configured sensors and send them to the master. The data collection rate can be set individually for each sensor and is either decided by the implemented driver for that sensor type or by the user. All values logged are sent to the master with an interval decided by the user (the update interval set in the CreateSlaveActivity) and then deleted at the slave.
(a) CreateSlaveActivity. The number in the top (here 25 seconds) is a countdown timer showing how long the user has to enter the fields and confirm. Below it is the two fields for name and update interval, and lastly a confirm button.

(b) M2MMasterActivity with one slave device added. The top field is the name of the application and below it is the IP to the device. The third field from the top is a list of all connected slaves. Each entry in the list has name and IMEI of the slave.

(c) ViewSlaveActivity with no sensors added for the slave. Only the name of the slave is shown and the list of sensors is empty.

(d) CreateSensorActivity. Three fields for the user to enter information and a confirm button.

Figure 3.8: Using LightweightM2M, steps 1 to 4 of 7.
3.5. SLAVE APPLICATION

(a) ViewSlaveActivity with one sensor in the list of sensors.

(b) ViewSensorActivity with two sensor values.

(c) ViewSensorActivity with ten sensor values.

Figure 3.9: Using LightweightM2M, steps 5 to 7 of 7.
end. This means that only the values since the last successful transmission are stored locally at the slave. If no transmission is successful, all values will be stored in the slave. The storing of values when the connection is down is limited to the memory of the slave device. There are currently no error handling if the memory is not sufficient, but that should only happen in very rare cases. A time diagram example of this where no connection error occurs may be seen in Figure 3.10.

<table>
<thead>
<tr>
<th>Time</th>
<th>S1 (3s)</th>
<th>S2 (4s)</th>
<th>Slave database</th>
<th>Update session (10s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S1:0, S2:0</td>
<td>Send the sensors configuration to slave</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>S1:0, S2:0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
<td>S1:0, S2:0, S1:1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>2</td>
<td>S1:0, S2:0, S1:1, S2:1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>3</td>
<td>S1:0, S2:0, S1:1, S2:1, S3:1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td></td>
<td>S1:0, S2:0, S1:1, S2:1, S3:1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>3</td>
<td>S1:0, S2:0, S1:1, S2:1, S3:1, S4:1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>S1:0, S2:0, S1:1, S2:1, S3:1, S4:1, S5:1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td></td>
<td>S1:0, S2:0, S1:1, S2:1, S3:1, S4:1, S5:1, S6:1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>3</td>
<td>S1:0, S2:0, S1:1, S2:1, S3:1, S4:1, S5:1, S6:1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td>S1:0, S2:0, S1:1, S2:1, S3:1, S4:1, S5:1, S6:1</td>
<td>Send: S1:0, S2:0, S1:1, S2:1, S3:1, S4:1, S5:1, S6:1</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>3</td>
<td>S1:4, S2:3</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td>S1:4, S2:3</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>5</td>
<td>S1:4, S2:3, S1:5</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>4</td>
<td>S1:4, S2:3, S1:5, S2:4</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td>S1:4, S2:3, S1:5, S2:4</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td>S1:4, S2:3, S1:5, S2:4</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td></td>
<td>S1:4, S2:3, S1:5, S2:4, S1:6</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td>S1:4, S2:3, S1:5, S2:4, S1:6</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td></td>
<td>S2:5</td>
<td>Send: S1:4, S2:3, S1:5, S2:4, S1:6</td>
</tr>
</tbody>
</table>

Figure 3.10: Timeline for a typical use scenario. S1 and S2 are sensors and for demonstration purposes they are represented by counters, increasing their value each time they are read.

At start up, the application fetches configuration information (such as IP address to the master) from the Recordstore database. The application then enters the *update session loop* which is the state where the application gathers data from the sensors and sends it to the master at a given interval. If no configuration information is found in the Recordstore database the application waits for a SMS containing a valid IP address. When such a SMS is received it connects to that IP address (which should be the address of the master device) to fetch configuration and if configuration information is received it enters the update session loop. A received SMS containing the text "reset" will reset the application, deleting all records in the database. The application will then wait for a SMS containing an IP address again. A flow chart of the events may be seen in Figure 3.11.
3.5. SLAVE APPLICATION

Figure 3.11: Flow chart of the slave application. When the application is started it checks for configuration in the persistent memory. If it has a configuration it enters the update session loop. If no configuration is present it waits for a SMS containing a valid IP address. When such a SMS is received it connects to the given IP address to get configuration from that master device and then enters the update session loop. A SMS containing ”reset” when in the update session loop will cancel the loop, reset all configuration and the device will wait for a new SMS containing an IP address.
3.5.1 Structure of the Slave Application

The classes in the slave application are described here. The architecture of the system can be seen in Figure 3.1.

CommunicationAndControl

This is the entry class for the slave application. It handles the flow of events in the application and all GPRS communication with the master. The CommunicationAndControl class extends MIDlet which lets the operating system control the application through some standard methods that have to be present, namely startApp(), destroyApp() and pauseApp(). The startApp() method is called when the operating system starts the application, and likewise the destroyApp() is called when the application is shut down. These methods contain code for initiating the application and cleaning up for shut down, respectively. The pauseApp() method is not used in this implementation.

When called, the startApp() method creates a new instance of the CommunicationAndControl class and starts it in a new thread. This is possible since the class implements the Runnable interface which is a standard Java interface supplying the run() method. This enables the class to be started, performing the run() method either in the same thread as the caller or in a new thread. The reason for starting the CommunicationAndControl class in a new thread is to not steal the calling thread from the operating system. By doing this the operating system is still able to call the other standard methods if necessary.

The class also implements the BearerControlListener interface which gives methods for observing the GPRS state of the device.

ATCommandListener

AT Commands are used to access features of the Cinterion TC65 not supported by the Java ME implementation. In this application AT Commands are used to handle the SMS functionality.

RecordStoreManager

Java ME has a class used for persistent storage called RecordStore. The RecordStore can only handle byte[] and therefore an abstraction layer above it is useful. The RecordStoreManager is that abstraction layer and provides store and fetch functions useful for the slave application.

AbstractSensor

The AbstractSensor class is a class helping the developing of other sensor driver classes. New sensor classes should inherit from it and only override the method for fetching data from the physical sensor. The superclass is
3.5. SLAVE APPLICATION

integrated in the application, so new subclasses of it are treated in the same way and no modifications in other classes are needed.

3.5.2 Using the Slave Application

Since the application is running in an embedded system without any external way of interacting with it except for some standard communication ports the slave application has no interfaces for direct interaction. All interaction is through the master device. There are two communication channels; SMS and IP. SMS is used to receive the IP address to the master and to reset the device. These two commands are the only two available. Interaction using IP is done by the master device. The user can only interact with the master device which then communicates with the slave to perform the requested actions.
CHAPTER 3. ANDROID-BASED M2M PLATFORM
Chapter 4

Evaluation

In this chapter the implementation is evaluated.

4.1 Requirement Verification

To see if the implemented system matches the intended requirements we revisit each requirement from Chapter 3 in Table 4.1, Table 4.2 and Table 4.3.

The following list contains the tests for each requirement. Each test is initiated by starting both devices and then sending a SMS containing the master’s IP address to the slave. The slave then connects to the master and dummy values for each configuration field is entered, along with a dummy sensor. The system in this state is here called \textit{running}.

- \textbf{Test 1 (requirement 2, 3, 15 and 19)}: Verify that the master application gets updates from the slave device while the system is running.

- \textbf{Test 2 (requirement 4)}: Tested by shutting down the \textit{Connection-Service} in the master while the system is running. This makes the slave device unable to connect to the master since it does not accept incoming connections. Restarting the \textit{ConnectionService} lets the devices communicate again. Verify that all expected values are present in the master after one successful update session.

- \textbf{Test 3 (requirement 5, 8)}: When the system is running, a SMS with the master’s IP is sent to a second slave device. The rest of the initiation process is done for this second device as well. Verify that all expected values are present in the master.

- \textbf{Test 4 (requirement 6)}: The system is initiated with another device as master device. Verify that the system behaves as expected.
Table 4.1: Functional requirements verification, part 1 of 2.

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Priority</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The system shall be implemented using a Master-Slave architecture.</td>
<td>1</td>
<td>Fulfilled by the chosen design. Yes</td>
</tr>
<tr>
<td>2</td>
<td>The master can send messages to the slave when connection is established.</td>
<td>1</td>
<td>Tested and working. Yes</td>
</tr>
<tr>
<td>3</td>
<td>The slave can send messages to the master when connection is established.</td>
<td>1</td>
<td>Tested and working. Yes</td>
</tr>
<tr>
<td>4</td>
<td>Messages sent when connection is lost should be saved and sent when connection is restored.</td>
<td>1</td>
<td>Tested and working. Yes</td>
</tr>
<tr>
<td>5</td>
<td>The system shall be able to handle at least two slaves.</td>
<td>1</td>
<td>Tested with 1 and 2 slaves and working. Yes</td>
</tr>
<tr>
<td>6</td>
<td>Slaves shall not contain master-specific software (IP address et cetera).</td>
<td>1</td>
<td>A slave can be connected to any master. Tested with two different master devices and should work with any other in theory. Yes</td>
</tr>
<tr>
<td>7</td>
<td>System shall be compatible with Android 2.2 or higher.</td>
<td>1</td>
<td>System is designed for Android 2.1 and tested with Android 2.1 and 2.2. The system should work with later versions of Android as well since Android is backwards compatible, this is not tested though. Yes</td>
</tr>
<tr>
<td>8</td>
<td>New slave shall be possible to add to the system without updating master software.</td>
<td>1</td>
<td>Tested and working. Yes</td>
</tr>
<tr>
<td>9</td>
<td>Communication between master and slave shall use GPRS.</td>
<td>1</td>
<td>Slaves use GPRS for communication. Yes</td>
</tr>
<tr>
<td>10</td>
<td>Communication interface shall use IP.</td>
<td>1</td>
<td>IP is the used protocol. Yes</td>
</tr>
<tr>
<td>11</td>
<td>The slaves shall support multiple connected sensors.</td>
<td>1</td>
<td>Tested with 0 to 8 virtual sensors and working. Yes</td>
</tr>
<tr>
<td>12</td>
<td>Restarting the master application should not remove monitored slaves.</td>
<td>2</td>
<td>The master application uses the SQLite database in Android which is persistent so a restart will not remove data. Tested and working. Yes</td>
</tr>
</tbody>
</table>

36
### 4.1. REQUIREMENT VERIFICATION

Table 4.2: Functional requirements verification, part 2 of 2.

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Priority</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>The master application should have a simple user interface.</td>
<td>2</td>
<td>A simple user interface supporting the functionality is present.</td>
</tr>
<tr>
<td>14</td>
<td>Slaves may be able to be updated remotely.</td>
<td>2</td>
<td>The slave software cannot be updated remotely in the current version.</td>
</tr>
<tr>
<td>15</td>
<td>Slave parameters may be updated over the air.</td>
<td>2</td>
<td>IP is sent as SMS. Slave name, update interval and sensor parameters are set from the master device using GPRS.</td>
</tr>
<tr>
<td>16</td>
<td>Identified security issues shall be solved.</td>
<td>2</td>
<td>Security has not been considered during development.</td>
</tr>
<tr>
<td>17</td>
<td>Communication interface may be compatible with &gt;2.5G networks.</td>
<td>2</td>
<td>Not tested due to lack of compatible hardware. Should work in theory since Java’s abstraction layers for IP-communication is used.</td>
</tr>
<tr>
<td>18</td>
<td>The android based master application shall be replaceable with a PC based web server.</td>
<td>2</td>
<td>A new master application using the same communication interface for communicating with the slave(s) may be implemented and should work. Not tested.</td>
</tr>
<tr>
<td>19</td>
<td>Slave parameters (sensor names, ranges, etc.) may be configurable via the server UI.</td>
<td>3</td>
<td>Tested and working.</td>
</tr>
<tr>
<td>20</td>
<td>The master may be able to retrieve sensor values from other sources (e.g. internet).</td>
<td>3</td>
<td>Not implemented.</td>
</tr>
<tr>
<td>21</td>
<td>The master architecture shall provide a developer friendly platform for developing logic controlled events.</td>
<td>3</td>
<td>Not implemented.</td>
</tr>
<tr>
<td>22</td>
<td>The slave may support sensors over I²Cbus.</td>
<td>3</td>
<td>Should work but not tested due to lack of hardware. Requires a new sensor driver to be added in the slave.</td>
</tr>
<tr>
<td>23</td>
<td>The master application may be replaceable with a Google based cloud service.</td>
<td>3</td>
<td>No support implemented.</td>
</tr>
<tr>
<td>24</td>
<td>The system shall be able to handle roaming networks.</td>
<td>3</td>
<td>Roaming networks are handled in lower abstraction layers in the devices and should not affect the implemented system. Not tested.</td>
</tr>
</tbody>
</table>
CHAPTER 4. EVALUATION

• **Test 5 (requirement 11):** More dummy sensors are added, one at a time up to 8 sensors, when the system is running. Verify that all values for all sensors are present in the master.

• **Test 6 (requirement 12):** Restarting the device running the master application when the system is running. Verify that all sensor values captured during the restart time is stored in the slave device and transferred during the following successful update session.

Requirement 1, 7, 9, 10 and 13 are fulfilled but not connected to a specific test. Their description is enough for verification. Requirements that are not fulfilled are not tested since the software does not intend to support any of them.

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>The slave software shall be developed using Eclipse.</td>
</tr>
<tr>
<td>26</td>
<td>All software shall be commented thoroughly and complex design documented separately.</td>
</tr>
<tr>
<td>27</td>
<td>All software developed shall be under revision control using SVN, all releases shall be tagged and repository kept on Attentec AB’s servers.</td>
</tr>
<tr>
<td>28</td>
<td>Possible security issues identified during development shall be documented.</td>
</tr>
</tbody>
</table>

Table 4.3: Non-functional requirements verification.

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Priority</th>
<th>Fulfilled</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>Eclipse has been used.</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>26</td>
<td>Code is commented and design is documented in this report.</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>27</td>
<td>Mecurical has been used instead of SVN, but for the same purpose and with the same result so this requirement is considered fulfilled.</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>28</td>
<td>Security issues are documented in Section 4.4 of this report.</td>
<td>1</td>
<td>Yes</td>
</tr>
</tbody>
</table>

4.2 System Viability

The objective of this thesis work is to examine the possibility of developing a lightweight alternative to existing M2M solutions using only common devices and protocols. A system with desired properties has been implemented and therefore the short answer to the initial question is: Yes, it is possible. The implemented system is though very limited in several respects (mainly security and performance). Due to this it can not give a practical verification whether more advanced applications such as security- and/or time-critical systems can be built in the same manner. It can, however,
4.3 RELIABILITY

serve as a practical base that can be studied and potentially extended with more advanced aspects.

4.3 Reliability

As described in Section 3.3.1, the system is designed to handle loss of communication. This is one of the main aspects for this thesis. In testing, the implemented functionality for storing lost messages and resending them works fine. Whenever connection is lost, both devices simply discard all messages received during the current transmission. The master device then returns to the state for waiting for connections and the slave device will wait for the next time it is supposed to send.

4.4 Security

The implemented system is only intended for verifying some basic functionality of a lightweight M2M solution in a protected environment and therefore does not consider security issues. If a similar system should be developed for a real application then security will probably (depending on the application) be of very high priority. Both the TC65T and the Android platform support HTTPS, which is a secure version of HTTP, which in turn is an abstraction of TCP/IP. Since TCP/IP is the used protocol for this implementation a further development of the system using HTTPS for communication between master and slave(s) should be quite feasible.

In its current state security can be breached in numerous ways. For example any device may connect to the master. If the communication interface is known the connected device may pretend to be a slave device and starve the real slave devices by flooding the master.

Other than the GPRS/IP communication the system does also have a security issue with the SMS functionality. The slave is designed to only listen to two different SMS commands: "reset" and valid IPv4 addresses. Since the slave does not check who sent the SMS and no authentication is done, a SMS sent from anywhere containing a valid IP address will make the slave connect to that IP, and likewise a SMS containing "reset" will reset the device. The only thing required to breach security in this way is the telephone number to the slave and a device that is able to send SMS.

4.5 Performance

The system is tested with two slave devices but the architecture is designed to work with arbitrary number of slaves. There is no theoretical maximum for this other than the memory limitations on the master device and the bandwidth offered by the GPRS connection. In the same way there is no theoretical limit on the number of sensors connected to a slave. The number
CHAPTER 4. EVALUATION

of input ports is limited, but a port may be shared by several sensors if some external device collects data from each of those sensors and forwards it to the slave device through a single port. The limit of the system is instead that only one slave at a time may be connected to the master since all slaves connect to the same TCP port. This means that the limiting factor is the frequency and length of the messages.

4.5.1 Performance Tests

A set of tests has been done to measure the time for each update session, where an update session is the event of preparing and transmitting all sensor values collected since the last successful update session. For the test 30 update sessions for each configuration are measured and the average of them is presented. Virtual sensors are used during testing. The virtual sensors are separate threads that give dummy values at a fixed frequency.

In Figure 4.1 each test has a different number of virtual sensors according to their X-axis. When no sensor values are sent only some bytes for synchronizing are sent between the master and the slave. This means that for the configuration without sensor values (test 0 in Figure 4.1) the only timing overhead is the time it takes for GPRS to connect. The figure shows that messages with values up to 8 sensor values take about the same amount of time to send. This indicates that also for those configurations the majority of the time is the set-up time of the GPRS connection.

Tests were also done with only a single virtual sensor giving all sensor values. The update session time for these tests can be seen in Figure 4.2. The difference between 1 and 8 sensor values is insignificant compared to the difference for higher values. One explanation for this behaviour is that for larger messages the connection may be interrupted a lot by the thread running the virtual sensor and therefore the time for the update sessions increases. Another interesting observation is that 8 values takes about the same amount of time with 1 and with 8 virtual sensors. This means that the extra 7 threads does not slow the system down. Here the test with more threads does have a higher performance, but this is probably due to having too few test samples.

To see the scaling for even higher number of values Figure 4.3 shows the update session time for 1 sensor giving 40 values and also the update session time if another similar sensor is added. This shows almost a doubling in time, and considering earlier observations that the GPRS connection time is pretty constant at 6-7 seconds the time for the actual data handling and transfer is more than doubled.

The virtual sensors used in this test to measure the time for 25, 40 and 80 values update their values every second while the other configurations use longer time intervals. This is for practical reasons (the test would take too long to perform for higher values). The sensor frequency may affect the results since a higher frequency of the sensor may interfere more with the
4.5. PERFORMANCE

Figure 4.1: Update session time, multiple sensors. The X-axis represents each measured configuration and the Y-axis the average time for one update session in milliseconds. The tests are performed with one virtual sensor per value (so test 3 has three virtual sensors giving one value each per update session).

Figure 4.2: Update session time, one sensor. The X-axis represents each measured configuration and the Y-axis the average time for one update session in milliseconds. Each test uses a single virtual sensor for all sensor values.
Figure 4.3: Update session time, 40-value sensors. The X-axis represents each measured configuration and the Y-axis the average time for one update session in milliseconds.

In Table 4.4 some examples of what the system should manage can be seen. These values are a theoretical interpretation of the test values from previous tests. The purpose of them is to give an estimate of the performance. The values have some margin compared to the theoretical max computed from the tests to allow for some practical variations.

Table 4.4: Theoretical number of devices the system can handle based on testing. The first column represent the number of sensors connected to each slave. The second column represent the frequency at which the sensors are read. The third column represent the number of slaves connected to the master. The fourth column represents three frequency of the update sessions.

<table>
<thead>
<tr>
<th>Sensors per slave</th>
<th>Sensor update interval</th>
<th>Slaves</th>
<th>Slave update interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 second</td>
<td>1</td>
<td>9 seconds</td>
</tr>
<tr>
<td>5</td>
<td>30 seconds</td>
<td>10</td>
<td>2 minutes</td>
</tr>
<tr>
<td>2</td>
<td>2 minutes</td>
<td>180</td>
<td>30 minutes</td>
</tr>
<tr>
<td>10</td>
<td>30 minutes</td>
<td>100</td>
<td>17 minutes</td>
</tr>
</tbody>
</table>

Optimizations can probably be done to decrease the time for longer messages but the time for GPRS to connect can not be neglected. Future applications using this core structure may utilize 3G instead of GPRS to greatly
decrease the time to connect and send messages.

4.6 Time-outs

When the slave connects to the master it cannot know whether it will get a response or not. The IP address may be false or the master may be shut down. This problem exists for most (maybe all) client-server connections. The common solution, which is also used here, is to have a time-out added to the connection request. When the time-out is reached the client stops waiting for a response and continues its execution. The master has a similar time-out since connection may also be lost during an update session. There are two different time-outs used for the connection.

- **First connection time-out** is used in the slave when it does not have any configuration and connects to the master. The purpose of this time-out is to give the user time to enter the name and the update interval for the slave. The time-out is set to 65 seconds. If no response from the master is received in time the slave will wait for 60 seconds and then try to connect again.

- **Regular connection time-out** is 15 seconds and is used for all connection occasions in the master and all except for the first in the slave.

Common for both time-outs are that no effort is put into finding the perfect values for them simply because the implementation in this thesis does not target a specific application. In a real application these values will probably make a significant difference and the regular connection time-out may be split into several different time-outs.
Chapter 5

Summary and Conclusions

M2M is a fast growing technology and many businesses are utilizing it for remote management of equipment. By developing a more cost-efficient M2M architecture the technology may become available in new business areas and new types of services may arise. Added mobility also increases the number of possibilities with the technology.

This thesis examines the possibilities for a new M2M architecture using an Android platform as both back-end server and user interface. This architecture greatly increases the cost-effectiveness and mobility while limiting the performance. The architecture developed is intended as a general architecture to be used as a core when designing application-specific M2M systems and therefore takes no account of application-specific parameters.

The developed system is not suitable for large scale applications where lots of nodes should be monitored. Instead the intention is that this architecture should be used to enable new services that require small-scale monitoring and/or control.

The hardware platforms used in this thesis has been chosen before the thesis started. Due to this there is a risk that the implementation has been affected by the functionality of these platforms. The use of requirements has reduced this risk since they were listed before any development had started (and therefore the knowledge about the platforms were limited in terms of programming). However, since not all of the priority 2 and 3 requirements are fulfilled the platforms may affect the possible implementation of some of these.

Most systems developed today have security as a high priority and therefore it may have been better to consider security, leading to some choices in the implementation. If the communication between master and slave used the HTTPS protocol instead of standard TCP/IP, an added level of security would have been present. This would probably have been more time consuming and would probably mean that some other part of the implementation would have been left out but the results may have been of more
interest.

The system implemented in this thesis is working and gives a practical indication that it is possible to develop M2M applications using only common hardware devices and protocols. It also demonstrates that such a system can be developed in a reasonable time. The use of general-purpose slave devices makes this system possible to use in a lot of different application areas and the mobility of the Android platform even further increases the possibilities.

5.1 Future Work

There are a lot of features that can be added to a system like the one described in this thesis. The most important ones regard security and reliability since those are probably the most critical if a system like this should be used in a real application. The protocol used for communication needs to be revisited as well. This thesis only aims to prove that a system that meets the requirements for a lightweight M2M solution can be implemented.

Performance will be increased without any effort from the developer since 3G/4G will soon be standard for the kind of hardware platforms used. The tablets will have increased performance as well. Using a standard platform for the master device (Android, iOS, Windows Phone, etc.) gives a lot of possibilities for the user interface so that should never have to be a limiting factor for an application. Currently there are standard components for different graphical views, like diagrams and map views.
Bibliography


På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Magnus Gustafsson
Machine-to-machine communication (M2M) is a generic term for technologies dealing with autonomous communication between machines. For the last 10 years a wide range of business areas utilize a variety of different M2M solutions for remote management of equipment. Common for almost all of those solutions is that they are expensive and require the infrastructure to be adapted to them. They are also usually built out of several different systems working together and thus there are several systems that require maintenance.

This thesis investigates the possibility to develop a lightweight alternative to existing M2M solutions using only common devices and protocols. Lightweight here means that the system should be flexible, have a low cost for set-up and operation and that both ends should be mobile. By developing a lightweight M2M architecture the technology may become available in new business areas and new types of services may arise.

In the thesis a prototype is implemented. The purpose of the prototype is to practically verify whether a lightweight M2M solution is possible to develop in this manner. The solution uses the Android platform for back-end and user interface and a Cinterion TC65T as slave device to which the sensors can be connected. The implemented system is limited in terms of security and performance but still acts as a proof of concept for this kind of M2M solution.