UNIVERSITY OF OSLO
Department of Informatics

Sensors Integration into Heterogeneous Services Platform and Domain Adaptation

Yen N. T. Pham

December 20, 2010

Acknowledgements

This thesis is written by Yen N. T. Pham. This is the final task for the Master's degree at the University of Oslo, Faculty of Mathematics and Natural Sciences, Department of Informatics. Work on this task has been running since August 2010 to the 20th December 2010.

In this occasion, I will first and foremost thank my parents who have brought me here to this beautiful country, and have given me the opportunity to education and careers. You have always been supportive and encouraging to me throughout my life. When I needed some time off, and filled this with many pleasant memorials in the family gathering. These gave me more strength and energy so I could continue my way forward.

I would also like to take this opportunity to thank my leaders at Oslo University Hospital, Arve Kaaresen and Erik Johannessen. Thank you for having allowed me so I could immerse myself in the profession, and complete this thesis. With this, I'm very grateful.

Many thanks to Professor Josef Noll at University in Kjeller (UNIK), who gave me this wonderful and exciting subject to study on. Thank you for allowing me to experience an interesting and instructive exhibit, which given me more understanding the contents of the subject. It has helped me to realize this thesis. And Sarfraz Alam who guided me, many thanks for all the details and good explanations about the topic.

A lot of thanks to Telenor Object project team, who have allowed me to use the Shepherd® platform and provided the materials. With special thanks to Espen Nersveen for sharing your excellent work with me, and been helpful in answering my inquiries and questions. Thanks for the many good and useful answers, which I appreciated.

Acronyms

AFEDEF	Association des Fabricants Euroṕens d'Equipements Ferroviaires
AICMR	Association Internationale des Constructeurs de Matérial Roulant
AMPS	Advanced Mobile Phone Service
AMS	Application Management Software
API	Application Program Interface
ATC	Automatic Train Control
BSS	Base Station Sub-System
BSC	Base Station Controller
BTS	Base Transceiver Station
CDC	Connected Device Configuration
CDMA	Code Division Multiple Access
CELTE	Constructeurs Euroṕens des Locomotives Thermiques et Electriques
CEPT	Conference of European Postal and Telecoms administrations
CHPC	Confirmation of High Priority Calls
CLDC	Connected Limited Device Configuration
COOS	Connected Objects Operating System
CO	Connected Object
CWI	Center for Wireless Innovation Norway
DAB	Digital Audio Broadcasting
DICO	Deployed Infrastructure for Connected Objects
EDGE	Enhanced Data rates for GSM Evolution
EIRENE	European Integrated Railway radio Enhanced Network
EIRENE MoU	EIRENE Memorandum of Understanding
ERIG	European Radio Implementation Group
ERTMS	European Rail Traffic Management Systems
ETCS	European Transport Communication System
ETSI	European Telecommunications Standards Institute

EURO	EURORADIO
EVC	European Vital Computer
FA	Functional Addressing
FDMA	Frequency Division Multiplexing Access
FN	Functional Number
GCF	Generic Connection Framework
GPS	Global Position System
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications
GSM-R	Global System for Mobile Communications - Railway
GUI	Graphical User Interface
HLR	Home Location Register
HSPA+	High-speed Packet Access
HTTP	Hypertext Transfer Protocol
ICT	Information and Communications Technologies
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IMEI	International Mobile Equipment Identity
IMP	Information Module Profile
IoT	Internet of Things
IN	Intelligent Network
ITEA	Information Technology for European Advancement
ITU	International Telecommunication Union
Java EE	Java Platform, Enterprise Edition
J2ME	Java 2 Platform, Micro Edition
J2SE	Java Platform, Standard Edition
JBV	Jernbaneverket

JVM	Java Virtial Machine
LCC	Life Cycle Cost
LDA	Location Dependent Addressing
M2M	Machine to Machine
ME	Mobile Equipment
MIDP	Mobile Information Device Profile
MMS	Multimedia Messaging Service
MORANE	Mobile Radio for Railway Networks in Europe
MSC	Mobile service Switching Center
MSISDN	Mobile Station Integrated Services Digital Network
NSS	Network Sub-System
NP	Numbering Plan
OMC	Operation and Maitenance Centre
OTA	Over-The-Air
PDA	Personal Digital Assistant
PFN	Presentation of Functional Numbers
PMR	Private Mobile Radio
pSHIELD	pilot embedded Systems arHItecturE for multi-Layer Dependeable solutions
RBC	Radio Block Center
RFID	Radio-frequency identification
SIM	Subscriber Identify Moules
SLA	Service Level Agreement
SMS	Short Messaging Service
SN	Subscriber Number
SPD	Security, Privacy and Dependability
SunSPOT	Sun Small Programmable Object Technology
SunSPOT JDK	Sun SPOT Java Development Kit
SunSPOT SDK	Sun SPOT Software Development Kit

TCP/IP	Transmission Control Protocol over IP
TDMA	Time Division Multiple Access
TETRA	Terrestrial Trunked Radio
TSI	Technical Specification of Interoperability
UDP	User Datagram Protocol
UIC	Union International des Chemins de Fer
UML	Unified Modelling Language
UMTS	Universal Mobile Telecommunications System
URI	Uniform Resource Identifier
URL	Uniform Resource Locators
VLR	Visitor Location Register
VM	Virtual Machine
VPN	Virtual Private Network
WAP	Wireless Application Protocol
WSN	Wireless Sensor Network
WLAN	Wireless Local Area Network
XML	Extensible Markup Language

Contents

List of Figures

List of Tables

Abstract

The M2M is a fairly new term and has been widely used in recent years. The capability and ability to build a network infrastructure of ubiquitous devices that performs automatic data transfer. The standard for such infrastructure is ETSI TS 102.690. Many enterprises such as the telecommunications, have introduced the M2M concept, as well as the implementation. The goal is to adapt this infrastructure in the existing network. Some have completed it successfully such as the railway domain, where GSM-R is an important technology.

The development of new technology should possess the security plan in the protection of data transmission and the crossover across the networks. This can be solved by adaptation of an M2M nettwork infrastructure, where security is defined at all levels. From the sources and the end users.

Here, the sources can be the nodes, a software entity for the middle layer or a third-party provider. While, the end users are defined to be the part who uses the M2M systems. These might be the railway headquarters, the train operators or the mobile devices, who want the access for obtaining the information to be displayed in-situ.

The M2M is expected to increase quickly in near future and to be applied to the Embedded Systems, where the wireless technology is the keyword. To begin with, it is necessary for an intermediate layer that enables the integration of different sensor types. In addition, this platform should offer opportunity for M2M implementation, so that it can be adapted for the railway domain or other domains if necessary. One such platform is the Shepherd® platform.

Chapter 1

1 Introduction

For many decades and still today, there has been a domination of wired networks that allow computers within a network to exchange information with other computers nearby. Then, the massive growth in using the Internet world wide made it possible for computers to exchange data, even though, the computers are not physically connected together, but rather through ”links connectivity”. Today, a new trend has already emerged on the market and it seems to be more and more popular in the future. That is a network of Wireless Sensor (WSN).

The sensor mote is a small device with low power that can sense the environment. The sensor can be embedded inside a machine or a larger device to make it possible for the machine to actually ”talk” and exchange data among other machines or devices without any wired connectivity. This development introduces many new applications and open a wide range of opportunities for it to be adapted to our daily lives, for instance the mobile devices in the telecommunications domain.

The next generation of development, is that the sensors can be more flexible in the sense that they can sense and detect the changing environment over a large geographical area, in spite of obstacles such as buildings, tunnels or power constraints. The world of automatic smart sensors can interact and communicate to deliver useful information which can make our lives easier.

There are sensors that can detect temperature changes, sense the light on/off, pressure changes or measure any kind of enviromental change. The sensors have, therefore, a vital role in many circumstances, especially, in the healthcare domain where the doctors need to know the patient's condition by monitoring, so the correct diagnosis can be set for that patient. Another important need to be mentioned here, is that the sensors can be embedded in a small device such as the cardiac pacemaker, which then, corrects the patients heart rhythms.

The communication between the sensors is wireless, meaning radio communication. The use of wireless technologies will increase more and more in the future. There are several open standards for wireless technologies such as IEEE 802.11X for WLAN[1] that has been used in laptops. While, in the telecommunication technol-

[1]IEEE wireless standards: `http://standards.ieee.org/getieee802/802.11.html`, last accessed 9. Sep, 2010

ogy, there are standards such as IMT-2000 for UMTS[2], ETSI for GSM[3]. Others, like GPRS, EDGE and Bluetooth/ZigBee have been adapted in mobile devices such as cell phones and PDAs.

As long as the evolution of new sensor technologies never stops, there will be demand for new applications to be developed and targed towards several types of sensors that are found in the market. The need of new applications to control any type of sensors as well as to automate the sensor's communication with each other. Despite of the distance between them, or how the sensors are scattered in the area. The data and information must still be reached and displayed.

1.1 Research Approaches

The methodologies that have been used in this study are mainly based on the investigation and searching of previous work on this topic. In addition, it also performed an analysis of standard TS 102.690, and the technologies that are related and have implemented, or in connection with this. It has used search engines from the University of Oslo Library, for example BIBSYS, E-journals, e-books and Google Scholar.

It has also been in contact with Norwegian National Rail Administration (JBV) to understand the current state. Since this thesis especially targets the JBV in Oslo. The research is also based on the problems that Norwegian National Rail Administration (JBV) are facing today. However, much information has also be explained by the supervisors, and the materials are also provided.

Furthermore, it has also been in contact with Telenor Objects from Telenor, which offers a platform for services integration. Where this is important in the implementation of protype for sensors integration that will carry out this thesis.

Telenor is one of Norway's largest telecommunications companies. It is responsible for delivering of services to many telecommunication systems in Norway. Telenor Objects[4] from Telenor offer a platform that supports the Machine-to-Machine (M2M), that guarantees a platform of stable, secure, reliable and scalable message exchange between the M2M devices. This platform named "Sherpherd®". It can be used to support any kind of application belonging to a device type, in communication with other different application of that device. This facilitates the integration of heterogenous applications and sensors, which the JBV has been looking for.

[2]UTMS: `http://www.umtsinfo.co.uk/imt2000standard.html`, last accessed 9. Sep, 2010

[3]The ETSI webpage: `http://www.etsi.org`, last accessed 9. Sep, 2010

[4]Telenor Objects AS: `http://telenorobjects.com`, last accessed 9. Sep, 2010

1.2 Related Works

A group of members from many countries in Europe, Asia and America, are coming together and formed the UIC[1][5] [6]. The goals are the *interoperability* of Railway Systems across the boundaries.
According to the definition [7], *interoperability* means, to be able to operate and co-operate with any rail network without noticing the differences. This is one of many directives made by the European Parliament and the Council in 2004 to obtain a high-speed rail system.

The UIC started the projects EIRENE and MORANE in 1992 that clearly defines the requirements, technologies and standards to be used. The ETSI which provides the standards for the ICT, including the mobile, radio, broadcasting and Internet technologies, that have been used in the EIRENE & MORANE specifications[8] [9]. The projects have been closed, but the researches are still going on in the application of new technologies of ERTMS to be used for the Railway domain in the future. The formation of EIRENE MoU consists of many countries around the world including Norway[10] [2] with the agreement and co-operations of common system for the railway domain, that is GSM-R.

Nortel[11] is the supplier of telecommunications equipment in Canada that has developed the GSM-R for wireless communication for the European railway networks. Currently has Nortel national contracts with France, Germany, UK and Australia, as well as China, Italy, Spain and India. But, the goal is also to bring the technology into other countries around the world.

AJA Solutions is another company that delivers devices for rail's mobile communication systems based on GSM-R[12].

SELEX-Communication[13] has the headquarter in Italy that has been used technol-

[5]http://portal.etsi.org/eeurope/Documents/RailwayCommunicationSystem.pdf, last accessed 9. Sep, 2010

[6]UIC webpage: http://www.uic.org, last accessed 9. Sep, 2010

[7]Definition: http://europa.eu/legislation_summaries/transport/rail_transport/l24015_en.htm. Published 23. Jan, 2007, last accessed 9. Sep, 2010

[8]MORANE webpage: http://gsm-r.uic.asso.fr/morane.html, last accessed 16. Aug, 2010

[9]Specifications: http://gsm-r.uic.asso.fr/specifications.html, last accessed 16. Aug, 2010

[10]EIRENE MoU: http://www.uic.org/spip.php?article435, last accessed 16. Aug, 2010

[11]http://www2.nortel.com/go/solution_content.jsp?segId=0\&catId=0\&parId=0\&prod_id=42102, last accessed 9. Sep, 2010

[12]http://www.ajasolutions.com/gsm-r.php, last accessed 25. Aug. 2010

[13]http://www.selex-comms.com/internet/?open0=5\&open1=28\&open2=64\&section=

ogy named TETRA[14], but also delivers handheld devices, that use GSM-R [3][15]. Their products are now being used in the railway of United Kingdom.

In 2008, Telit Wireless Solutions[16] company developed and manufactured M2M modules that enabled machines, devices and vehicles to communicate wirelessly such as GSM, GPRS, UMTS, ZigBee, and so on. The technologies are now available on the markets. While, the products are produced by the company's headquarters in Seoul. The main headquarters are in Rome, North America (Rlaeigh), São Paulo and Seoul. Telit has established a presence in many other countries such as Germany, Austria, Switzerland, Israel, Russia, Nordics, Baltics, Benelux, South Africa, Spain, Turkey and many other countries.

1.3 Problem Statement

The Norwegian National Administration (JBV[17]) is formed in 1996, with headquarters located in Oslo. The JBV is an important railway authority in Norway, it has the main responsibilities in transport, communication and traffic management for Norwegian national railway network.

Today, the majority of the population in Norway prefer daily travel by train, and the number of passenger will be increased every day. Based on a report from JBV, there are about 57 millions per year who use the national railway. The transport of goods are more than 25 millions tons [4]. In line with the increasing number of travelers and the transportation of goods. There will be a need for more trains on the railway to meet the demand. It is also necessary to renew the railway infrastructure, which is sometimes 40 years old, as well as to build new railways with the possibility of crossover between the country's borders.

The current railway systems exist in Norway[4] are based on the remote and interlocking of signals, that control the "stop" sign or "allow" sign of the railway for the train either to wait or to have a free railway. The speed monitoring systems (ATC) can monitor the trains speed over a large geographical area, which are controlled by the traffic control unit. The information from the interlocking is sent to the computers on the train through the baliser places on the rail track. The

CORP\&showentry=5436, last accessed 25. Aug, 2010

[14]http://www.selex-comms.com/internet/?open0=5\&open1=28\&open2=64\&section=CORP\&showentry=5436, last accessed 9. Sep, 2010

[15]http://www.selex-comms.com/internet/media/docs/GSM-R_Hand_Held_EN_LR.pdf, last accessed 25. Aug, 2010

[16]Telit: http://www.telit.com/en/, last accessed 25. Aug, 2010

[17]Jernebaneverket: http://www.jernbaneverket.no/en/Startpage/, last accessed 18. Nov, 2010

interlocking is installed on every station along the railway, and consists of train detector, rail path and rail track interactions. About 90% of Norway's railways have such monitoring system installed since 1980. The lifetime for such systems is estimated to be 40-50 years (LCC).

The Norwegian Parliament and Ministry of Transport and Communications have approved the establishment of ERTMS network at the national plan according to the European's master plan[18] [4]. The strategies have been clearly defined earlier in 2005, while the directive for higher speed train has been worked out in 2004. A number of measures have been implemented in recent years e.g. safety regulations and roaming regulations.
In May 2010, the JBV started a project that aimed to build a new railway system based on ERTMS/ ETCS as the new national signal system for future railway domain. The project will be ended by 2019.

Another challenge that the JBV faces is the weather conditions and geography in Norway. To mention, extreme weather conditions, no range covering in high mountains and tunneling. The GPS system does not offer enough signal strength when the train is in the tunnel or at high mountain. Poor signals lead to problems of tracking the train's position which means a lot of delays for the passengers. The old railways and communication systems used in bad weather are inadequate and there is a lack of systems, that can handle such conditions.

In 2006, the JBV started a project of railway safety based on UIC where EIRENE specifications for *functional* and *requirements* were used. The recommended technology GSM-R[19] had been introduced. It is not for commercial telecommunications, but rather targets the rail systems. It has mainly has been applied in Oslo and some cities in mid- and north of Norway [5][20]. The GSM-R technology is basically based on current GSM that is found in mobile devices.

The need of new technologies also leads to the need for new devices that is equipped with new technologies. This introduces several different applications within the devices. There are therefore need for a platform that supports the communication systems between different software.
Moreover, different devices with different applications is a challenge for the integration of heterogeneity in the network. This is also the target to the mobile devices, that are existing today in others countries. *Interoperability* is an important word

[18]ERTMS-ETCS: `http://www.jernbaneverket.no/no/Prosjekter/Utredninger/ERTMS-ETCS/`. Published 22. Jun, 2010, last accessed 9. Sep, 2010

[19]`http://www.jernbaneverket.no/en/Startpage/Market/GSM-R-mobile-services`, last accessed 18. Nov, 2010

[20]Map: `http://www.jernbaneverket.no/Documents/Marked/GSM-R%20mobiltjenester/GSM-R%20i%20Norge.pdf`, last accessed 9. Sep, 2010

for interaction. The *interoperability* should be seamlessly and agile when applying to the device communications. If these goals are achieved, they can be applied to many domains and realize the business goals, such as facilitating railway traffic across the borders.

1.4 Proposal for Solution

To apply the M2M architecture as a solution for the JBV according to the standard ETSI TS 102.690 (refer to section 2.2.2) is described in Chapter 2. This thesis suggests a solution for integration of sensors into a platform called *Shepherd*. The advantage of this platform is that it targets the heterogeneity, and the platform provides a set of services to integrate several kinds of sensor types, such as RFID, GPS, light, temperature, accelerometer and so on.

Chapter 3 introduces this solution in more details based on defining the needs of the sensors at different scenarios, where the rail meets the difficulties along its trip. The survey is aimed at the Norwegian conditions.

Based on the solution, this thesis developed a prototype using the SunSPOT from Sun Labs. The SunSPOT communicate within a short distance of 10 metres using the protocol IEEE 802.15.4, which is designed for the device with constraints such as memory and power.
The SunSPOT has three embedded sensors with the ability to measure the temperature, light and accelerometer. One of the advantages of the SunSPOT is that there can be added another sensor into the SunSPOT such as the GPS. This sensor can be used to track the rail's location, which is right on the target for the JBV.

1.5 Structure of Thesis

This chapter gives an introduction, motivation and point out the problems to be found for the railway system, the reasons for replacing the railway domain with new technologies.

This thesis is structured as following; Chapter 2 describes the state-of-the-art about the technologies and the analysis of the M2M and its architecture according to the standard TS 102.690, then desribes the characteristics of wireless sensor network. It also includes an analysis of GSM/GSM-R, the SunSPOT[21] as well as the Shepherd® platform.

[21]SunSPOT webpage: `http://sunspotworld.com/`, last accessed 9. Sep, 2010

Chapter 3 suggests the technologies to be applied at the JBV, and points out the scenarios where the SunSPOT can be used with the Shepherd, and how the end-users can retrieve the information from the SunSPOT through the Shepherd platform.

Chapter 4 explains in more detail the devices and technologies' characteristics. Then, it introduces the implementations of a prototype. Later, it shows the achievements in developing a program for the integration of SunSPOT into the Shepherd® platform.

Chapter 5 takes up the aims of the ARTEMIS pSHIELD project, which this thesis is part of the project. Later in this chapter, the prototype is presented and demonstrated at the Co-Summit in Belgium.

Chaper 6 gives an introduction of performance analysis and performance measurement of the developed program. Along with this, evaluate and discuss the measurements. At the end of this Chapter, it discussed the maintenance aspects of software, devices and system, which is useful for JBV. As well as taking up works that can be further developed based on problem condition mentioned in the Chapter 3, and what should do next for the program so it can be adapted and used in the railway.

Chapter 7 conclude whether it was necessary for implementation of M2M nettwork infrastructure with GSM-R technology or not. If it is, should it be adapted for the railway domain, and how this can be done.

Chapter 2

2 State-of-The-Art

This section describes the state-of-the-art specific for the railway domain. This includes the European standards recommendation to be applied to the railway systems. Later, the explanation of M2M architecture based on the standard ETSI TS 102.690, and applying to the existing wireless network. Then, exploration of GSM technology in relation to GSM-R. Again, the railway domain is the target. Later, a short description of the sensors that can be adapted to the JBV, as well as the components in the Shepherd® platform.

2.1 Domain Specific - One Railway System

Towards a common system - ERTMS/ETCS

The Association of the European Railway Industry - UNIFE[22] consists of three associations AICMR, AFEDEF and CELTE, that all are located in Brussels. The associations represent 62 rail-supply companies around Europe and have six members Alstom Transport, Ansaldo STS, Bombardier Transportation, Invensys Rail Group, and Siemens Mobility and Thales. They have a common goal, that is to introduce the ERTMS to European Union railway systems [6].

Today, there are about 20 different national control systems for the railway in Europe. Each system operates seperately and only works for that country. The incompatibility of systems is the reason that the UNIFE, currently, co-operates to develope a common ERTMS. This aimed to create a seamless rail crossing without signal failure. Clearly, there is a need for the *interoperability* system that allows, easily, switch between one railway system to another when crossing the international borders. This, also allows to operate at high-speed up to 500km/h along the railway traffic, and supporting of data exchange in traffic management as well.
As it has been suggested in [7] for the so-called "trans-European traffic management", which means to apply the ERTMS as one common and known system to achieve the interoperability assessement. This could be a solution to the railways intersection between the boundaries.

[22]`http://www.unife.org/`, last accessed 4. Des, 2010

The UIC guideline [8], that describes and defines the requirements for the *interoperability*, declares that it should include and cover the following assessments:

(i) The railway emergency calls should be in both directions between the driver and the controller.

(ii) Accept non-emergency calls, either from the controller or from the driver.

(iii) The ability for registration and de-registration of procedures.

(iv) Uses a common European Train Control System.

(v) Allows for the messages from Driver Safety Device.

(vi) Allows driver to driver calls for assistance.

(vii) Apply the public emergency calls.

The architecture of ERTMS consists of three main parts[23]:

I. A Trackside involves the eurobalise on the rain, euroloop and RBC. The eurobalise is a beacon that allows communication between the track and the crossing train. The euroloop makes it possible for the data exchange along the railways. And the RBC is, mainly, consisted of a server for accessing, a telecom board, and a vital computer.

II. A Trainborn has an integrated component in the train (EVC), which is a set of interfaces for the communication with mobile environments such as sensors, balise or "mobile terminal". These are the traffic control centers.

III. Exists a GSM-R network to allow the communication between the trackside and the trainborn.

The main activity component here is ETCS. This component is the main control system with full automatic protection. This means that it handles the safety, a continuos signalling, as well as the data exchange. So, when a train is in movement, it has the authority to move on that rail safely based on the ETCS, that has a well-monitoring system.

Further, the ERTMS consists of three levels [9]:

I. Level 1: Figure 1 shows the communications between the track and the train. These are based on *Eurobalises*, which are placed along the trackside, so the signaling exchangers occur at the time the train is passing by. At this time, the train control center communicates to the train driver through the lineside

[23]What is ERTMS - Introduction, `http://www.ertms.com/2007v2/what.html`, last accessed 9. Sep, 2010

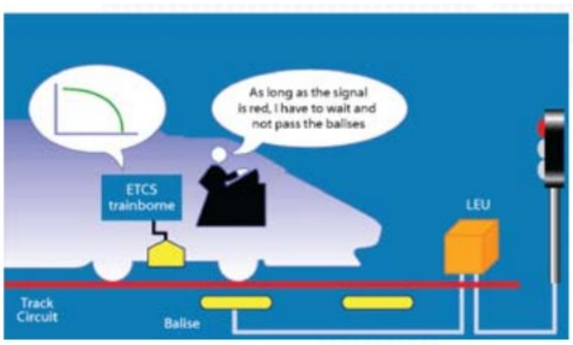

As long as the signal is red, I have to wait and not pass the balises
ETCS trainborne
LEU
Track Circuit
Balise

Figure 1: ERTMS level 1. Copy from [9].

signals to give the train the permission to continue.
Another advantage of using ETCS is that it can operate on board to calculate automatically the maximum speed for the driving authority. If the train drives at a higher speed than it permits, it will be slowed down and stopped.

II. Level 2: The communications between the train driver and the RBC are more stable and faster based on GSM-R network. Figure 2 shows replacing of signal system with RBC, and introducing the wireless technology at this level. Here, the interlocking can still be used as the signal exchange. But, the messages transfering is based on the *balise* instead. The messages can

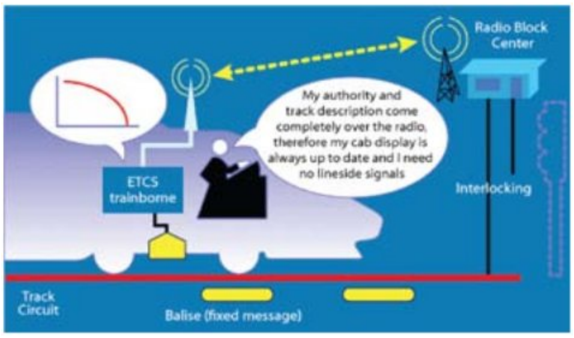

Radio Block Center
My authority and track description come completely over the radio, therefore my cab display is always up to date and I need no lineside signals
ETCS trainborne
Interlocking
Track Circuit
Balise (fixed message)

Figure 2: ERMTS level 2. Copy from [9].

be, for instance, the train's location, the gradient, the speed limit, or even calculation of an optimal speed, and so on. The advantages of this level are that, the capacity or the number of train movement on the same railway is increasingly. To mention, the cost for maintenance is low and that is a benefit for the business model.
Currently, the JBV has plans for the implementation of ERTMS level 2 as the future rail network.

III. Level 3: The communications consist entirely of the GSM-R network. Figure 3 shows removing of signalling exchangers, and replaces with wireless RBC

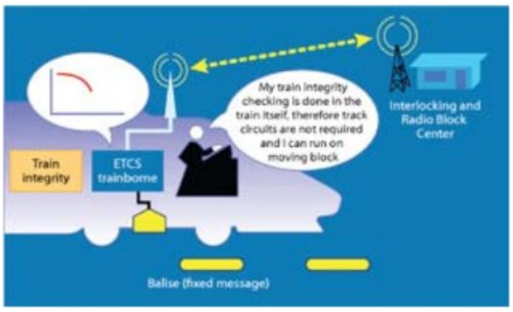

Train integrity
ETCS trainborne
My train integrity checking is done in the train itself, therefore track circuits are not required and I can run on moving block
Interlocking and Radio Block Center
Balise (fixed message)

Figure 3: ERTMS level 3. Copy from [9].

base station. Here, there will be a continous monitoring of the train's movement all the way.
The advantage with level 3 are that, it increases the possibility for monitoring and detecting more than one train on the same rail-track at the same time. But, the preparation for this level is still going on. When the time this can be implemented, there will be possible to upgrade from level 2 to level 3 by replacing the track-based equipments with GSM-R technologies. The upgrading from level 1 to level 2 is also possible.

The concept of ERTMS levels has been validated by the TSI. Therefore, it should be safe for the railway domain to plan and replace the recent ATC system.
For the Norwegian railway system, introducing ERTMS/ETCS needs to be based on the standards and the concepts defined for European interoperability. This means, it also satisfies the JBV's requirements, goals, and needs.

2.2 Network Architecture

This section describes the network in different perspectives for the wireless with M2M communication in relation to the standard for this infrastructure.

2.2.1 Wireless Sensor Network

The sensor nodes (motes) and the base stations are the main elements to be considered and brought up in this subject, when talking about what is to be found in a wireless sensor network (WSN).

Roman et al. [10] describe these *devices* with the important roles and characteristics, that are bound to them as the functionalities these can have, to form a WSN. The ability of automation and self-maintenance, as well as co-operation are the major and central factors in WSN today. To mention, these are the key points, and also the reasons for the adaption, and the popularity of WSN.
The main components in WSN are:

1) A sensor mote is a small device equipped with limited battery and memory. A mote composes of several unit including a sensing unit, a processing unit, ta ransceiver unit and a power unit [11]. The sensor node has therefore the capability of sensing the environment in real-time, for instance, a sensor can sense the temperature, one senses the light, while, another senses the accelerometer, and so on. These sensors are actually measured the gradient in values, that are changeable which people does not do, but needs to know. These sensors can be embedded in a larger device.

2) A base station is a more powerful device than a mote in the sense that, the base station can perform more processing such as collecting and storing the values, that are sent from the motes. There can be found many base stations in a WSN, and their task is mainly information collection, and can therefore form a *"data acquisition network"* [10]. In addition, the base station also has the function to control the sensor's behaviour.

The devices that are used in this thesis are the SunSPOT Development Kit. The kit consists of devices equipqed with sensors, and a basestation. The basestation is used to communicate with the devices equipped with the sensors.

Another example where the sensors have been used are in mobile device, such as GPS sensor in the cellular phone for location tracking.
The sensor that plays a very important role in the healthcare domain is the pacemaker. This device has the sensors to monitor the patient conditions, then, sends an alert to the doctor if the condition has worsened. Or it can even adjust and correct the heart beat of the patient.

Today, there are many other sensors that interact and provide the information to other larger systems for use in statistical purposes.

Figure. 4 shows the interactions of sensors within devices and the formation of a WSN. If we take a look around, we are already surrounded with sensors around us that we are not aware of.

Going from analog systems to digital systems where one expects the WSN to become more intelligent, yet behave automatically in the network.

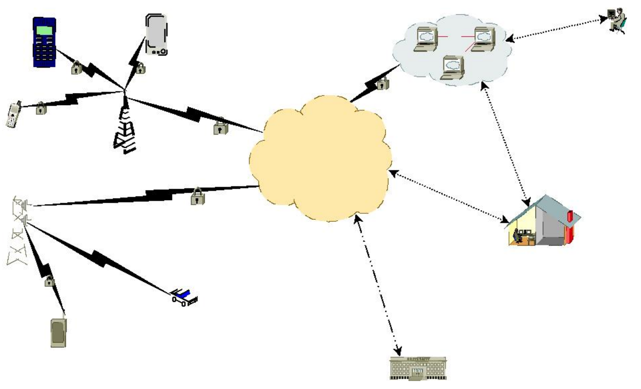


Figure 4: Wireless Sensor Networks

The first generation of radio cellular network [12] is the analog system, the so-called 1G systems. Here, the mobile device is equipped with antenna, which communicates wireless with the basestation. This communication direction is called an *"Up-link"* . Several mobile devices can be connected to the same basestation at the same time. While, in the opposite direction is called *"Down-link"*.
The first 1G was released and was targeting the commercial cellular network, that provides AMPS. This techonology uses the FDMA, which allows the multiple accesses of analog voice over channels that operate at the frequency band of 800MHz.

The second generation which is 2G, opened the evolution of the digital voice communication. Here, the channel is seperated in several time slots so it can allow many more users access to the same channel at the same time. For example, the technology uses a GSM-2G circuit-based method to allow for both voice and data communications, such as text message (SMS). The property of 2G is that the channel access methods are operated at physical layer such as TDMA and CDMA.

Today, many researchers are still working on the third generation (3G), but also the fourth geneneration (4G), that will be on the market in the future. The most popular current technology is based on the International Mobile Telecommunications-2000 (IMT-2000). Meanwhile, there are many technologies available on the market such as UMTS, HSPA+, EDGE and so on. These technologies are based on spread spectrum radio transmission, which can give a *Down-link* up to 56Mbit/s, and an *Up-link* of 22Mbit/s. All are based on the standards from the ITU.

2.2.2 Machine-to-Machine

In recent years, there has been an enormous increase in mobile devices. Several names have been used such as smart phones, smart talks, smart services[24] and so on. On the economy perspective, this adoption has been the main course to the business model [13][25]. The reason is especially in development of new services for these devices. The reason for this is that there has been an increase in service needs required by customers. But it has also been a need in other context such as the healthcare domain[26] [27].

According to the definition from [14], the concept of M2M denotes a communication mechanism between two or more machines in a network. The main goal of M2M is to make the devices communicate with each other without human interaction, the so-called *automatic network of smart devices.*
The idea of M2M technology is to enable the flow of information between different devices, so that the device can deliver the "observations" it has performed in-situ to the end users.

The ETSI[28], who has designed the architecture for the M2M, has defined a new concept for M2M as the *"Internet of Things"*. The new concept represents a platform of sensor integration and service integration. The application is developed independently, but still allows for communication back and forth between the devices. So it may seem that the devices *talk* to each other. The communication can take place in both the fixed or wireless network. This means that the computer can communicate wirelessly with a wireless device, or vice versa.

In fact, M2M is very important in today's network architecture, especially for the telemetry world where programs are designed to enable the automatic remote transmission, for example, provide measured values [15]. Figure 5 shows M2M concept that is based on three modules:

1. First module is the *"M2M device"*. This is used as an *intermediary* that links between sensors and application.

[24] `http://m2m.orangeom.com/`, last accessed 9. Sep, 2010

[25] White paper: `http://www.orange-business.com/content/mnc/kc/wp_M2M.pdf`, last accessed 9. Sep, 2010

[26] Medical Equipments: `http://www.vti.fi/en/applications/medical-solutions/`, last accessed 9. Sep, 2010

[27] Noninvasive Real-Time Optical Sensors For Measuring Vital Physiological Parameters (Mendelson), `http://www.wpi.edu/academics/Depts/BME/Research/mendelson.html`, last accessed 9. Sep, 2010

[28] M2M: `http://www.etsi.org/WebSite/NewsandEvents/M2M/2010_M2M_INTRO.aspx`, last accessed 9. Sep, 2010

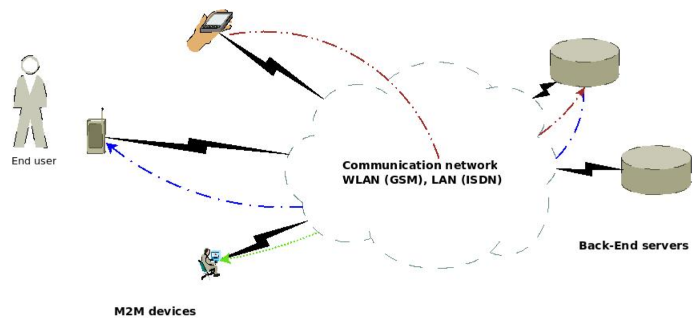


Figure 5: Concept of M2M

2. Second module is a *"Communication network"*. This can be based on WLAN with technologies GSM, GSM-R and so on. The purpose of this module is to etablish a communication link between any devices with M2M applications.
3. Third module is *"Back-end server"*. This is used as a storage medium and serves as an information collection where it is stored information that M2M devices collected and sent away. To retrieve information from the storage medium, it can develop applications that use protocols such as HTTP, FTP. Or applications based on XML, Web services (SOAP) or SQL queries.

Currently, there is a standard for M2M, the *ETSI TS 102.690* [16]. But this is still under validation. However, Telenor Objects from Norway has already started to implement M2M infrastructure, the so-called *Shepherd platform*. This platform consists of several modules such as connectivities to the Shepherd via the Service Enabler API. This requires the M2M device to behave like a Connected Object (CO)*.

2.2.3 The Standard ETSI TS 102.690

The specification for the TS 102.690 standard [16] is provided by ETSI. This standard is adapted and can be used for any infrastructure based on M2M concept. This section describes the main points of the standard based on the this concept.

Figure 6 pulls out the most important elements and reqirements of the standard. These must be implemented at both the application level, it must also be implemented at the network level as well. The standard also addresses the security and authentication for each module.
The start of the module is the *Service Capabilities* that provide adequate access

*See Chapter 2.5 about *Telenor Objects*

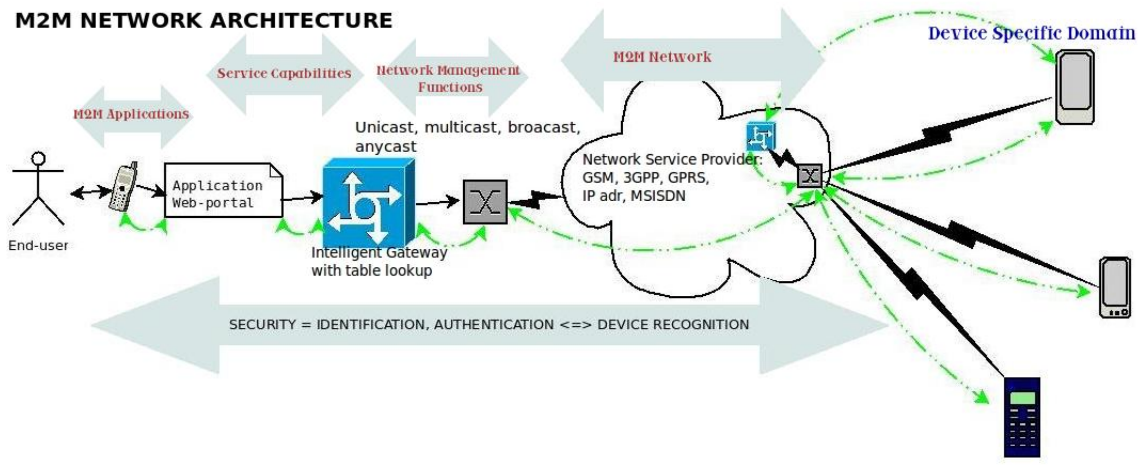

M2M NETWORK ARCHITECTURE
M2M Applications
Service Capabilities
Network Management Functions
M2M Network
Device Specific Domain
Unicast, multicast, broacast, anycast
End-user
Application Web-portal
Intelligent Gateway with table lookup
Network Service Provider: GSM, 3GPP, GPRS, IP adr, MSISDN
SECURITY = IDENTIFICATION, AUTHENTICATION <=> DEVICE RECOGNITION

Figure 6: An overview of M2M network based on the ETSI standard TS 102.690

to the *Core Network* via an *Intelligent Gateway*, which is a linking point between an M2M device to another M2M application.

The main modules described in standard ETSI TS 102.690 are:

M2M Applications
The M2M application is deployable to a device. The application consists of interfaces with a set of *service logic*, that enables the access to the *Service Capabilities*.

Service Capabilities
This module has a set of open interfaces with the capabilities for accessing the *Core network*. These also include how a network can be hidden, optimization of other network applications. Plus, to ensure that there is a M2M Gateway Service Bootstrap that detects new devices based on the SIM card. Other capabilities that can be found in this module are data aggregation, data storage, and message transmission via multicast, unicast, broadcast and anycast.

Network Management Functions
This module provides a set of functionalities for accessing *Access Network, Transport Network and Core Network.*

i) The *Access Network* allows a M2M domain specific Device, to communicate with a Core network based on technology xDSL, satellite, WLAN, WiMax or any technology that is available today.

ii) The *Transport Network* allows the data transport mechanism within that network and applications domain.

iii) The *Core Network* provides IP connectivity, interconnection and roaming using technologies GPRS, GSM, GSM-R and so on. These can be provided by the IP Service Provider Network.

M2M Network Area

This module provides connections between M2M Gateways and M2M devices. The connectivity can be based on IEEE 802.15.X, ZigBee, Bluetooth or local network (LAN). It can also be based on applications with the role of M-Bus, Wireless M-Bus and so on.

Specific Device Domain

This module consists of device with M2M applications that uses the capabilities and the network functionalities. This means that a device can connect directly to an *Access Network* or *M2M Network Area* through the M2M Gateways.

Communication and Connectivity Properties

The M2M Service Class that implements *Network Mobility (NEMO)*, will define the *"Mobility"* property of a device. If a device is in motion, then, the value for its *"Mobility = Yes"*. Otherwise, it is set to *No*. The Service Class is important in communication and connectivity manner, because it relates to Messaging in the aspects of the cost, complexity and performance of network.

In addition, the Service Class also defines the *Security* properties. The following are the minmum requirements:

1. *Authentication* between M2M devices and applicaitons.
2. *Integrity mechanism* for transfering of information within M2M services to provide the best-effort. This relates to delay factors such as when information is being delivered at a high limit with maximum delay, when using the store-and-forward mechanism according to the network loading and priority settings. Last but not least, how to handle the delay of real-time. These factors should be taken into account in advance before given the maximum average and peak data rate in a M2M network bandwidth.
3. *Confidentiality* of end-to-end encrytion. The reason is to guarantee if the information is lost during the transmission over the networks. As well as protecting source and destination localization.

4. *Device integrity* that checks when a M2M device sends a request to M2M service.
5. *Persistence* in etablishing of network connection.
6. *Confirmation* in *Transaction Control* manner. These include commit, roll-back or locking the data sets.
7. *Anonymity* in the way, the identity and localization of the requestor remain hidden when accessing the M2M network.

2.3 Network Technologies

2.3.1 GSM-R

There has been a lot of changes for the mobile telecommunication companies since 1980 when GSM was first introduced in the market [1, 17]. However, GSM was not adapted to the railway domain before 1992. Moreover, in parallel with GSM, there are also other technologies such as TETRA[29], which has been used and is available in the market. TETRA implementation is based on open standard ETSI. The technology enables voice and data transmission in digital mobile radio, but only within a *closed private user group (PMR).*
Some years later, UIC introduced of new technology - GSM-R. In 1994, this technology can be implemented and adapted. The reason for this is that there is demand for a technology applicable to a specific domain, which is for the railway. Establishment of GSM-R is also based on the standards from EIRENE, MORANE and ETSI. This also must be able to adapt in the European Union. Moreover, interoperability is the key for common European railway network [18].

According to the specifications as defined by EIRENE & MORANE for GSM-R is that must first exist a basic implementation on a national basis, which supports the national requirements that this country has. And not least, it is equally important to consider these requirements in relations to the *business model.*
Figure 7 below shows the major networks and the dependency that can adapt to the country, where GSM-R is the main network for *interoperability.* This also means that all members of the EIRENE MoU should also implement the same in order to ensure and facilitate inter-connection and inter-work between the countries.

[29]TETRA Critical Communications: `http://www.tetramou.com/tetramou.aspx?id=44`, last accessed 9. Sep, 2010

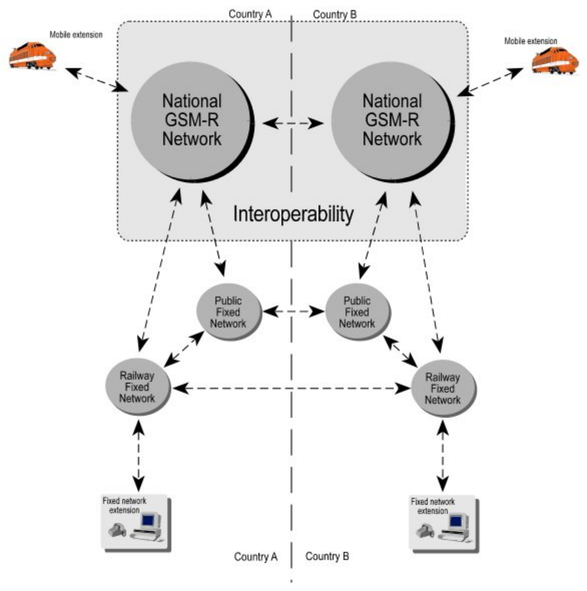


Figure 7: National network and interconnection. Copy from [8]

2.3.2 GSM as The Basis for GSM-R

Realization of GSM-R network is primarily dependent on the existence of a GSM network in the country. Because GSM is the public cellular technology and can therefore be used as the basis radio bearer.
According to the procedure [8] for GSM, there are three different implementation options that can be used in the national network deployment. These are:

1. *Public GSM* network. This is a *public* network, which implies the involvement of a third party such as telecommunication company. It also means that there is some limited freedom for railway operators. This means, in connection with the design of GSM coverage, because the network is outsourced by the *public network operator*.
   Moreover, some *public network operators* do not yet support the service requirements for EIRENE specifications which have been submitted to the railway domain. This is considered a hindrance to *interoperability*. Just to mention, *public network operators* must provide some form of a special solution that can still provide coverage, even in tunnels or when GSM coverage is breached. Anyway, the train plant operators must be taken into account, that many network operators do not offer a secure and safety voice communications over radio signals at low cost. Something that is not in line with their business model.

2. *Private GSM* network. In this network model, railway operators have complete freedom in provision of network designing. It also implies that the system requirements and system functionalities can be implemented and fulfilled. Although, there is more freedom. There is another factor that railway operators have to take into account. It is to ensure a proper operating frequency band when the railway operators are to allocate the frequecies. The allocation also means that it must allow for expansion or extension of GSM network to GSM-R network within their acceptable budget framework.
   Furthermore, railway operators must have a well thought out plan, and the determination of which areas are most needed for the radio coverage. How these base stations are installed, the number of installation that is needed. And not least, the performance level of such systems and how to maintain them. In addition to these, the railway operators also must acquire the necessary equipment for GSM and GSM-R networks. So that, these network elements enable the inter-connection between the country's borders.

3. *Hybrid GSM* network. This network model is a combination of *public* and *private* network models. Here, the railway operators have more choices. First and foremost is the implementation of a completely *private GSM* network over a specific area, while introducing the *public GSM* network over another area. However, there are some measures or issues of *hybrid GSM* network implementation. These are such as:

   a) equipment compatibility in relation to the existing GSM network

   b) does *public network operator* supports the railway applications

   c) who will be responsible for maintenance of the equipment

   d) who has the obligation to the performance of the networks

   e) what is the cost of providing the various services of the railway, and what are the profits.

   Otherwise, allows the *public network operator* to build and owns the GSM/ GSM-R network and its infrastructure. The result is that they have complete control over these networks.

   Ultimately, the best provision of this model can be considered to make use of the *private GSM* network in the most crowded areas, and instead use *public network operators* in areas with the least traffic, and in areas where the implementation costs are minimal.
   When all the aforementioned points are resolved, railway can then think further on a seamless integration solution between existing GSM network with

GSM-R network, as a further extension addressed to national and international interoperability.

The ETSI standard for GSM also defined implementation methodology for encryption and authentication algorithms. There are A3/A8 and A5/3 crytography, which is suitable for SIM card-based implementation. The A5/3 algorithm ensures the private voice communication in both GSM and GSM-R networks. More details about this subject, its architecture and protocols, are discussed in the article [19].

As the Figure 7 shows, a GSM network can be either fixed or mobile. The following describes these two methods:

(A) *Fixed network* is a network of switches. These switches are used to route calls to the correct called party, and act as termination points in the interconnection between networks, which every railway has its specific systems and network links such as management centre for performance monitoring, fault management, management of subscriber, and so on.
In the fixed network, one can build an infrastructure where resources can be shared across the network using VPN. The fixed network can be applied to *private*, *public* or both networks. This is a choice that railway must take the necessary decision.

(B) *Mobile network* is a network that is used terminal equipped with radio communication systems. Equipment can be firmly fixed in the train, but is mobile in the sense that the train with this equipment is in motion. Otherwise, it may be that the mobile devices as portable handheld devices that drivers have with him. To prevent unauthorized devices to access the network, or if the equipment is stolen. The railway can use a system called IMEI.

GSM/ GSM-R Infrastructure

The main elements [20] are present in the GSM and GSM-R network, which is also illustrated in the Figure 8. These elements are:

i) A *Base Station Sub-System (BSS)* composes of a *Base Station Controller (BSC)*, and a set of *Base Transceiver Stations (BTSs)*. To check a specific BTS, it is assigned a *transceiver ID number*, where this can be traced out in the BSC.

ii) *Operation and Maintenance Centre (OMC)* is the component that is takes care of monitoring and maintenance of the mobile network.

iii) *Mobile Equipment (ME)* is the unit with interfaces to the BSS component.

iv) *SIM* is a small card with a unique ID. The card contains information about a single specific user or subscriber. SIM links to ME and combines to form a *Mobile Station (MS)*.

v) *Network-Sub-System (NSS)* is a system with interfaces to the BSS with the intent to etablish the connection in a GSM/ GSM-R interface. In the NSS system, there are switches that are responsible for calling.

vi) *Billing Platform* (optional) is a platform for billing if the NSS system is used, for example, send the bill/ invoice to the subscriber when the user used the GSM service.

vii) *Fixed links* are used to establish the connection between the various components of a mobile network, for example, to connect the BSC to the BTS, NSS to Billing Platform, and so on.

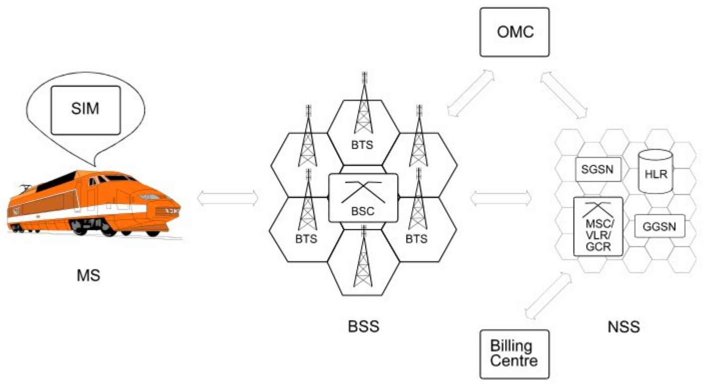


Figure 8: An overview of GSM architecture. Copy from [20]

In the GSM infrastructure, a component named *Group Call Registers (GCRs)* is used to record and store the information. It aslo records voice broadcasting and other services such as group call. Network infrastructure has also *Short Message Service Centre (SMSC)*, which enables the application services such as SMS.

It is also possible to use the system with the SIM cards based on a GSM network. Here, the Shepherd [21] also provides support for SIM cards.

When it comes to establishing a pure GSM-R network, a group name ERIG (established in 1999) has defines a standard for GSM-R. This group is also responsible for the implementation of GSM-R, such as counseling and support procurement of GSM-R systems. This also involves setting up the network.
The advantage of a GSM-R network is that, it implements a complete ERTMS

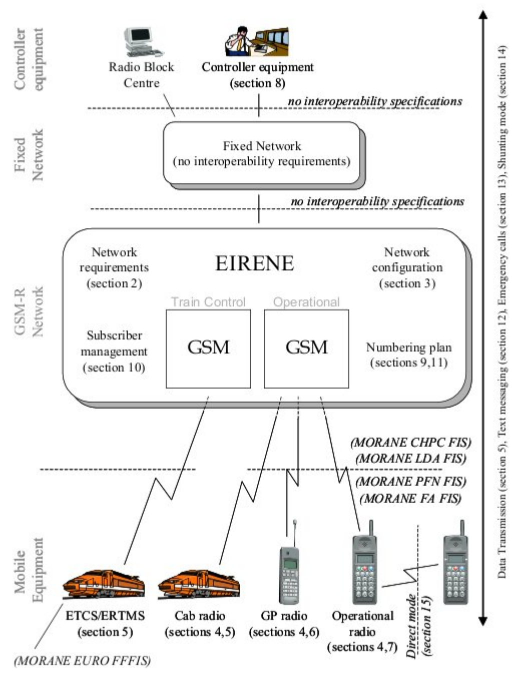
system, which is specifically designed and suitable for railways, where the sensors have the central role.

Today, one can find some companies that have implemented GSM-R in their business model, for example Nortel GSM-R company. Their products have reached the market, and a number of countries, such as Germany, France and United Kingdom have adopted the technology and put it to use in the railway.

2.3.3 GSM-R Network as Important Interoperability

As mentioned in the previous section, in order to complete the implementation of the international GSM-R network. It is dependent on that it must first have been implemented the national plan, in order to achieve *interoperability.* There are a number of measures that must be taken into account, for example systems for logging and monitoring, equipment compatibility, as well as systems for data and signaling that can operate and function optimally.

According to the specification for the System Requirements applicable to the GSM-R network, which also applies to the *Mobile Equipment* as it is shown in the Figure 9 This is a common solution for international co-operation not only between the



Controller equipment
Radio Block Centre
Controller equipment (section 8)
no interoperability specifications
Fixed Network
Fixed Network (no interoperability requirements)
no interoperability specifications
GSM-R Network
Network requirements (section 2)
EIRENE
Network configuration (section 3)
Train Control
Operational
Subscriber management (section 10)
GSM
GSM
Numbering plan (sections 9,11)
(MORANE CHPC FIS)
(MORANE LDA FIS)
(MORANE PFN FIS)
(MORANE FA FIS)
Mobile Equipment
ETCS/ERTMS (section 5)
Cab radio (sections 4,5)
GP radio (sections 4,6)
Operational radio (sections 4,7)
Direct mode (section 15)
(MORANE EURO FFFIS)
Data Transmission (section 5), Text messaging (section 12), Emergency calls (section 13), Shunting mode (section 14)

Figure 9: Networks of interoperability. Copy from [20].

country's borders, but also between networks. This *inter-network* is based on two routing principles:

1) *Numbering plan (NP)* is number based, that can be used in two ways:

   (a) The first is to use so-called *Functional Number (FN)* and is based on *Train Running Number*, which is a set number that is reservered for a single railway domain. Each train driver is assigned a unique number. This number represents both railway belonging, and train driver. This number is therefore used as identification on the train's driver, and is identified every time the driver performs a call to the train controller.
   The advantage of this method is that if the locomotive or the engine is replaced, the FN will remain unchanged. The driver could still call and receive calls in a GSM-R network. If the NP is used in a private network, the railway should allocate these unique serial numbers, for both fixed and mobile networks from the *National Administration.* Noted, the *public network operator* does not use FN number, they used either IN, VPN or *"Follow me"* method instead. This means that this method does not use *Subscriber Number (SN).*

   (b) The second method involves a third party, which is *public network operator* to route the calls. Therefore, it uses regular phone number based on MSISDN number. This method uses the *Subscriber Number (SN)*, which is translated into the FN nummer, where the FN can be traced from the routing database where it has been registered. According to the procedure [8], when it comes to emergency calls, it uses a set of predefined short codes to get faster call.

2) *Routing* is the second principle, which allows users to operate in different networks and inter-connections by means of routing. This solves two problems:

   (a) Solution 1: If a train is in its *home network* and makes a call to other country, then the first thing that happens is the national call is being dialed. A FN is looked up in routing database at the *home network*, and is being used for translation. Then, a new forward call is dialed to the home network. If it is a reply, the *home network* uses the FN number to route the call back to that mobile device/ train. This is illustrated in the left side of the Figure 10.

   (b) Solution 2: If the train is in motion and located in a foreign country. When a train driver performs a call, it establishes a GSM connection for resolution of its FN number where this number has been stored. The last step is that the *home network* uses FN number and translates to the relevant

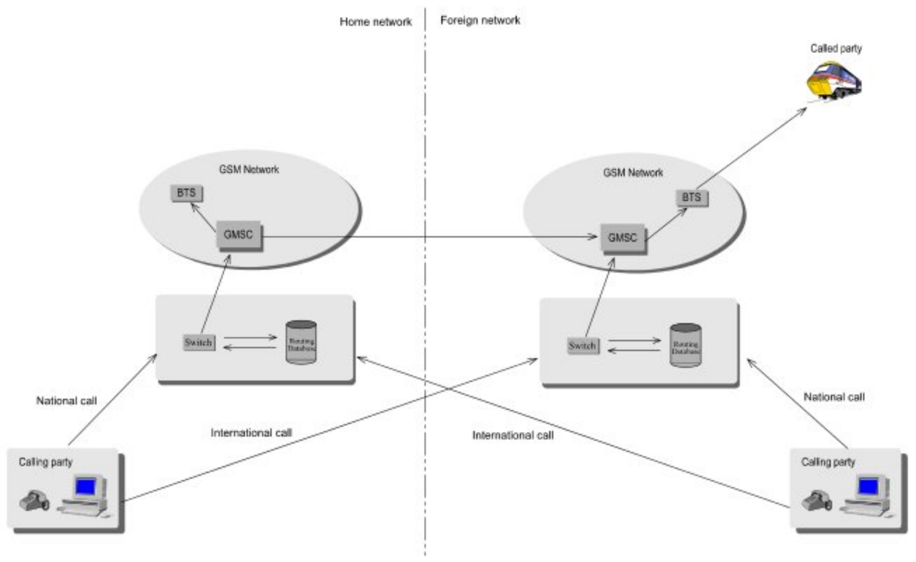


Figure 10: Call routing operations. Copy from [8].

mobile SN number. Then, performs a new international call to that mobile device/ train. This solution is illustrated in the right side of the Figure 10.

Three options for translation

a. An *Intelligent Network (IN)* is defined by EIRENE and uses the EIRENE Network Access Number instead of FN number.

b. *HLR* uses the *"follow-me"* service. This option uses FN number for routing to the public networks.

c. *Switch with databases.*

GSM-R properties

Characteristic for GSM-R network is the use of digital radio channels for voice and data communications, so that different railways can operate and communicate with each other in a GSM-R network. To achieve this seamlessly without any interruption in hand-over of ongoing calls of roaming mobiles, there is a need for:

1. The interfaces for managing *physical interconnection* to a gateway, called

the break in / out points of the network. So it is possible to choose to use a private network or public network.
The advantage of using a private network with a NP, is that it will provide the greatest degree of flexibility and coordination between the national railways. When it comes to public networks, one can use VPN to a public line network. This means that it uses the network interfaces and low level protocols, so information can not longer be regarded as private.
The intention here is that it is used *"public switch"* with interfaces and signaling protocols for setting up the call, route the call and information transfer.

2. The interfaces for *logical interconnection* based on the protocols to enable message exchange. Not least, the recording of the train's FN number in a so-called Gateway Switch, and the storage of this number in the data storage. Later, by looking up a table whenever there is a need to establish a further call.
   Registration is necessary so that information can be exchanged between the Gateway Switches. While, the de-registration can be done either manually or automatically on the train if the train is in operation. Here, the driver must register in any GSM network as the train passes through. It is therefore a need for a positioning system such as GPS[*] that can detect the train's position, so the train can be tracked when it is about to leave or arrive to the network.

   Finally, to achieve best-effort and optimal exchange of messages, it needs to use international FN number recorded and be translated in the Gateway Switches. Here, it is dependent on a number of measures to achieve best-efforts such as the solution has been selected for the FN number, what protocols have been used, and what format the data exchange has.
   If a driver performs a call within his own railway network, or when receiving a call from a calling party that is outside its network, only the need for a national FN number as an internal number. Then, it will be translated into mobile MSISDN number for use on international connections in order to establish the call.

GSM-R Frequencies

In 2005, a agreement was made between the UIC and Frequency Management Working Group on a common frequency range to be used in the European railway. PAN-allocation is shown in the table 1 below.

[*]See section 2.4.1 about GPS

Table 1: Frequencies allocation for GSM/GSM-R [8].

Technology	Down-link	Up-link
GSM	935 - 960 MHz	890 - 915 MHz
Extended GSM	925 - 960 MHz	880 - 915 MHz
GSM-R	921 - 960 MHz	876 - 915 MHz

Furthermore, each individual national rail network is responsible for allocating the exact frequencies it needs, and on what type of network, such as whether it is private, public or hybrid.

Radio coverage

The table 2 below shows the probability of coverage from ETCS, where the boundaries are to achieve 95% coverage in a distance of 100 meters. Here it is estimated a loss of 3dB between the antenna and receiver.

Table 2: Radio coverage for GSM-R [8].

Coverage level	Speed	Description
38.5 dBμV/m (-98dBm)		For voice and non-safety critical data
41.5 dBμV/m (-95dBm)	$\leqslant$ 220 km/h	For online with ETCS at level 2 and level 3

Requirements

The main requirements demanded by a radio system is that there should be reliable coverage between different systems, applications. And not least, the various mobile devices can co-operate in order to work properly. If the radio system is not physically located within the range, this will affect the communications infrastructure. Besides, it will also have an impact on the operating object and its security [8].
There are two methods that can be used to secure communications:

(A) To use the *roaming* mechanism. Here, the train can be allowed to register itself with its national FN number on a GSM-R network, when it is in motion and operates in another GSM-R network. The rest is handled seamlessly by the train's home network that has a lookup table for all trains' international FN numbers.

(B) The other possibility is *non-roaming.* This applies to the train that has registered in one national home network, which will run to another national home network where the train temporarily registered. The solution in establishing a connection here is to use international FN numbers between these countries.

2.4 Sensors for Embedded Systems

This section describes briefly the various sensors that are recommended in the railway. This thesis is primarily about the SunSPOT sensors.

2.4.1 GPS

GPS dates back to 1959[30]. The U.S. military army had a strong requirement for a navigation system. Several years later, the technology expanded so that it could also be used in civilian, and now also well known in the commercial telecommunications. It is especially used in mobile devices. But, it is also used in other contexts such as navigations on highway, waterway, aviation as well as emergency.
To avoid signal interference, separate signals [22][31] have been implemented, so that the GPS signals for military would not be the same as the GPS signals for civilians. Nevertheless, to ensure compatibility with legacy GPS signal systems that are already existed.

GPS is based mainly on the time calculation of the radio signal among 24 satellites at a distance from orbit to Earth. This is the most accurate technology in terms of geographical positioning. In order to use GPS, the equipment must have built-in GPS sensor.

Brown et al. explain that the GPS consists of three main components [23], and as shown in Figure 11:

1. The *space segment* consists of 21 satellites and three spares. It utilizes the distance values between 4 different satellites that are closest to the user, then a complex calculation to determine the exact position the user is located in. The result is then shown to the user.

2. The *control segment* is the master control station. This station is connected with several other stations that monitor, track the satellites and collect data

[30]A Guide To The Global Positioning System (GPS), `http://support.radioshack.com/support_tutorials/gps/gps_tmline.htm`, last accessed 9. Sep, 2010

[31]Q&A: `http://www.ngs.noaa.gov/FGCS/info/sans_SA/docs/GPS_SA_Event_QAs.pdf`, last accessed 9. Sep, 2010

from each satellite. In this way, the exact location is calculated.

3. The *user segment* is equipment segment, where the equipment is equipped with a GPS sensor so that the equipment can receive GPS signals.

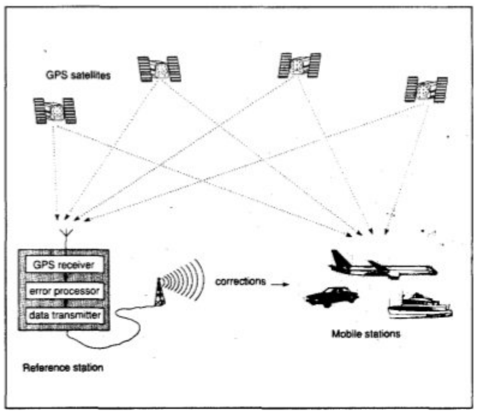


Figure 11: GPS concept. Copy from [24].

The reason for mentioning GPS here is that the GPS sensors can be placed at various locations in the wagon to a train, to trace the train's position. Something that is relevant to JBV.

2.4.2 The SunSPOT Devices

Figure 12[32] below shows the different devices contained in a Sunspot Development Kit (JDK). The kit provided by Sun and developed at Sun Labs and tested on various operating systems. SunSPOT is suitable for application areas such as robotics, surveillance and tracking. The main units are Sunspot devices with embedded sensors and base station. Each Sunspot has a so-called eSPOT with battery, while the base station is not equipped with battery and must be powered from the host computer via an USB cable.

An overall view of a sensor architecture is shown in Figure 13 [25]. The unit is equipped with the following boards: processor, sensor, free-range radio transmitter and battery. The eSPOT consists of:

i. A rechargeable LI-ON battery

ii. A motherboard contains the CPU, memory, power management circuits, radio transceiver and antenna, battery and daughter board connector

[32]Picture from SunSPOT World: `http://www.sunspotworld.com`

Figure 12: SunSPOT Kit

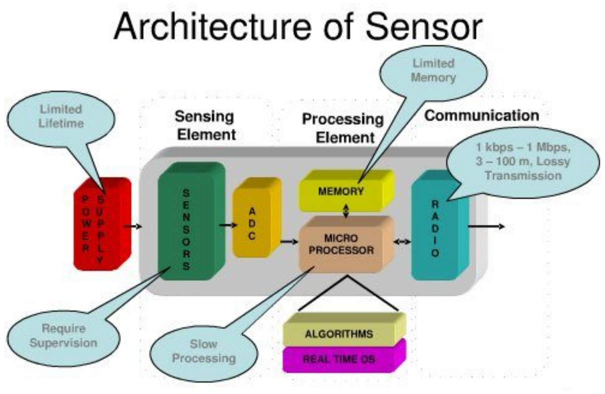


Figure 13: Architecture of a sensor mote. Copy from [25]

iii. eSPOT daughterboard is used to connect to the main board, and acts as a slave when communicating with the motherboard. There are also flash memory for storing configurations. This also has a flash memory for storing the configurations.

iv. eDEMO board contains Atmega88 processor, flash memory, light sensor, temperature sensor, accelerometer, 8 LEDs tri-color, and 2 switches.

Sunspot runs on a virtual platform called Sqwauk with Java ME platform. This platform is useful for developing applications for the ubiquitous devices. The advantage of the squawk is that it can host multiple applications simultaneously without underlying an operating system.
The Sunspot does not need to run the operating system, it need only JVM that runs on bare metal, and executes directly out of flash memory.
Stack-boards composed of specific sensors and actuators such as accelerometers, light sensor and temperature sensor. But is also has push buttons and I/O pins. Figure 14 [25] shows the hardware components of a processor board:

- ◇ 180MHz for 32-bit ARM920T core processor with 512K RAM and 4M Flash, runs on Squawk
- ◇ 2,4GHz based IEEE 802.15.4 radio (radio ChipCon TI CC2420) which is integrated in the antenna
- ◇ USB interface for connecting to a host computer
- ◇ 3.7V battery (720 mAh)
- ◇ Sleep mode (32 uA)

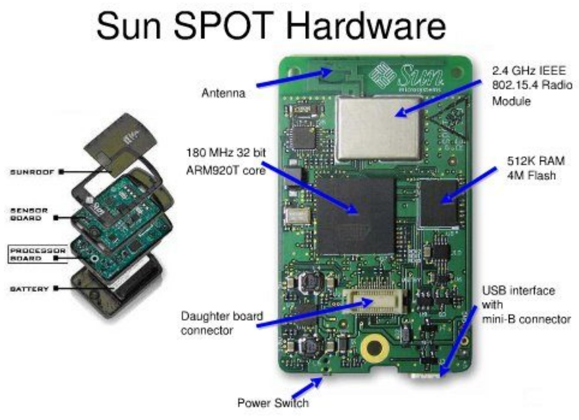


Figure 14: Sunspot processor board. Copy from [25].

Fig 15 [25] shows the hardware components of a sensor board:

- ◦ 3 axis accelerometer (2G/6G)
- ◦ temperature sensor
- ◦ light sensor
- ◦ 8 tri-color LEDs
- ◦ 2 push-botton control switches
- ◦ 5 digital I/O pins
- ◦ 6 analog inputs
- ◦ 4 digital outputs

Temperature sensor
Chip-type is ADT7411 sensor that measures temperature with ADC. ADC is integrated into eDemo, and can measure temperatures between -40℃ to +125℃.

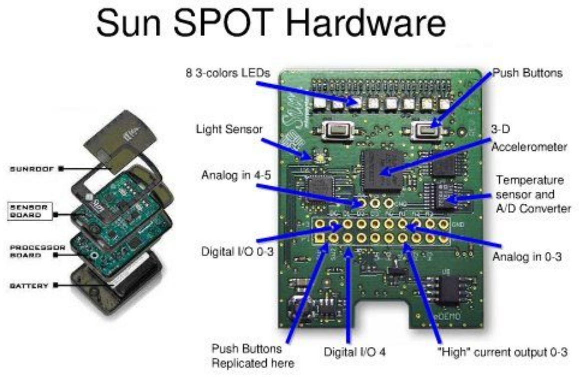


Figure 15: Sunspot sensor board. Copy from [25].

Accelerometer sensor
3-axis accelerometer of the type LIS3L02AQ, designed by ST Micro Systems and is in eDemo Board. This sensor can measure the x-axis, y-axis and z-axis in the direction up and down with the value either $\pm$2G or $\pm$6G. When the Sunspot is at rest, it measures x = y = 0 and z = 1G

Light sensor
Light sensor is of the type TPS851, designed by Toshiba. The sensor can measure the voltage between 0.1V (dark) - 4.3V (light), and converts the voltage to the brightness of Luminance (lx)[33] [34].

Battery lifetime
Battery life depends on whether the CPU is active and is under processing. Normal life of the active CPU is about 7 hours, while in deep sleep, it may have battery life of more than 37 days. If all the sensors are used, the battery will have a maximum of 3 hours lifetime.
All sunspot's batteries are rechargeable, apart from base station that needs power through the USB connection to the host computer.

Radio communication

[33]For more details, refers to `http://www.astronomynotes.com/starprop/s4.htm`. Published by Nick Strobel on 2. Nov, 2010, last accessed 24. Nov, 2010

[34]Powerpoint "Luminosity" by Clinton Raymond `http://www.astro.umass.edu/~rdubois/Astro%20499Y/Class%209%20Student%20Presentations/Student%20Slideshows/Luminosity_Clinton.ppt`, last accessed 24. Nov, 2010

The Sunspot Library is using Java ME Generic Connection Framework (GCF), which also includes the CDC for both virtual[*] and physical[†] layout scheme, and provide interaction with the network through a set of protocols including HTTP, TCP, UDP, and the radio connection.
The sunspot uses CLDC 1.1 to open the radio connection between the spots when a client performs an I/O (using instances of the Connector connection factory) to open a direct connection. After opening a connection, the client can do one of these options:

- Using *Broadcast* to transmit messages between Sunspot that is within range. There are two types of exchanges in *Broadcasting*, it is either with the *Radiogram* or *Datagram*. Choice of type depends on which mode a base station is in. It is allowed for *Broadcasting* from 2 hops, but broadcasting in this way can also be received by other units that are in the same PAN.
  The disadvantages of broadcasting are, first, a SPOT can not receive broadcast data directly, unless it must have opened a server connection. Second, the data may not be delivered, or data is simply ignored by the spots, if this SPOT has already received packets. Moreover, the transfer of data is unreliable.
- Using *server connection* to listen for a connection and request from other Sunspots.

The Figure 16 [26] below shows the various protocols stacks in the Sunspot. The

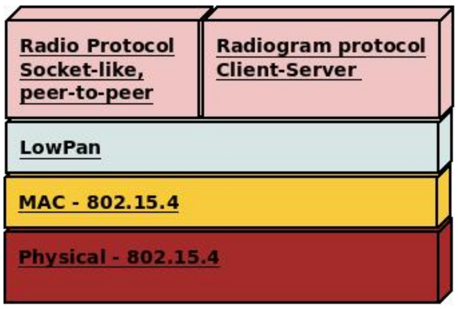

Radio Protocol Socket-like, peer-to-peer
Radiogram protocol Client-Server
LowPan
MAC - 802.15.4
Physical - 802.15.4

Figure 16: IEEE 802.15.4, 250 kbps OTA. Reproduced from [26].

radio stack has access to the MAC interface and physical layer, through a protocol 802.15.4 (PAN) in the 2.4 GHz ISM band. An IEEE MAC address is a 64 bit identifying a spot. Address format is 0014.4F01.XXXX.YYYY, where the last 8 digits are the identifier. Here are the ports between 0-32 reserved for sys-

[*]See testbed on Chapter 4.1.1 about Virtual setting
[†]See testbed on Chapter 4.1.2 about Physical setting

tem access, while the port of the node between 32-255 can be used by applications.

IEEE 802.15.4 protocol

The standard used here for the Sunspot is protocol IEEE 802.15.4. This protocol is suitable for and can be adapted to the devices that have limitations such as memory and power.

The IEEE 802.15.4 has several properties that are listed in [27]. It includes:

1. *PHY 802.15.4* provides data services and management services to the physical layer. The main tasks here are to enable and to disable the radio transceiver and to ensure the link quality indication. This means that the link is stable and is sufficient so the receiver be able to receive all the packets. In addition, this should be able to discover and detect power, channel, and to clarify the channel to receive and send packets over the physical medium.

2. *MAC 802.15.4* provides data service and management service at the MAC layer. This includes the beacon management which ensures synchronization with the connected device. In addition, the MAC layer identifies the PAN and the structure of the super frame. And not at least, provide free channel access, control and approval of the framework as well as acknowledge frame delivery, and guarantee of the reservation time slots (GTS). Reference is made to articles [27], [28], [29] and online document[35] that might be interest to those who would like more details about the MAC layer.

3. *LowPan* provides packet fragmentation, while it also supports IPv6 packets. If one uses the IETF specification for intermediate layer which is described by Howitt et al. [29].

4. There are two types of protocol for radio links, which are both found in the Sunspot's library from packages *"multihoplib_common.jar"* and *"multihop_common_source.jar"*:

    (a) *Radiostream* is stream-based caching with (buffer). This protocol is a socket-like peer-to-peer and provides reliable communication between devices. Data is sent over the air (OTA) when an output stream buffer is full. It carries no automatic handshaking when it opens a stream connection. However, an acknowledgment will be sent from the MAC level of each data transmission between the two devices, and then waits for receipt of an ACK from the next hop. When the next hop is 0, the message has reached the destination.

[35]`http://www.ietf.org/rfc/rfc4944.txt`, last accessed 18. Nov, 2010

(b) *Radiogram* is a datagram-based and uses I/O between two devices. This protocol is used in a client-server communication model. It does not guarantee that the packages are delivered, nor that they are delivered in the correct order. When the datagrams are sent over more than one hop, the packets may be lost. They can also be delivered more than once. These are the disadvantages when using the Radiogram. However, datagrams sent over a single hop is guaranteed to be delivered.

The *"physical"* schema uses physical radio to send data packets, while the *"virtual"* schema uses the standard Internet sockets.

Development environments

In this paper, the author has used the Linux platform as a development platform, and uses the NetBeans IDE development tool for developing Sunspot applications. However, other platforms can also be used as Windows or Mac, which both have been tested by Sun Labs.
Before the development of an application, Sunspot tool called *SPOTManager Tool* must be downloaded and installed. The tool can be found at the project site[36]. When the tool is loaded, it can be installed *all-in-one* with development tools such as NetBeans IDE, JDK, SPOT SDK, Apache Ant. In this thesis used the NetBeans IDE 6.9.1, JDK 1.6.0_07, SPOT SDK version Red-100104. The advantage of SunSPOT SDK is that it has an emulator for creating virtual SPOT, and testing of applications on virtual SPOT.

The SPOTManager Tool

The *SPOTManager Tool* is Java WebStart Application, and must be installed on the host machine to develop applications for the Sunspot and the base station. SPOTManager has GUI interface with the following tabs:

- ◇ SunSPOTs tab: configure SunSPOTs and SPOT base station, as well as update software on SPOTs.
- ◇ SDKs tab: installing, update SunSPOT SDK version from Sun.
- ◇ Preference tab: conFigure network, setting up proxy, and polling of SPOT-Manager tool.
- ◇ Solarium tab: lauches emulator tool for creating virtual SPOTs, load and run deployed application.

[36]SPOTManager: `http://www.sunspotworld.com/SPOTManager/`, last accessed 4. Oct, 2010

◇ Console tab: displaying outputs.

Java Micro Edition (Java ME)
Java ME also called J2ME, is a development platform designed for applications that are suitable for small devices with limitations. This platform also supports the CLDC and CDC, where applications are developed and targeted at robust devices. More details about CLDC and CDC, can be found in Chapter 2 in book [30]
Advantages of Java ME is that it consists of a set of built-in interface that provides security, network protocols for both online and offline applications.

J2ME platform has two configuration methods. It is the CLDC (current version 1.1 in use) and CDC. The difference between them is that CLDC is used and is targeted on devices with memory limitations, which is aimed at applications such as widgets and network connection. Whereas, the CDC apply to applications on devices such as PDAs, set-top boxes, and Internet applications. The details of these configurations and their architecture will not be addressed in this thesis.

Sun recommends using J2ME as the base platform for application development for small devices, including Sunspots, because it provides flexibility and freedom to create standalone applications.

Today, many device companies, including telecommunications operators, focus on finding the best and latest solution to offer their customers. The result is a massive growth in applications, not just for wireless devices, but also setting boxes, Blu-ray players and many other embedded systems. This is an area in which the J2ME [30] is being used actively.

Sqawk
Squawk[37] [38] [39] is open source and has been written in the Java programming language. It is a virtual machine, and is a highly portable Java VM. Figure 17 below shows the architecture of Squawk. The advantage of Squawk is that it can run on bare metal instead of being run on top of the operating system. This means that applications can be isolated and be treated as application objects. This allows multiple applications running on the same *virtual machine.*
Squawk also supports *CLDC* 1.1 that facilitates connectivity to mobile phones.

[37]Squawk: `https://squawk.dev.java.net/`, last accessed: 18. Nov, 2010
[38]`http://wiki.java.net/bin/view/Mobileandembedded/Squawk`, last accessed 9. Des, 2010
[39]Labs project: `http://labs.oracle.com/projects/squawk/`, last accessed 18. Nov, 2010

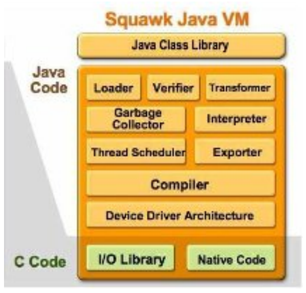


Figure 17: The architecture of Squawk. Copy from [25].

2.5 Telenor Objects as The Middleware Layer

Telenor Objects[40] [6] from Norway, have introduced a platform for *interoperability* and *integration* with the name *Shepherd*, that supports communication between devices and enable the diverse of application integrations.
This is an opportunity to create applications that can offer new services that aim to M2M devices. *Interoperability* here means that there is a freedom to choose the ubiquitous devices [31] and any applications (API) you want to connect to the Shepherd. It means so that the platform allows for heterogeneity in the integration of these devices and their applications. In short, Telenor Objects offers a management service for the integration of heterogeneous services via the so-called Shepherd® platform.

Currently, the Shepherd supports only a few number of sensors like GPS and RFID, but does not yet support the sensors from the sunspot. This thesis aims to develop a prototype for two-way communication between Sunspot nodes. The values obtained from the sensors is routed to the Shepherd platform either through the use of Coos endpoint (explained later in this section) instances, or via HTTP.

There is still a need to integrate multiple APIs to the Shepherd® platform, and enables the integration of diverse applications.

2.5.1 The Shepherd® Platform

The Shepherd® is a platform for Connected Objects (COs) [32]. This means that any the pluggable component can be connected, and be integrated in Shepherd® platform as a *Connected object (CO)*. In addition, the Shepherd® platform includes a set of other pre-defined APIs and services [33] that are shown in Figure 18. A number of services are offered, including:

[40]`http://www.telenorobjects.com/`, last accessed 18. Nov, 2010

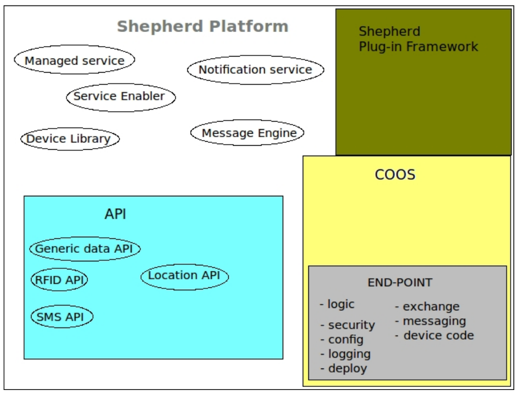


Figure 18: The Shepherd Platform

(A) *Service Management* for monitoring, device configuration, SLAs, and supporting.

(B) *Service Enabler* has a specific API that allows further access to other modules.

(C) *Message Engine* handles and secures the process of message flow, including capturing, processing, routing and storage of data in an environment.

(D) *Notification services* that inform about the status of devices and applications.

(E) *Device library* consists of interfaces for tools and services recognition.

Shepherd® platform consists of a set of APIs. Each API is an interface to a *Service Enabler*, which can interact with the interface of a specific device. For a device to *talk* to other devices, it requires the implementation of these APIs to the *Service Enabler*, so it can become integrated into the Shepherd® platform.

When an end-user has become a part of the M2M network. It can send the request direct to Shepherd in order to to access and retrieve the necessary information. The end-user can either do this via a web browser or via a mobile device, such as using the SMS service or WAP service. The technologies for exchanging messages that can be used here include XML [34], Semantic Web and Ontology.
Chart 19 shows an illustration of the interaction between the various components including nodes site, Shepherd as middleware layer and the end-user in a M2M network.

Pre-defined APIs

As mentioned earlier, Shepherd consists of a set of predefined APIs [35]. The most important are mentioned here is:

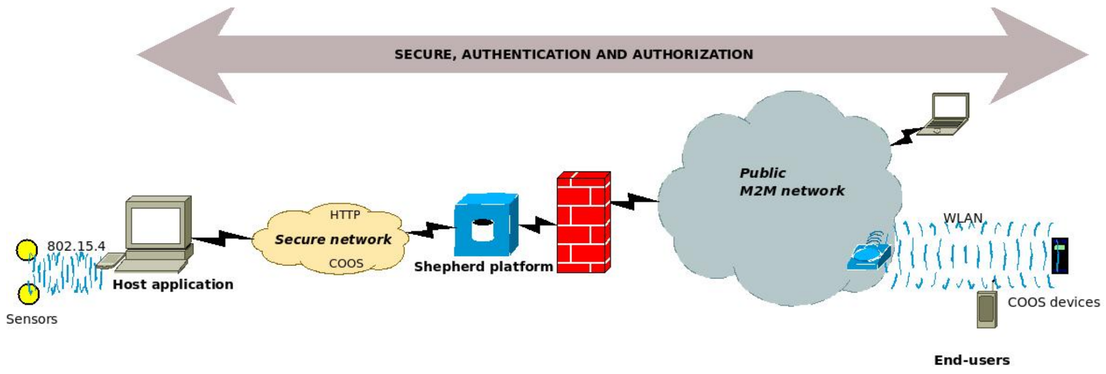


Figure 19: The Shepherd platform in M2M networks.

- *Generic Data API* allows communication between devices so that they can send, receive and subscribe to data. In addition, also allows real-time data exchange to and from Shepherd. This is particularly useful in connection with the Sunspot sensors.
- *Location services API* supports tracing location with GPS sensor integrated into the device, and uses the GSM network as communication network. This requires that the device has a Telenor SIM card.
- There are other APIs such as SMS and RFID. These will not be explained here.

All integrated devices are registered in the *Device Library* in Shepherd. The platform does not yet support all the devices available on the market today. The task undertaken here is to create a prototype for integrating the Sunspot to Shepherd® platform.

2.5.2 Connectivity

There are two methods that can be used to establish a connection to Shepherd. It's either using *HTTP API* or *COOS API* shown in Figure 19.

HTTP Connection API

This mechanism establishes a direct connection to the Shepherd by using the HTTP protocol. With this method, it requires the development of the *HTTP API* of the object. Shepherd accepts both methods *POST* and *GET*. When the connection is established, the Shepherd sends a reponse code back to that object as a confirmation of success or failure of reception.

To be able to connect to the Shepherd, the "*device object*" is identified with an *Application ID* and an *Object ID*.

For more information about this method, reference is made to contact the project team[41].

Connected Objects Operating System (COOS) Connection API

Connected Objects Operating System (COOS) is an open source and has been written in Java. When using the COOS instance, the applications can connect to Shepherd in a secure, reliable and stable manner. In particular, it is important in this respect that eavesdropping by third parties is not possible when using COOS. Reliable in the sense that it is an M2M network, and communication between objects with COOS instance and the Shepherd will not be interrupted or delayed more than necessary. Thereby, ensuring a stable environment for the users and the applications.

It requires therefore, to develop an application using COOS, so this can applied to device so it can communicate with the Shepherd.

From a programming perspective, "*Connected Objects Operating System (COOS)*" is an application distributed in a container so that it can enable data exchange between the *object* and the Shepherd.
In COOS concept, every component that is integrated, and can be pluggable is called "*Connected Object (CO)*". This means that a *COOS* instance can have multiple Connected Object connected, and each *COOS* instance carries its own distinctive character. The Figure 20 (Copy from[42]) below illustrates the relationship between *COOS* instance and *CO*.

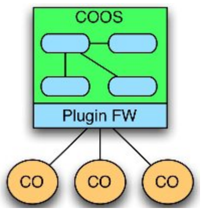


Figure 20: Coos instance with Connected Object.

[41] *http://coos.sourceforge.net/maven-site/1.2.2/team-list.html*, last accessed 8. Nov, 2010

[42] http://telenorobjects.onjira.com/wiki/display/coos/About. Published on 24. Aug, 2010, last accessed 8. Des, 2010

COOS has the following characteristics:

(i) Provide flexibility in the sense that communication between the various service components can be easily controlled and managed.

(ii) With model-driven development, provides an agile adaptation of new applications.

(iii) Ensures that the existing modules and application deployment is scalable on both horizontally and vertically.

(iv) Behave as an independent transport mechanism, which allows integration of various objects in the network.

(v) Can be used to form an independent network technology of message routing, because it provides a two-way message-bus that can initialize from both sides.

(vi) It provides the ability to assign priority to a message when it is delivered. If this message has higher priority than others, the message may pass by the message bus, while the lower priority message must yield.

(vii) Coos instance can be used with other components such as ActorFrame, or other modeling objects using UML.

2.5.3 An Overview of Integrations

This paper also proposes a solution for the integration of Sunspot with Shepherd via Coos instance. The Figures 21, 22 and 23 below show the possibility of implementing COOS, and the use of the *Ping* and *Pong* APIs[43] for additional flow of messages between the object and Shepherd.
Here, the *Ping API* and *Pong API* behave like the edges to enable message exchange within the messaging-bus.

A) Pong is an edge that listens for the incoming message, and sends back a response to the sender (Ping).

B) Ping is an actor who sends the message to the receiver (Pong).

The Figure 23 also shows two main roles that the COOS API may have as an endpoint (see definition on the next page). It is alternating between being the *Producer* or *Consumer* in relation to the message bus. In the sense that the message is either delivered to the message-bus or retrieved from the message bus.

[43]Ping and Pong projects: `http://telenorobjects.onjira.com/wiki/display/coos/Ping+and+Pong`

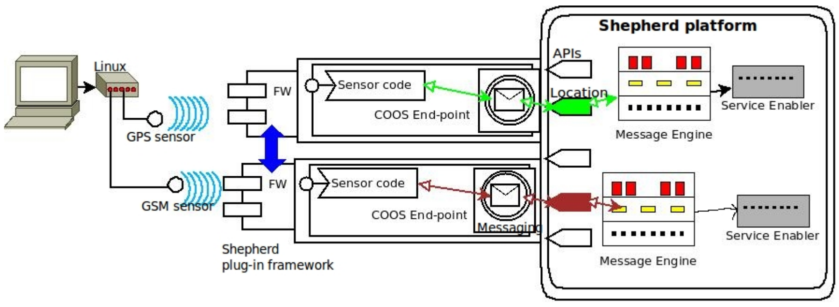


Figure 21: An overall view between the components.

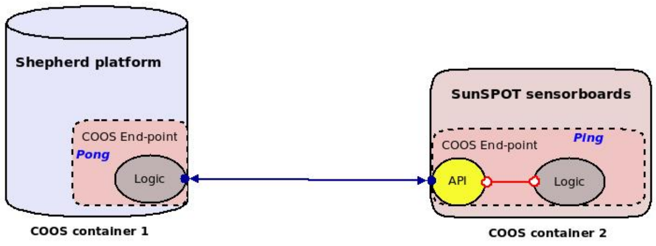


Figure 22: Interactions between Shepherd and Sunspot.

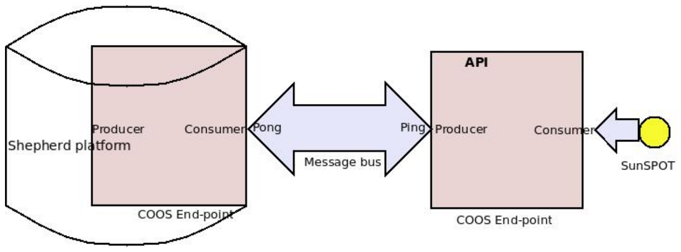


Figure 23: Communication path.

Then, in Shepherd that handles the process, including connection establishment with Generic Data Enabler, storage of messages or other necessary operations that are needed.

What is an Endpoint?

According to the definition from [44], the concept of an *Endpoint* is that it is an *inter-process* communication between the two parts, where each part is using an endpoint for the connection. In other context, an *endpoint* can for instance be an address with the URL/URI format, or it can be a *software entity*.

[44]Camel Apache: `http://camel.apache.org/book-getting-started.html`, last accessed 18. Nov, 2010

Chapter 3

3 Adaption to Norwegian National Rail Administration

This section describes the scenarios in which a train can be met during the journey in Norway's weather. Then, comes with the possible solutions that can solve problems for Norwegian National Rail Administration (JBV). At the end of this Chapter, illustrates the proposals referred through this Chapter on a solution with the integration and integrity between the various technologies, systems and actors

3.1 Scenarios with Sensors Integration

Use-case: JBV

Among the problems that the National Rail Administration is facing today, the most important include the weather conditions. It is important for the National Rail Administration to obtain the relevant information, which is necessary to maintain security on board, not to mention increasing numbers of passengers and decreases the number of delays.

Let us imagine that Sunspots can be firmly fixed in several places in the carriages so that they can monitor the conditions around them. Meanwhile, Sunspot also placed near the railroad where it suggests that the accident may occur. The Figure 24 below shows the trains with the various sensors integrated. So we have the following scenarios:

(I) *How can the railway headquarters know where the train is? How can they track the train's position or the train's location, so that the main office will be able to make its decision?*
In this case, the wagon should be equipped with multiple GPS sensors or technologies, that support the connection if the GPS sensors are repealed.

(II) *But, what if the train is in the tunnel and communication signals are not available, how can the main office know that the train is still in operation and safe in the tunnel?*
The best solution here is to use mobile devices that support GSM or GSM-

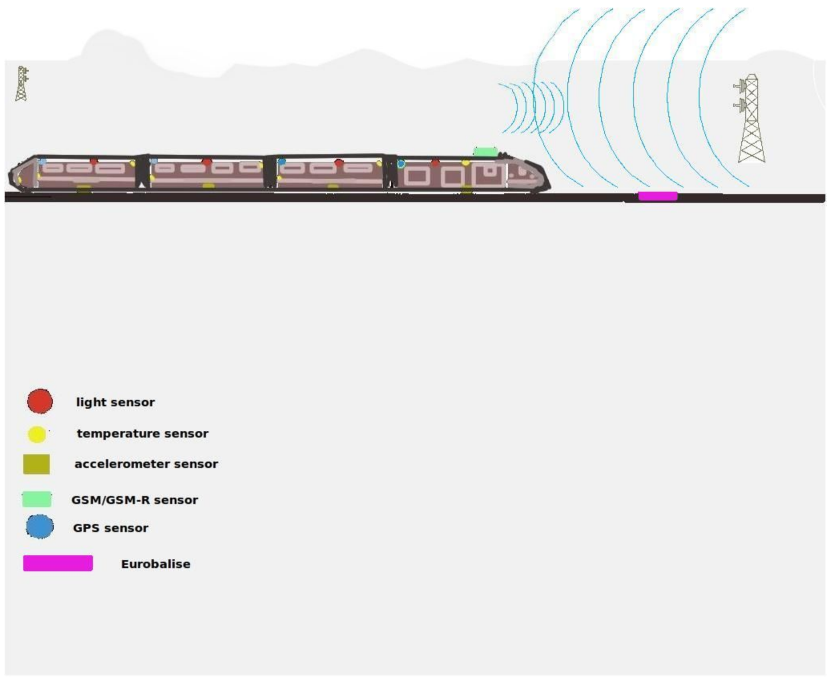


Figure 24: Sensor integrations for the JBV

R network* . These devices can also be installed in the train or handheld devices. Benefits of GSM and GSM-R network is mentioned in the previous Chapter. For example, to make use of voice and data transfer in either the private or public networks.

(III) *How can the headquarters detect what speed the train has? What must be done so that the train can run at high speed on free and flow path?*
Here, the National Railway Administration can implement an ETCS railway system as mentioned in the section 2.3.1 with ETCS level 2*. This system requires a GSM/ GSM-R network for the railway. The system also involves constant monitoring and the provision of the movement. ERTMS allows trains the speeds up to 500km/h, and guarantees a free path.

(IV) *How can the railway ensure that the railway network is reliable and data secure?*
Here, the National Railway Administration can introduce an M2M nettwork infrastructure with GSM/ GSM-R network to their systems. The National Railway Administration can use a hybrid network that involves the public GSM network, which can relieve some of the complex tasks, as well as to

*See section 2.3.1 about GSM/ GSM-R network
*See section 2.1 about ETCS levels

ensure that the relevant and necessary data will be available for the passengers. The Shepherd® platform is provided by Telenor Objects, which can be a good example for a co-operation.
Maybe, they will add support for text message (SMS), multimedia message (MMS), or technologies for E-mail solution for mobile devices over GSM-R network in the future. Possibly, one could also try out the so-called *face-tracker* technology for the railway as it has been implemented for the health domain[45] [46].

(V) *What about the landslide near the railway, how can the headquarters know that the train is in danger if the area can be prone to this?*
Here, one could use the accelerometer to measure the surface movements and constantly send these values to JBV for monitoring. If the abnormal parameters are detected, an alarm will be triggered that alerts the headquarters and informs the train driver about the area's dangers.
Accelerometer can also be used in other contexts, such as detecting the rockslide, skewed train tracks or train tracks are out of track, flooding nearby and so on. Therefore, it requires a warning to the train driver about this before the train runs on this track. This is illustrated in Figure 25.

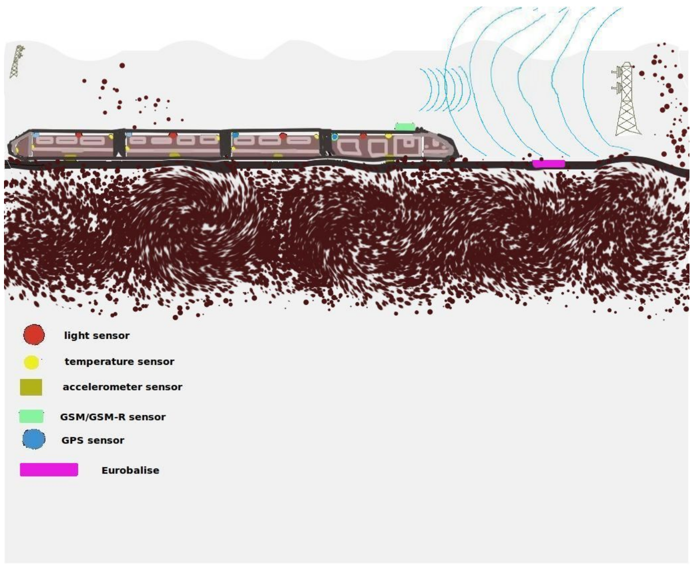


Figure 25: Detecting of earthshake or landslides along the trip.

[45]`http://www.dailymail.co.uk/sciencetech/article-1324278/New-software-brings-facial-recognition-technology-mobile-phones.html`, last accessed 9. Des, 2010

[46]`http://www.mobioproject.org`, last accessed 9. Des, 2010

(VI) *How can the train driver know that the last carriage is still in good condition during the journey? The train driver would also need to be informed about the condition of other train wagons when train is in movement. How can this information be obtained?*
To detect this, one can make use many different types of sensors. Here, only Sunspot sensors will be mentioned and used. In the worst case, a fire can occur without the train driver being aware of this. One can use the temperature sensor that can detect heat in the carriage. If the temperature exceeds a limit, then it is a good reason to check the wagon.

(VII) *Last but not least, how to detect the brightness of the wagon in terms of being "Eco-Friendly" and save electricity?*
Light sensors placed around the wagons can be used for this purpose. If the sensors detect daylight, the lights in the wagons can be turned off. The sensors can turn them on when they detect darkness, for instance at night or in a tunnel.

3.2 Components Integrity

As mentioned earlier, a wagon can have several different sensors as well as having more of the same type of sensor. Assuming that a base station can be mounted at the drivers compartment, it can receive the signals from the various sensors. Note that if one uses the Sunspot with 802.15.4 protocol, the distance between Sunspot be a maximum of 10 meters.
In the carriage, it establishes a kind of *Sunspot network*. The network will allow for the Sunspots to send the values to each other or to the next Sunspot which is nearby. So, the nearest base station is an *End Transmitter*, while all other Sunspot in the network act as message-bearers.

Imagine that the train is approaching a radio transmitter, where the communication link is GSM or GSM-R that can be used in data transmission. The values from the base station are forwarded to Shepherd® platform for storage. One can also imagine that the data values from sensors around the train, but located in the fields can be sent to the base station as well. The base station can forward the messages to the Shepherd.

When the end-users wish to obtain information about the train, they can turn to the Shepherd and submit request for it. Here, the end-user for instance is the JBV, other headquarters around the country, a home office computer or any mobile device that supports GSM/ GSM-R. This is illustrated in Figure 26.

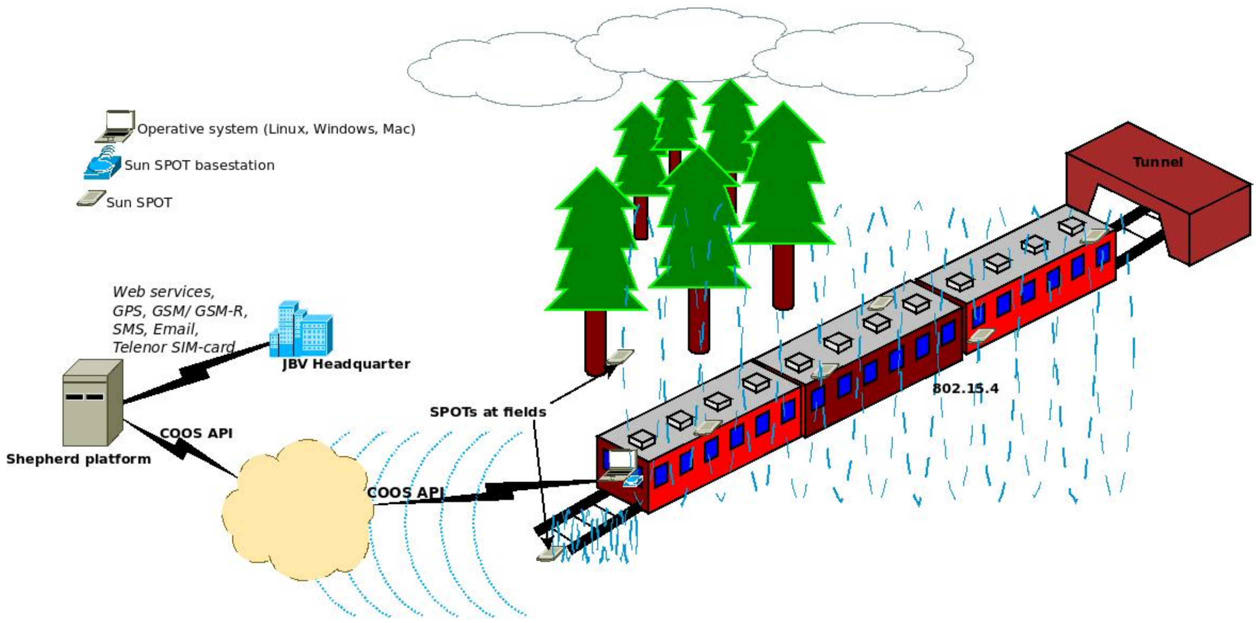


Figure 26: System implementation for JBV.

3.3 System Implementation Overview

As mentioned earlier, the Sunspots are mounted at different locations within 10 meters, forming an *ad-hoc wireless network*, where each train is equipped with a base station near the train conductor. One can use the method of message broadcasting between them, the so-called *Hitch-hiking* broadcast as mentioned in the article [36].

The advantage of using Sunspot is that it is possible to connect other types of sensors, such as GPS sensor in addition to temperature, light and accelerometer sensors. That is exactly what JBV need, which is to be able to track the trains.

Broadcasting and message exchange between them is wireless, while the base station has an API that supports GSM/ GSM-R technology, for example, using the GSM SIM card, making it possible to communicate with Shepherd. To establish connection with the Shepherd, it can be used *HTTP API* or *COOS API*. However, to achieve security, reliability and scalability, it is recommending to use the COOS instances in an M2M nettwork infrastructure. This thesis will use the *HTTP API* to establish a connection with the Shepherd.

In order to realize an M2M infrastructure, one can use the Shepherd® platform. Because, Shepherd is a dynamic system. This means that, the system is designed

and aimed to enable application distribution, not to mention an infrastructure of load balancing that ensure scalability of the distributions. In addition, data stored in the Shepherd is well protected by firewalls. Besides, there are mechanisms that ensure communication link and information retrieval. One of the advantages of using Shepherd, is that it also supports multiple access users / clients simultaneously. An opportunity for unlimited number of users in the future.
The optimum in this case is that, Shepherd should also support GSM-R network infrastructure in addition to M2M adaptation. The intention here is that, it may be advantageous for Norwegian National Rail Administration with a such infrastructure, and with a network that is specific to the railway domain. By this, is meant to realize the standard ETSI TS 102.690, which is designed for railway. And not least, ensure that there are devices that support the GSM-R with M2M nettwork infrastructure. This is an even better advantage.

For the railway, they can choose whether to use a public network or own their own GSM/ GSM-R network. A previous Chapter discusses the choice between these networks. If the railway will have a hybrid network, it can also consider using VPN as a solution for home offices.

For handheld devices, one can use a WAP solution if Shepherd provides support for this. The applications for mobile devices or handheld devices can be developed in J2ME[*] or J2SE,[37],[38] environment.

Web based applications can be developed to use the browsers to display data or statistics of measured values retrieved from the Shepherd, which are delivered by the sensors. These applications can be developed in the Java EE,[39] platform. This is relevant for the JBV headquarters, which one can offer their customers *online* information about the train. The technologies can use here are for instance, Web Services, Semantic Web or Ontology.

Figures 26 and 27 illustrated the scenarios described earlier in this chapter. The chart also showed the sensor integration on the wagons, and communication with Shepherd. Plus, the various end-users can retrieve information from Shepherd through an M2M network infrastructure .

This thesis starts with the implementation of a prototype for Sunspot sensors. Application for Sunspot sensors are developed in J2ME where the CLDC version 1.1 is used. It also developed an application for the base station is J2SE[47]. This application is the main entry point for the program. More details about these applications implementation are reviewed in the next Chapter.

[*]See section 2.4.2 about Java ME

[47]`http://download.oracle.com/javase/1.5.0/docs/relnotes/features.html`, last accessed 14. Des, 2012

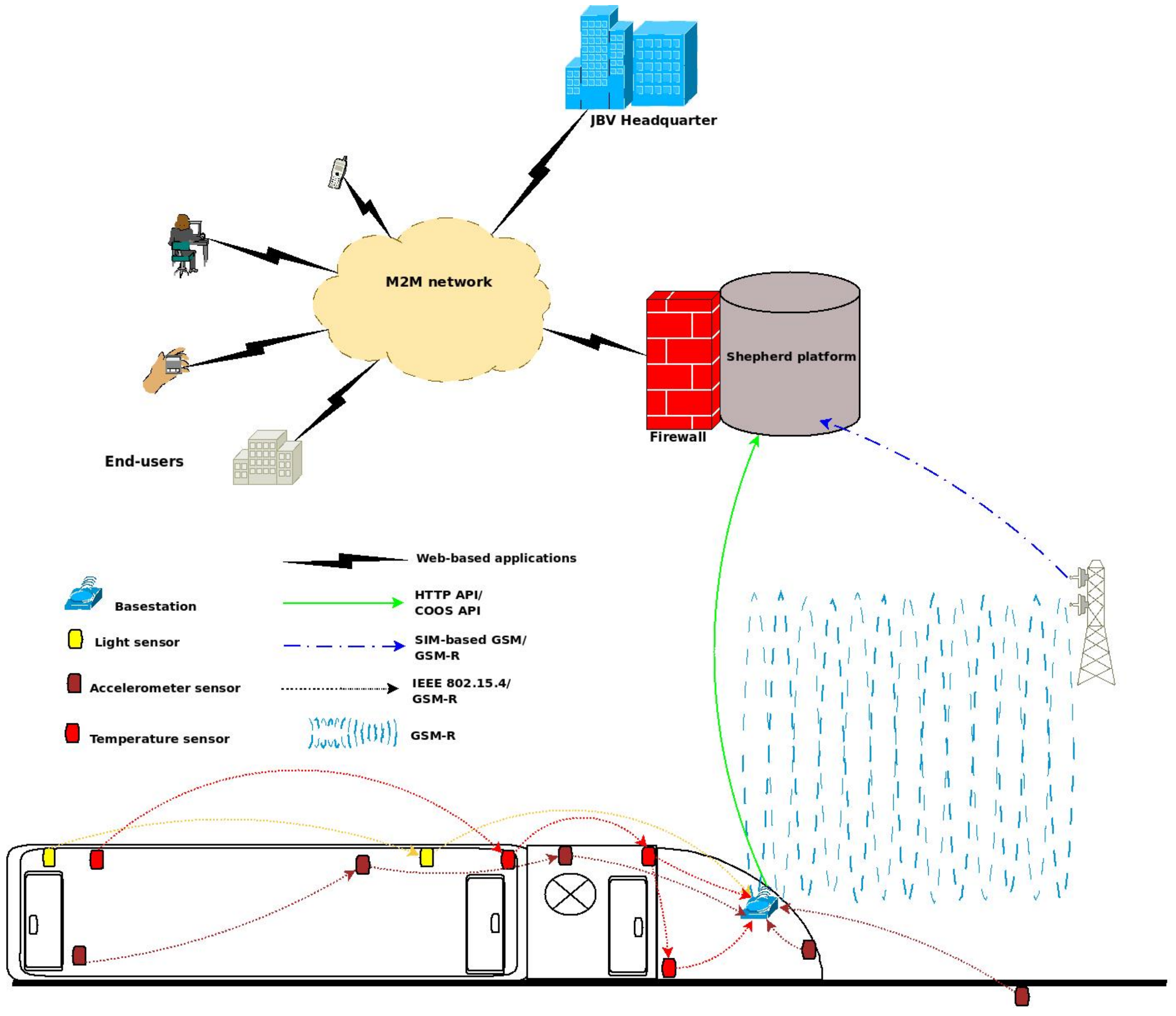


Figure 27: Integration of sensors for the JBV in the M2M network

Chapter 4

4 Implementation

This section will first describe briefly the technologies of Sunspot that are necessary to implement a prototype. Later, a description of the virtual and physical layout, so that the developed application can be tested and run on both virtual and real devices. In addition, going through the implementation of the applications.

4.1 Testbeds

4.1.1 Virtual Setting

The virtual setup is required for testing of an application that is under development. The advantage of the virtual setting is that it has virtual Sunspot, called *Emulator Sunspot* which is identical to the real Sunspot. The Emulator Sunspot has all the sensors that are necessary for this ease study. These sensors are adjustable. Refer to Appendix 8.4 for viewing the images. The virtual setup is very easy to handle and useful for testing by using the *Solarium* to run the *MIDlet* applications.

In order to use Emulator Sunspot, one must download a SPOTManager as mentioned in section 2.4.2. When installing the SpotManager, one is prompted to install the NetBeans IDE, JKD and SDK that are needed to develop Java applications.

Other benefits of SPOTManager is that, one can upgrade to the latest SDK or change the SDK version that one wants to use. Besides, it is possible to change the proxy settings if one has a proxy, enable the base station mode, and the like.

The Figure 28 below shows the libraries required to run applications on virtual setup. These packages must be added to the project's CLASSPATH before the application is built.

When one is satisfied with the development and the application is bugs-free in the virtual setting, it can then be deployed to the real Sunspot device.

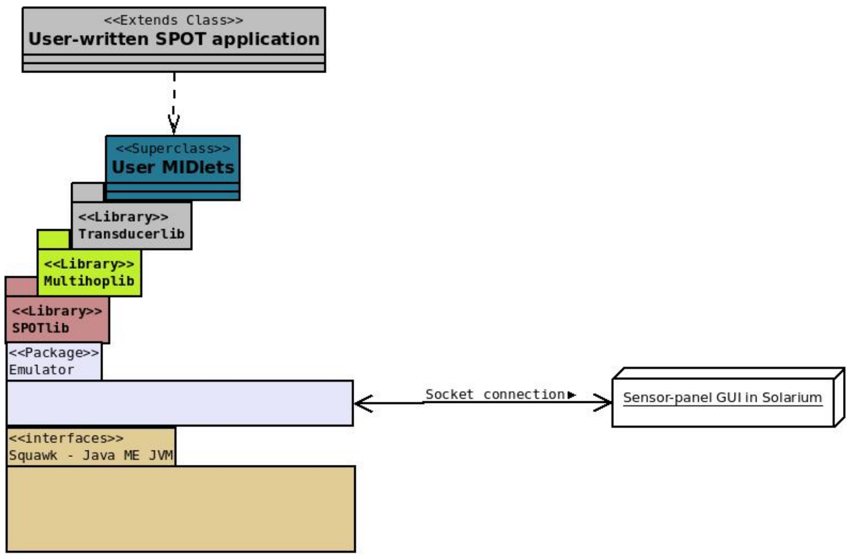


Figure 28: Software stack for Virtual SPOT. Reproduced from [37]

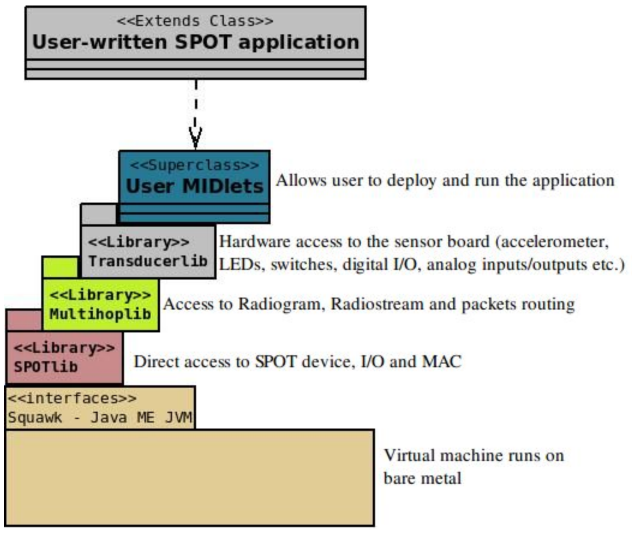
4.1.2 Physical Setting

Software stacks

The software stacks or the libraries for the wireless Sunspot devices are slightly different from the wireless base station. There are developed two applications that each have their own libraries:

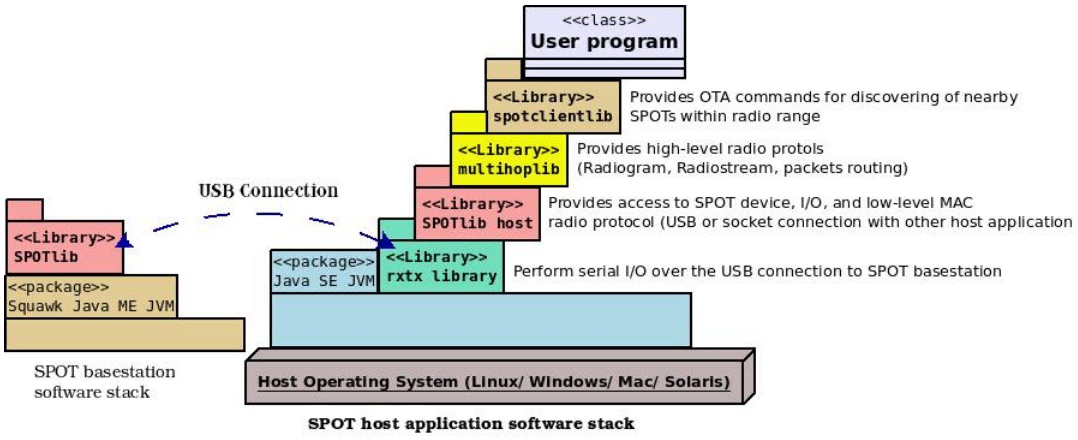
1. *Application for wireless Sunspot device* is developed with J2ME. To check the version of CLDC, see the *"MANIFEST.MF"* file under project's *resource* folder. It is used *MIDlet* application, and the libraries that are required here are defined in Figure 29.

2. *Application for wireless Sunspot base station* is developed with J2SE [38]. This application has one main method as an entry point for the program. The base station has no battery supply so it must be connected to a host computer. For the program to run with the wireless base station, it requires among other libraries for communication between the application and the USB connection to the base station. Other required libraries are shown in the Figure 30.



<<Extends Class>>
User-written SPOT application
<<Superclass>>
User MIDlets
Allows user to deploy and run the application
<<Library>>
Transducerlib
Hardware access to the sensor board (accelerometer, LEDs, switches, digital I/O, analog inputs/outputs etc.)
<<Library>>
Multihoplib
Access to Radiogram, Radiostream and packets routing
<<Library>>
SPOTlib
Direct access to SPOT device, I/O and MAC
<<interfaces>>
Squawk - Java ME JVM
Virtual machine runs on bare metal

Figure 29: Software architecture for wireless SPOT. Reproduced from [37].



<<class>>
User program
<<Library>>
spotclientlib
Provides OTA commands for discovering of nearby SPOTs within radio range
<<Library>>
multihoplib
Provides high-level radio protols (Radiogram, Radiostream, packets routing)
<<Library>>
SPOTlib host
Provides access to SPOT device, I/O, and low-level MAC radio protocol (USB or socket connection with other host application
<<package>>
Java SE JVM
<<Library>>
rxtx library
Perform serial I/O over the USB connection to SPOT basestation
USB Connection
<<Library>>
SPOTlib
<<package>>
Squawk Java ME JVM
SPOT basestation software stack
Host Operating System (Linux/ Windows/ Mac/ Solaris)
SPOT host application software stack

Figure 30: The base station connects to the host machine via USB and run the Host application. Reproduced from [37].

Setup

How to set up the devices physically, and establish the connection between them?
Figure 31 shows the physical connection between the base station to the host machine via an USB cable, and it communicate wireless to the Sunspot devices. Here, the Sunspot device located anywhere in the room, but within the range of 10 meters.

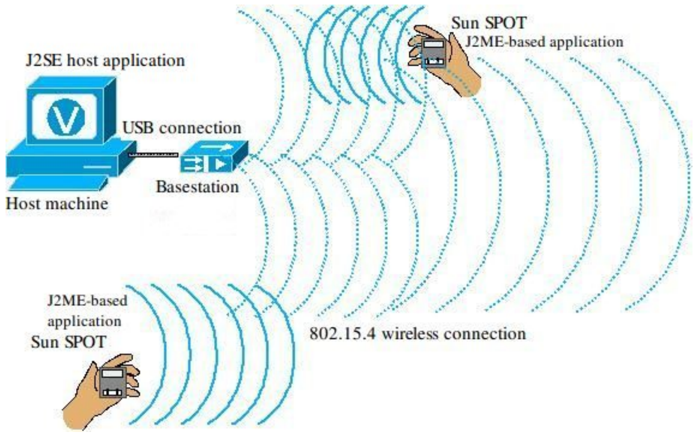


Figure 31: Setting up the SunSPOTs

The host computer is a Linux platform which is used to develop applications for the Sunspot devices. Application for the base station shall henceforth be called *Host application*, while application for the wireless Sunspot devices is called *SPOT application*.

Properties of the base station

The basestation can either be one of alternatives modes [37]:

(A) *"Dedicate mode"* is when the base station is being connected directly to the host machine via the USB, and the communication is controlled by the JVM, which is running on the host machine. Here, only *one* Host application can be accessed, and runs on that JVM at a time. This mode is conFigured to be a default mode for the basestation when connected to the host machine the first time.

(B) *"Shared mode"* has several segments, each segment is assigned with an address. An application is running on its own assigned address. This means that this mode allows for *multiple* applications running simultaneously on the same JVM.
The advantages of Shared mode

i. When using the Emulator for testing of the application, it requires that the base station is in *Shared mode*. That way, one can easily troubleshoot the problem in the application on the Emulator before deployment.

ii. Another advantage is that, when the base station is in *Shared mode*, one can build a network of Sunspots using the Solarium. Thus, Emulator Sunspots and real Sunspot devices can communicate with each other wirelessly.

iii. When the base station is in *Shared mode*, multiple host computers communicate with each other about their Sunspot devices. For example, host A has two Sunspots, and host B has 3 Sunspots. Host B can access A's Sunspot devices through A's base station. This is illustrated in Figure 32.

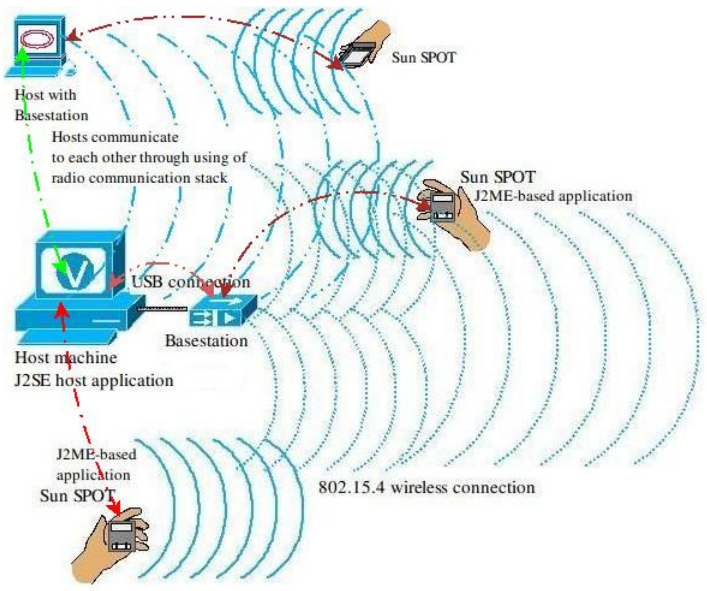


Figure 32: Communication between multiple host machines

4.2 Description of Prototype

This section takes up the main core of the programming code. Then, review the main elements of the code developed for the prototype, and includes the methods used here, and their properties. The program is divided into two main parts:

I) Part I. The first part is to develop an application used by Sunspot devices. In the application, the Sunspot device measures its own sensors, then temporarily, stores these values in variables. Then, establish wireless connection with the Host application via the base station, to deliver these values.

II) Part II. The second part is to develop the Host application that communicates with the Sunspot device, then receives Sunspot sensors's values. Next step, is to establish a connection with the Shepherd® platform and transmit these values for storage.

4.2.1 The Main Core of The Code

Modifiers

In the program code *keywords*, and *access modifiers* have been used. These are explained below:

(i) The *static final* keywords are used on the constants in the class. The keyword *final* ensures that the constant could not be changed once it has been initialized.

(ii) The *volatile* modifier is used when declaring local variables in the class. Modifier tells the virtual machine that, if one thread will access this variable, it must reconcile its own private copy of this variable with the master copy that is in memory.

(iii) *public, private* are both access keywords, that have been used in classes, class' variables and defined access of the method. When the *private* keyword is used to the class, variable or method, it can only have the access within that class where it has been defined. While, *public* access keyword applies globally, that means also outside the class.

(iv) The *synchronized* keyword is used in almost all the methods in *Sunspot application* and the *Host application.* The method that has defined this keyword indicates that only one thread can access this method at a time.

Multithreading

This thesis has used threads from object class in the package *java.lang.Thread.* The Thread class has the methods that provide and manage a thread to be created as instance object, then started, and can be paused at anytime. The most important and most used methods in the program are:

(i) *start()*

(ii) *run()*

(iii) *sleep()*

(iv) *wait()*

All actions and works have actually been carried out in *run ()* method, and therefore started first. When one initializes and defines a class to use Thread with either the keyword *extends* or *implements Runnable* interface, this thread will always start with its *run()* method. Like this thread:

```
HostDoBroadcasting hostbroadcast = new HostDoBroadcasting();
Thread broadcastThread = new Thread(hostbroadcast);
broadcastThread.start();
```

Once the *start()* method is called, the thread is considered to be *alive*, and starts the tasks inside the *run()* method.
Some methods in the program have also overwritten *run ()* method for instance:

```
public synchronized void hostIsreceivingPackets(final String spotId, final int spotPort){
new Thread(){
@Override
public void run(){
...
}
}.start();
}
```

Each class is initialized as an instance object. Each object is assigned a thread. This thread runs separately in its own heap. Besides this, there are many more threads as well, such as reading temperature value, setting accelerometer value, etc.

When it comes to scheduling of threads, it is important that they run without having to wait for each other, causing *deadlock*. The methods *sleep ()* and *wait ()* are used in the program to avoid this. The difference between these methods is that, when a thread goes to sleep, it does not commit all the resources along it. Whereas, a thread that has been called the *wait ()* method, will give up all its resources to the threads that are waiting for these resources.

To avoid having threads waiting on resources and on each other for eternity, the thread is assigned a specific time when it should go to sleep or have to give up and wait for the resources, for instance:

i. *sleep(5000)* tells the thread to go to sleep for 5 seconds.

ii. *wait(3*60*1000)* tells the thread to give up the resources and wait for 3 minutes.

After the allotted time is over, the thread wakes up and continues the tasks which has been interupted.

Exception Handling

When a thread is trying to establish a connection, it may be that this connection is already open, or it failed in some way to establish the connection. These failures must be caught with *IOException*, so that the codes are improved.

This also applies if a thread is in a deep sleep or wait for the resources longer than necessary, where the resources are already available. If it failed to wake up the thread, an *InterruptedException* will be caught.

Communication path

In order to transmit values from one spot to Host via a base station, it must establish wireless connections between them. The chart 33 below shows the main communication path of the programs which the author has developed in this thesis.

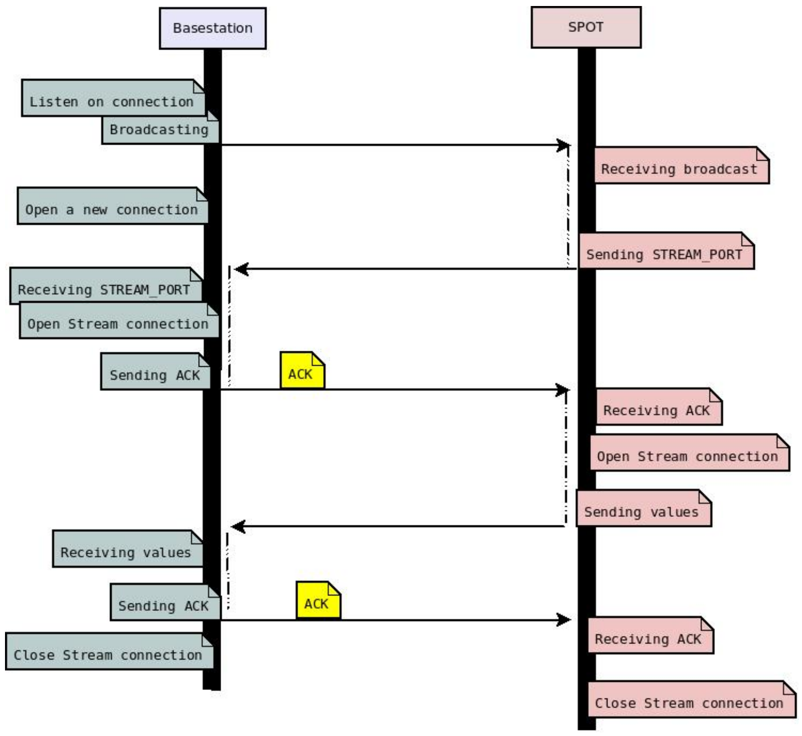


Figure 33: Establishment of connection and message exchange between the spot and base station.

Description of message

The message that has been used to exchange between the spot and the base station consists mainly of an address, a destination, a message body and an *EOF* number where 0 indicate the end of the message. These are illustrated in Figure 34.

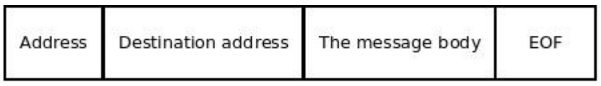


Figure 34: Message contents.

4.2.2 Part I - Developing of SPOT Application

This section starts by explaining the background of a *MIDlet* application and its lifecycle. *SPOT application* is a *MIDlet* that is suitable for devices with limited resources. Later, reviews the project structure and the main files contained in the SPOT application.

Lifecycle to a MIDlet

A SPOT application that is using a *MIDlet* when that Java class inherits from the superclass *MIDlet*, and defines for instance the class, as follow:

public class SunSPOTDatagram ***extends*** *MIDlet* {}

A Java class that inherits the *MIDlet* superclass, needs also to define an empty constructor, for example:

public SunSPOTDatagram (){}

When the constructor is called with the keyword ***new***, it initializes an instance object. The object is then used to invoke the methods.

Figure 35 shows the lifecycle of a MIDlet [30]. There are three main states. Each state is explained in detail below:

1) *startApp()* method will be invoked by the AMS so the application will be launched or resumed such as destroy the MIDlet. This method will be called first at bootstrap to initialize and present the application. This method is simular to the *main* method in Java SE or Java EE application.

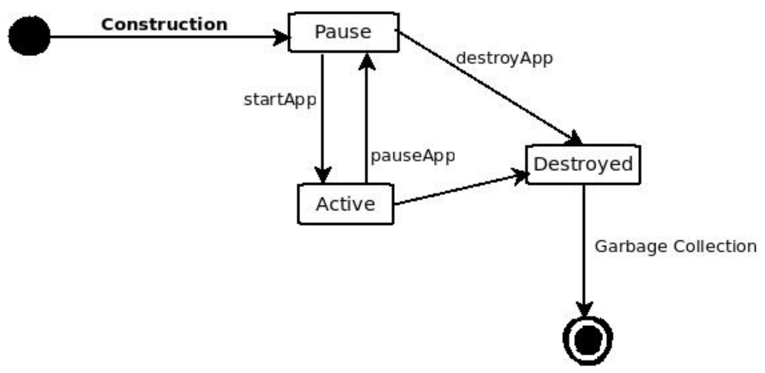


Figure 35: The MIDlet life cycle. Reproduced from [30].

2) *pauseApp()* method is invoked by the AMS if the system needs to perform an interruption of the application. This method releases all the resources that have been used. After this method has been called, the MIDlet remains and does nothing at this point.

3) *destroyApp()* method is used by AMS when the application must exit by the user-interrupt or by the system input. The native garbage collection is performed to end the MIDlet.

At the *Pause* state, the MIDlet can enter to the activating phase and the code is started or is continued. But, the MIDlet can be destroyed by the systems at any time at the *Active* phase.
While, in the *Active* state, which is a MIDlet's entry point. An user can interact with the system through the application that causes an event to occur. Here, at anytime, the AMS can force the MIDlet to enter back to the *Pause* state. This is the case for wireless mobile device when detecting an incomming call or lack of power, which results the application to go from *Active* state to the *Pause* state. However, the AMS can force the MIDlet to terminate at any time and lets the virtual machine's garbage collector take care of the destruction of a MIDlet.

Another way to let the application change the state by itself without forcing by the AMS, is to invoke the method *notifyPaused()*. This tells the AMS that the MIDlet *does not want* to be activated. Instead, it enters to the *Pause* state. By this way, the MIDlet can later return back to its *Active* state when invoking the *resumeRequest()* method. When calling this method, the AMS makes the application to move to *Activate* state, which will force the *startApp()* to start.

When it comes to end the MIDlet without AMS interaction, it is only needed to call the *notifyDestroy()* method. This will notify the AMS about the MIDlet is going to terminate. So, all the resources are being released, as well as all threads are exited and all procedures for the clean up is performed.

These are important characteristics of the MIDlet when handling callbacks and asynchronous of the application.

Next generation of MIDlet

According to an article from the Oracle Web site[48], the next generation of the MIDlet is IMlet which has an additional module for the MIDlet version 2.0. This additional module called IMP, which is MIDP profile of Sunspot application. This can be found in the file *MANIFEST.MF*, that defines the following:

MicroEdition-Profile: IMP-1.0
MicroEdition-Configuration: CLDC-1.1

IMP is used for application viewing on a simple user interface and targets the mobile devices and embedded systems with constraints on power. IMP is also application for M2M Devices.

The SPOT application is using CLDC with version 1.1 that has the GCF interfaces in the package *javax.microedition.io*, which supports for instance, the floating-point operations and establishes the connection port.

Storing of MIDlet

Each Sunspot device has the ability to store the values temporarily in the flash memory [37]. When using the *Record Management Store (RMS)* for writing directly to the flash memory as persistent data, and reading from memory.
The package *javax.microedition.rms.RecordStore* has the method *addRecord (byte[] data, int offset, int numBytes)* and is used to allocate an area in memory for adding the values. The value is retrieved from memory by using the method *getRecord (int recordId)*, where ID is a specific sector in memory.

SPOT application

SPOT application is a MIDlet application with the project structure as follows:

i. *build.properties* file located in the root of the project. This file contains the script that Apache Ant uses to build the program. If no script is specified here, Ant will look for other files *.sunspot.properties* and *build.properties*. Here, one can define the USB port to be used for this application, the IEEE address to

[48]`http://developers.sun.com/mobility/imp/impstart/`. Published by Qusay H. Mahmoud on Mar, 2006, last accessed 15. Nov, 2010

the spot to be communicate with, or its own address so this spot can be used *remotely.*

```
spotport=/dev/ttyACM0
-DremoteId=0014.4F01.0000.493B
```

ii. *MANIFEST.MF* file is located in folder *resources*. In this file, one can specify the name of the package, profile of the MIDlet, the main application, etc. It can also specify multiple MIDlets here if the program uses multiple MIDlets.

```
MIDlet-Name: Sensors
MIDlet-Version: 1.0.0
MIDlet-Vendor: Sun Microsystems Inc
MIDlet-1: SunSPOTDatagram, , org.sunspotworld.sensors.SunSPOTDatagram
MicroEdition-Profile: IMP-1.0
MicroEdition-Configuration: CLDC-1.1
```

iii. *src* folder contains the Java classes for the application. SPOT application has two Java classes. These are:

   (a) *SunSPOTDatagram.java* is the MIDlet application. In this class, there performs the reading from the sensors and store values in the class's local variables, as well as retrieving the values from these variables. The main method that starts up as an entry point of the SPOT application is in this class with the name *startApp()*. In this method, it includes a listener which starts and detects the USB connection. In addition, it also can be set a number of properties such as spots's addresses in a *whitelist*, defined an object's thread and started this thread. As well as call method *notifyDestroyed()* for the destruction of MIDlet when the works have been done.

   (b) *SpotExchangeDatagram.java* class has the main tasks that are to listen to broadcasts from a base station, establishe the connection, and send the values to it. The methods that perform these tasks are the *run()*, *replyRadiogram(String destination)*, and *sendValuesToHost()*.

iv. *suite* folder contains an executing *.jar* file for the application. This folder contains one jar-file name *Sensors_1.0.0.jar*.

Description of the main methods in SPOT application
Hereafter, describes the main features in details:

(A) *Accessing the sensor boards.* The EDemoBoard class represents the so-called *"out-of-box"* sensors of the SPOT. To get a single instance of the board, it is only needed to call *EDemoBoard.getInstance()*. For instance, to access the *Temperature sensor*, simply call *EDemoBoard.getInstance().getADCTemperature()*. Other sensor boards that have been used are:

- ◇ Light sensor: *getLightSensors()*
- ◇ Accelerometer sensor: *getAccelerometer()*

(B) *Reading, temporary storing and retrieving of values.* The methods for reading, setting and retrieving the values from the sensor boards always start with *read*, *set* and *get*, for instance:

(i) *public synchronized void readTemperature()*{}

(ii) *public synchronized void setTempval(String tempval)*{}

(iii) *public synchronized String getTempval()*{}

(C) *Listen to broadcasting.* The method that is used to listen to the broadcasting is *public synchronized void run()*. This method will always start first when the MIDlet is up and run. This method uses a DatagramConnection to establish the connection to the base station on a server port for receiving the Datagram broadcasting. The spot will listen to and receive the broadcast from the base station every 60 seconds.

(D) *Send reply with a stream port.* After receiving the broadcast, spot responds back with a request to the base station to open a new connection to receive values in a stream port = 60. The spot will wait until it receives an *ACK* from the base station about received the request. These actions are been taken care of in the method *public synchronized void replyRadiogram(String destination)*.

(E) *Send the values.* Then, the spot establishes a new direct RadiogramConnection to the base station on port number 60, and sends the values to the Host via the base station.
Once the SPOT has etablished a direct connection to the base station, it sends the message directly to the destination. At this time, the message does not contain the destination address, but mainly of the sensor values that the spot has measured. Here, the author assumes that the message will not be lost once the direct connection to the base station has been etablished.

4.2.3 Part II - Developing of Host Application and Communication with Shepherd® Platform

For the base station to communicate wirelessly with other spots, the following must be defined for the base station:

(a) Define the base station mode to be in the *Shared mode*.

(b) Turn on the property for starting *over-the-air (OTA)* Command server.

(c) Enable the detection ability to automatically detect the USB port each time the Host application is running.

These properties can either be done using the SPOTManager, via the Ant commands, or in the *main()* method of Host application.

Host application

The main actions in a Host application is to perform broadcast every 15 seconds on the server port with the message *"I'm the base"*. At the same time, the base station listens to the answer for establishing of connection from the spot that hears this message.
When communication between the base station and the spot is established, the base station opens a new RadiogramConnection for receiving the sensors's values, which are sent with a Datagram.

Note, for security's sake, the Host application gets a new address each time it starts. It is therefore not used the base station's address, when the spot's values are sent to its destination.

The main classes in the host application are:

(A) *HostApp.java* has the *main ()* methods, which is application's entry point. In *main ()* method, there initialize an object to perform broadcasting with its own thread. While, another thread will be responsible for establishing of connections with spot on the stream port, and receives the values from spot on this port.

(B) *HostDoBroadcasting.java* establishes DatagramConnection and sends out the broadcasts Datagram.

(C) *ExchangeDatagram.java* has the following methods:

   i. *run ()* method that will always start first, and listening on the server port for receiving request from spot to open a new connection on a stream port.

   ii. *hostIsreceivingPackets()*. When the base station has received a stream port, it establishes a direct RadiogramConnection to the spot. After the values are received, there initializes a Sunspot object, and add this object in an ArrayList of type *SunSPOT*.

To define the Sunspot device to use as an object, so it is possible to initialize the device as an object instance, the following classes are defined:

(a) *ISensors.java* is an interface class that declares and the setter and getter methods that describe a spot, such as spot identification, spot address, timestamp, light value, temperature value, and accelerometer value.

(b) *SunSPOT.java* is a class that inherits from interface ISensors. This class implements all the methods it inherits from superclass.

Establish a connection with the Shepherd® platform

The purpose of using Shepherd is that it provides a service platform that allows integration of different sensor types. That means, there is the possibility of any end-users, in the future, would benefit from the various services the platform offers, also includes Sunspot devices. The second step here, is to establish a bridge between Sunspot application and the Shepherd® platform.

This thesis has established a relationship with the Shepherd over HTTPS connection using the HTTP protocol.

To be able to access the Shepherd® platform for sending the values, it needs to apply the application to Telenor Objects AS, to be assigned with an Application ID and an Object ID. These IDs are the identifiers that identify and allow a specific device application to access the platform, as well as retrieve the information that has been sent to Shepherd from that device application.

Besides the Host application, there are two classes that are defined in the program to establish this connection. These are:

A) *ISendMessageToShepherd.java* is an interface that declares only one method, and some ID constants such as hostname, https port number, object id, application id.

B) *SendMessage.java* is a subclass that inherits all constants and implement the method:

public void doPOST()

The method performs the following tasks:

(i) Initializing an URL connection with the HTTP protocol

(ii) Open a *HttpsURLConnection* to Shepherd

(iii) Set the request property to use the POST method

(iv) Set other properties such as header, object id, application id, hostname, etc.

(v) Encoding the message string to be sent with the url

(vi) Using an *OutputStream* and sends out the message in bytes.

Then, wait until it receives a response code to confirm or deny from Shepherd. The code must be *200*, which is an *OK* message and means that Shepherd has successfully received the message. Otherwise, code:

i. *404* means *Not Found* or *Unknown*

ii. *413* means that the message size is larger than the allowed size of 8KB.

Here, the request uses the POST method, which allows the message body to be long.

Data storage using PostgreSQL

Imagine that, the sensors continuously measure the values whether the values are abnormal or not. While, the values needs to be sent to the Shepherd is determined by time, or if it exceeds the limit. In the meantime, it is perhaps important to have an internal storage location, that stores absolutely all the measured values. So that, one can at any time, by using an application to send the requests to this storage for data. Therefore, in this thesis PostgreSQL is used as data storage for the Sunspot's values.

PostgreSQL version 8.4.5 has been used. In the next paragraph PostgreSQL will be described briefly.

PostgreSQL[49] is a Database Management System (DBMS) with a relational database model, that supports the standard SQL query language. The reasons for choosing PostgreSQL as a storage for the sensor values are, firstly, PostgreSQL is an open source software, and provides stability and reliable with good performance characteristics.

The second reason is that PostgreSQL has an architecture that can be applied into the Client/Server environment. It runs on many platforms, such as Unix, Linux, Mac and Windows, and it can be installed on a single server. Applications can access the database through the database process to obtain the values. The client programs cannot access the data directly, but must use the specific message protocol to be able to connect PostgreSQL. This separation is an advantage, because it

[49]`http://www.postgresql.org/docs/`, last accessed 11. Des, 2010

allows an application to be distributed as well as it can be a platform independent. When installing the software on the client, the Java application must use a Java Database Connectivity (JDBC) to to access and use PostgreSQL database.

Thirdly, database allows for multiuser access, and automatically ensures that the updating has no conflict when multiple users are updating the same table.

The data storage is actually a flat data file, that consists of text which can be read by an application. Any record is stored as table or tuple. A tuple is an ordered group of components, or attributes with the defined type such as *char, real* and so on. And each attribute in a record is *ATOMIC*. This means that an attribute in a database table can have relationship with an attribute in another database table. If one update of an attribute, it will also be updated the related attribute. This is a *All or Nothing* principle which is the database property.

To etablish connection to a specific database by using the *DriverManager*, and provide the url format:

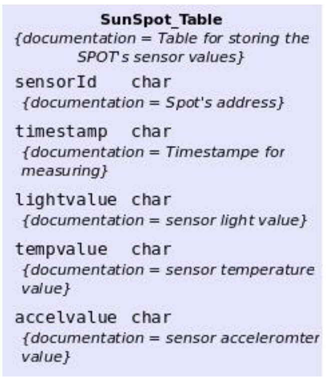
Connection databaseConn = DriverManager.getConnection("jdbc:postgresql://hostname:port/", "DATABASE_NAME", "DATABASE_USER", "PASSWORD");

The JDBC driver version *postgresql-8.4-702.jdbc3.jar* is used here, and it also needs to be specified in the application classpath, as well as the following configurations:

a) The JDBC uses an IP socket connection to access the database. Add the line *tcpip_socket = 1* in the file *postgresql.conf* to turn on the IP socket.

b) Allow the IP connection to the database in the file *pg_hba.conf* with *host all all 127.0.0.1 255.0.0.0 trust*. It can also provide the IP address where the database is located.

The Java class name *SunSPOTDatabase.java* contains this connection setting to database name "*sunspot*". Note, this database must exist for the application to be able to etablish the connection.
This class also performs several actions, such as creating a table name *SunSpot_Table* and insert the values into this table. To check that the values are inserted correctly, the method *showData()* is used to display each column and row with values that are stored in the table *SunSpot_Table*. Figure 36 shows the properties used in database table *SunSpot_Table*.



SunSpot_Table
{documentation = Table for storing the SPOT's sensor values}

```
sensorId    char
 {documentation = Spot's address}
timestamp   char
 {documentation = Timestampe for measuring}
lightvalue  char
 {documentation = sensor light value}
tempvalue   char
 {documentation = sensor temperature value}
accelvalue  char
 {documentation = sensor acceleromter value}
```

Figure 36: Database table to SunSPOT and its attributes.

4.3 Achievements and Prototype Demonstration

This Chapter described the tasks required to implement a prototype using the Sunspot device. The Figure 33 shows a success in the establishment of an intended two-way wireless communication between the Sunspot device with its base station. The technology used here, is the IEEE 802.15.4 protocol. The following explains the tasks that have been implemented as shown in Figure 37.

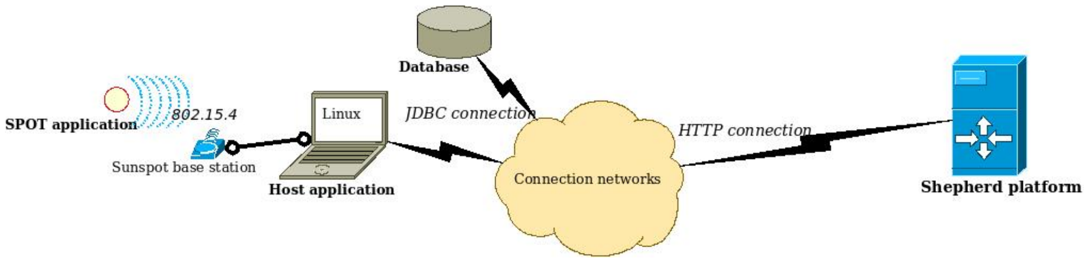


Figure 37: Task achievements.

An Host application has been developed, that performs broadcasts every fifteen seconds. While, a spot application that is developed for the Sunspot device, will detect the broadcasts every thirty seconds. But, it does not transmit the values to the base station after one minute has passed since the last envoy. The message sent to Shepherd looks similar like this:

1. 2010/11/30 17:46:58
Tue Nov 30 08:46:54 PST 2010, SunSPOT Id: 0014.4F01.0000.493B
The current temperature is: 31.25 °C
The accelerometer is: X=-0.034908700322223416, Y=-0.040279269602577876, Z=0.9693877551020409. Tilt: -1.0
Light is: Bright = 154

When the values arrive, there will also be added to an *ArrayList* continuously. At the same time, the Host application sends out a request to Shepherd on receiving the values. The connection is opened until the application has received confirmation from the Shepherd of receipt. However, the values to be sent to Shepherd, only happens in every five minutes. Below shows the program's output when Shepherd received the message:

```
Prepare to send values to Shepherd
HttpsURLConnection is established.
Setting properties...
Ready to send...
Sent to Shepherd!
Response: OK = 200
Closed all connections to Shepherd.
```

The SPOT application is also designed to detect spot's battery level prior it using the wireless communication. If spot battery is either lower than -32 or greater than 32, then the MIDlet will be destroyed, and the application is terminated as well.

The whole program is also shown in the appendix 8. The applications have been tested both on Emulator spot and real Sunspot device.

The programs have been carefully constructed and implemented in the way that the programs should not or do not have bottleneck, or Hot-Spot that can affect the programs' performance. The applications are utilized by multi-threading, each application is running in a thread-safe environment.

In addition, the author has also established a connection with the database to store all measured values internally. This is with regard to further development of this program to be more flexible when one want to retrive the values at a specific time.

Chapter 5

5 Demonstration at ITEA2 Co-Summit

5.1 Artemis pSHIELD Project

SHIELD is an abbreviation for *Systems arHItecturE for multi-Layer Dependeable solutions*. This is a framework, and the description of this framework is defined in document [40].
The SHIELD framework is aimed to separate and clearly define the different layers so it may be adapted to the M2M infrastructure of the Embedded Systems. The *multi-layer* consists of node layer, network layer, middleware layer and application layer.
This allows any innovative sub-projects, freely to adapt the SHIELD's concepts to their projects according to the technology solutions that are currently existing today. The SHIELD is also proposed for the ARTEMIS Calls.
The SHIELD project has a duration of 12 months, and started on June 1, 2010.

The framework consists of the four main levels:

I. *Node layer* can be any intelligent hardware component with a *built-in* encryption mechanism.

II. *Network layer* is a layer for data transfering. This layer must adapt the security for trusted and dependent network interactions across domain. The network can also has the properties of *≪self≫-automations* such as *self-management*, *self-configuration*, *self-recovery*, *self-maintenance* and so on.

III. *Middleware layer* must also implement the security, be efficient for inter-operation. In addition, acts as a resource management for heterogeneity. This means that this layer should provide support for the interaction of different entities with different applications.

IV. *Overlay layer* is aimed at application development. The applications can be targeted the industrial sectors, or other public services.

The SHIELD concept is to address the ≪built-in≫ properties as mentioned earlier. These are based on *≪Security, Privacy and Dependability≫*, and is illustrated in Figure 38.

This shows the inter-relationship components in an overall view. The infrastructure needs to be based on a full *buil-in* mechanism of the security when applying

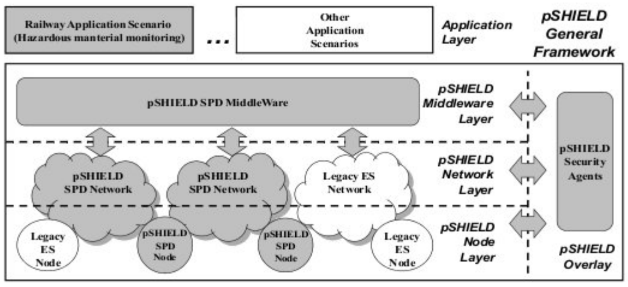


Figure 38: *Multi-layer-based built-in* concept. Copy from [40]

to the Embedded Systems. In the sense that facilitates and realizes the diversity of system integrations.

In addition, the concepts that are to be applied, should also be transparent for the end-users, while the SPD cycles are repeateable.

In the realization of SHIELD, the SHIELD project introduces a new *pSHIELD* as a pilot project with the members shown in Figure 39. The project's goal is the

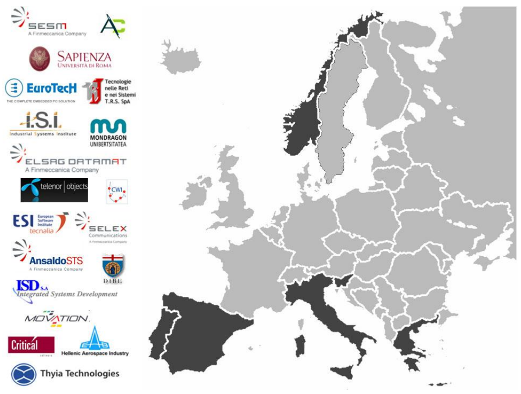


Figure 39: SHIELD European Consortium. Copy from [40]

adaption of SHIELD's concept and infrastructure to any ambitious and crucial domains such as the railway, or the heathcare as well.
This thesis is part of the *pSHIELD*[50] realization. The project has an use-case that is to apply to the JBV based on the SHIELD concept. The work was presented in an exhibition in Ghent.

[50]`http://wiki.unik.no/index.php/CWI/PSHIELD`, last accessed 18. Nov, 2010

5.2 Exhibition and Demonstration

ITEA 2 is a program where the aim is to gather different partners including industry companies, research institutions and universitities around Europe to come together with their inventions of technologies for the embedded systems. The ITEA 2 has more than 100 projects in different areas ranging from IT, telecommunications to the nanotechnologies, and many more.

This year, the ITEA 2 and ARTEMIS (Advanced Research & Technology for EMbedded Intelligence and Systems) co-organised and arranged the Co-Summit 2010[51] in October at Ghent in Belgium. The theme for the conference was *"Mobile and cloud power enabling massive scalability and opportunities for growth"*.

I had the honor to participate this exhibition. I represented the CWI[52] research center in where I also wrote this thesis.
As part of pSHIELD project, CWI research center has a project which aims to implement and realize the pSHIELD concepts. That is, the integration of different sensors to a service platform for heterogeneity. This thesis is part of the project, which aims to integrate the Sunspot sensors to this platform. We have the JBV as an use-case, and in the exhibition I presented this.

On the top layer, as shown in Figure 40[53], applications can be developed for the railway with the possibility of system interoperability in the future. The realization

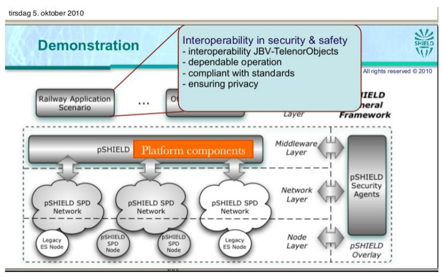


Figure 40: Laboratory prototype based on pSHIELD concept. Copy from [41]

of the *pSHIELD concepts* with adaptation to JBV as the use-case. Here at the *Node layer*, there can be any kind of sensors for instance, SunPOT devices, GPS sensor or IQRF in this case. These sensors can be integrated in any embedded device, or the sensors can be fixed, mounted inside the wagons of the train. Figure 41 also shows this possibility of integration.

[51]`http://www.artemisia-association.org/cosummit_2010_home`, last accessed 30. Nov, 2010

[52]`http://wiki.unik.no/index.php/CWI/AboutUs`, last accessed 18. Nov, 2010

[53]`http://wiki.unik.no/index.php/CWI/PSHIELD-Oct`, last accessed 18. Nov, 2010

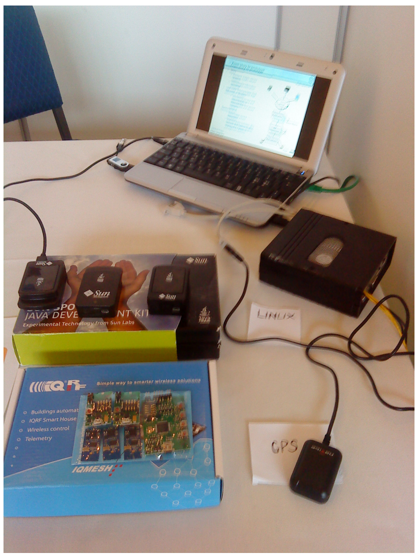


Figure 41: Sensors integration (private photo).

At the *pSHIELD SPD Network layer*, the GSM/ GSM-R networks are the opportunities. Again, the goal is the embedded systems with embedded sensors. They must therefore support these networks in a safe manner. In this layer, JBV must make the decision on whether they will build their own network, or make use of telecommunications operators's network. Anyway, it must be built on a reliable network.

At the *middleware layer*, Telenor Objects helps with their invention of a heterogeneous service platform as a software entity. The mechanism behind this, will guarantee all the requirements and shall satisfy the *SHIELD concepts*.

Figure 42: SESM and CWI partners at the exhibition (private photo).

Chapter 6

6 Performance, Discussion and Evaluation

6.1 Background

Generally, when running a Java program, the program is compiled so it can execute. This means that the Java source code is being translated into Java bytecode, which is a native language for the Java Virtual Machine (JVM). The Java interpreter acts like a buffer between the host machine and the JVM and allows the JVM to run the bytecode version on the host computer.

In this case, after compiling, the program generates *.class* files of the Java classes under the folder structure named *build*.

Java programs are not running in native mode (directly on the computer), but must go through the Java interpreter. This makes the Java programs run 100 times, or more, slower than other programs written in for instance C/C++. But, the advantages with Java programs, are that firstly, they can run on any machine that supports the JVM. As been said "*write once, run anywhere*". Secondly, in fact, every instruction in the code needs to be validated, because the JVM operates at a higher level. Thus, ensuring that the code is more secure and does not harm the computer.

Performance is, therefore, very important. For example, a program that has been programmed correctly, can take more memory than is available on the computer it is running on, and the time it spends in using memory is higher than expected. Then, there is a reason to worry about the performance of that program.

There are two methods for enhancing the performance of Java programs. Either one can use the *just-in-time* (JIT) compilers that translates Java bytecode to the native computer language, or one can install a Java microprocessor in the computer in order to run the Java bytecodes in native mode via a microprocessor.

This section covers two benchmarking approaches. These are, to determine a reasonable performance analysis of the program and to implement a performance measurement of the program codes. In addition, it discusses the maintenance of the network and the system, which is related to the JBV's use-case. And finally, it makes suggestions for future work and what remains for the realization of the system, that can be used by the JBV.

6.2 Performance Analysis

The main task here is to measure the time it takes to execute a certain task, a procedure or a piece of code when the program is running.

In this thesis the author aims at analyzing the performance based on the measurements of time (*time analysis*) for a code to execute. This is a quantitative measurement. The purpose is to determine the so-called *Hot Spot* of the program. Then, evaluate and discuss whether it is necessary to tune, or improve the code so that the program can run faster. This is to achieve optimization and efficiency of the program.

Data materials are based on measured time comsumptions of the code, in the determination of a quantitative measurement for the observation and evaluation.

The method *currentTimeMillis()* from the *System* was used to measure the time of a program code during its execution. The test was carried out in the Host application, but not on the SPOT application. This was because the methods to be tested were similar to those used in the Host application. The following operations were measured:

a) broadcasting with a *DatagramConnection*

b) reading from a stream

c) writing to a stream

d) receiving values from Sunspot on *Stream port*

e) adding an object to an ArrayList

f) sending the values using *HttpsURLConnection* and receiving a response code from Shepherd .

The Host application is running on Linux computer with two processors (one core cpu) Intel(R) Atom(TM), CPU N270 and 1.60GHz. The Java version is:

> *java version "1.6.0_07"*
> *Java (TM) SE Runtime Environment (build 1.6.0_07-b06)*
> *Java HotSpot (TM) Client VM (build 10.0-b23, mixed mode, sharing)*

While, the Sunspot device is equipped with a 32-bit ARM920T core processor with 180MHz, 512K RAM and 4M Flash. The SPOT application is running on this device.
The communication between the Sunspot base station and the Sunspot device is wireless with protocol IEEE 802.15.4. The Internet connection with the Shepherd has the bandwidth of 10/10 Mbit/s.

During the test, the Sunspot device is placed at different locations in the room to detect and observe the time in conjunction with the distance between the device and the base station.

One issue that needs to be mentioned here, is the time measurement. This can vary greatly from one run to another. For example, the machine that is currently used to measure is under heavy load, which can have an impact on the measurement.

Another purpose of this test is to observe whether the traffic between the application and Shepherd is reliable. It is expected that the data sent to the Shepherd has little or no impact on the network traffic.

The tests were carried out during four days of total 20 runs to observe the trend as well as the daily variation. Sunspot device sent the values to the Host application every minute. Then, the values were sent to Shepherd just after the reception.
During testing, the Sunspot device was tested while being shaking and in continuous motion, to detect the stability of the wireless communication between the Sunspot device and the base station.

6.3 Performance Measurements Process

The program was run 20 times (n) for collecting a set of samples/ repetitive of occurrences (m). Each measurement was carried out in half an hour. Below, showed the list in Table 3 of the number of occurrences (m), that occured at each execution of one measurement: The bar chart 43 shows the summary of one

Table 3: Numbers of repetitive in the code executions.

Code execution	**# of occurrences (m)**
Base station broadcasts on Server port	119 - 122
Base station receives and reads message from Spot	1
Base station sends a reply message	1
Base station receives the values	30
Add object to ArrayList	30
Messages exchange between Shepherd and Host application	30

test.

The measurements were carried out as follows. At the beginning of the code, the clock was started, and at the end of a particular task, the clock was stopped. The operating time was then being recorded to a spreadsheet for further analysis.

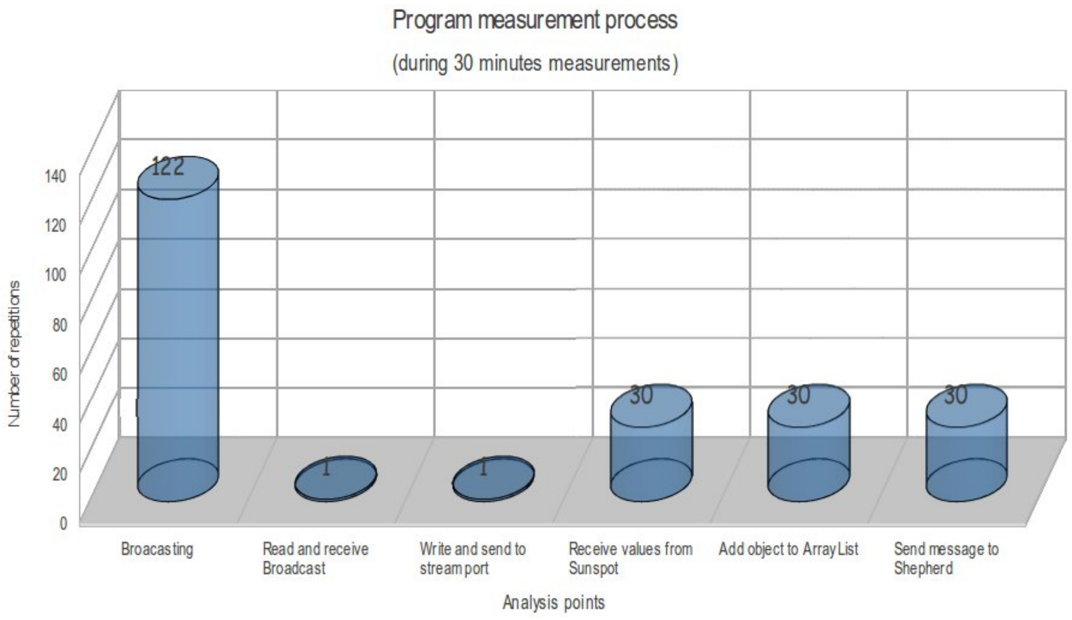


Figure 43: Measurements process during 30 minutes.

6.4 Evaluation and Discussion

The program started with the base station that sent out the broadcasts, so the Sunspot device could hear and receive these messages. In the broadcast, the code had the following tasks, which the time had been measured on:

(i) set date timestamp in a variable

(ii) set a string of message body in a variable

(iii) use method *reset()* to reset the datagram

(iv) write to datagram

(v) send datagram on a DatagramConnection

These tasks are repeated inside a *while()* loop that was always defined to be true. Time spent on these tasks is shown in the Figure 44.

In 20 measurements (n = *20*), there were approximately 119 to 122 repeating broadcasts. The reasons for this variation, was either some delay in the exchange of messages over the Internet during this particular half-hour experiment, or it was because no Sunspot device heard the broadcast when the program was started.

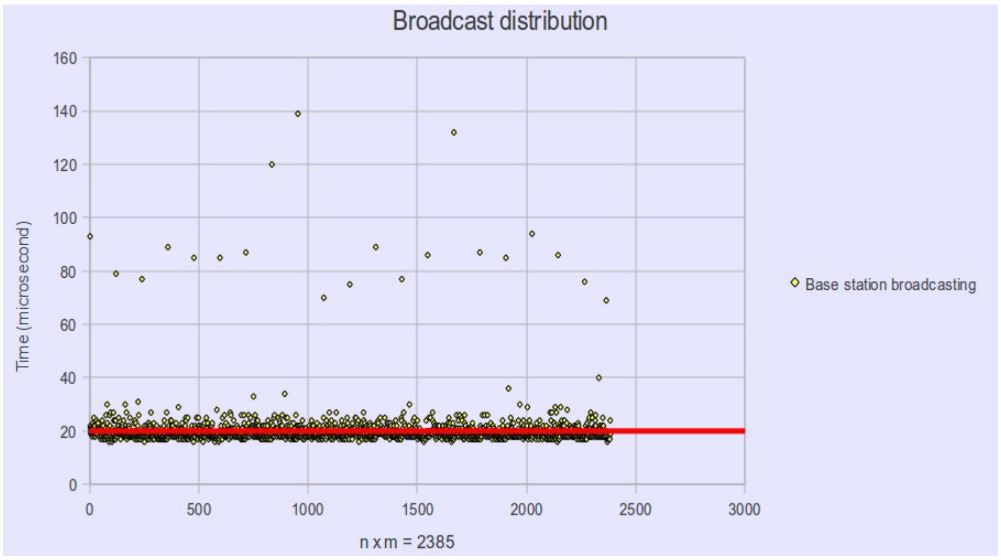


Figure 44: Broadcasting measurement.

The Figure 44 also shows that about 99 percent of all the broadcasts $(n \times m)$ had the broadcasting time about 20 microseconds. While only one percent had slightly higher time. The reason was that the connection had to initialize and establish a stable connection on the first broadcasting. Here, the time consumption varied between 70 to 140 microseconds.
After the connection had been established, the time consumption of this connection was leveled down to the average level and remained quite stable. The red line shown in the Figure 44, represents an average of the total repetitive broadcasts.

The average $(\overline{x})$ was calculated on the following forms:

$$\overline{x_i} = \frac{[n_i \times m_i]}{n_i},$$

where $0 < i \leq n$. And the total *Average* $(\overline{X_A})$ is the sum of the averages in a data matrix of $[n_i \times m_i]$ for each code execution (see table 4).

$$\overline{X_A} = \sum_{i=1}^{n_i \times m_i} \overline{x_i} = \frac{[n_i \times m_i]}{n_i}$$

The Figure 45 shows time-consumption, which the Host application had used to receive (*receive()*) and read (*readUTF()/ readInt()*) a Datagram sent by the Sunspot device on a RadiogramConnection. This included the time that it took

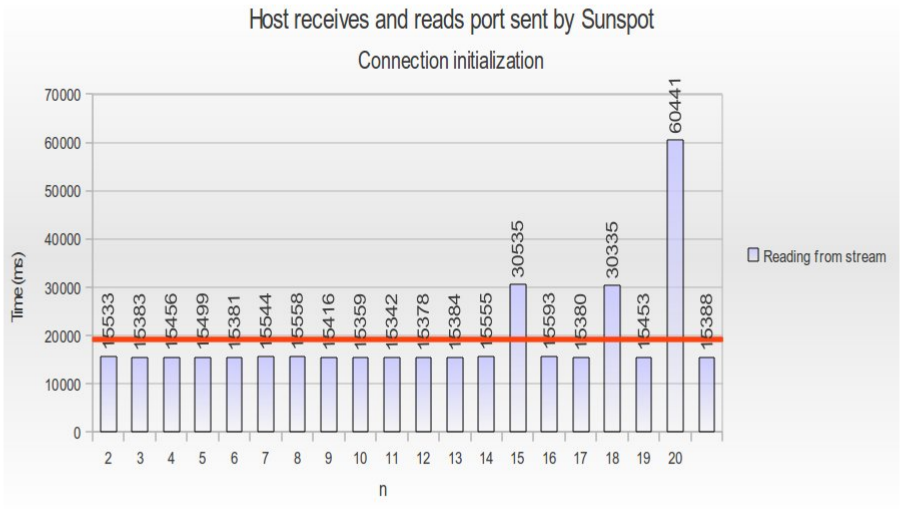


Figure 45: Receive stream port from Sunspot.

to add the value into a variable either of type *String* or of type *int*.
The measurements had the time around 15.3 to 15.6 milliseconds. The Figure 45 also shows that three measurements that had the time slightly higher than others. The reason for these peaks was the system (host computer) at these points maybe was being overloaded. Thus, affecting the average to approximately 19.2 milliseconds.

However, the variation in time that the Host application had spent to send a response (see Figure 46) back to the Sunspot device was minimal. The chart 46 shows the time taken on 20 measurements in which the application had used the methods *writeUTF()/ writeInt()* to write to a Datagram, then used the *send()* method to send that datagram.

The wireless communication between the base station and the Sunspot device indicated that, immediately after a connection had been established, it often took longer to either receiving or sending a message. This was compared with one connection (Figure 44) that sent (*send()*) out the broadcasts. While, another connection (Figure 46 and Figure 47) started with receiving (*receive()* then *send()*) the message from/ to Spot.

When a connection was maintained and stable, and if there existed a reuse datagram (as situation occurred in Figure 46 and Figure 47) that is ready to either send or receive a message. In this case, there was no significant difference in time consumption, and the time consumption was stabilized and were roughly equal.
Moreover, one can see that the size of exchange messages have no influence on the time consumption for the methods *receive()* and *send()*. Because, in this case, it

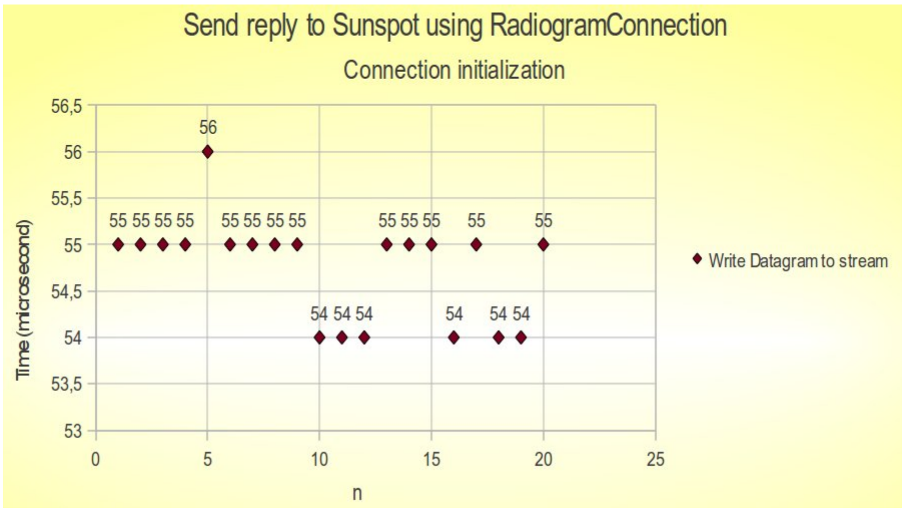


Figure 46: Send reply to Sunspot.

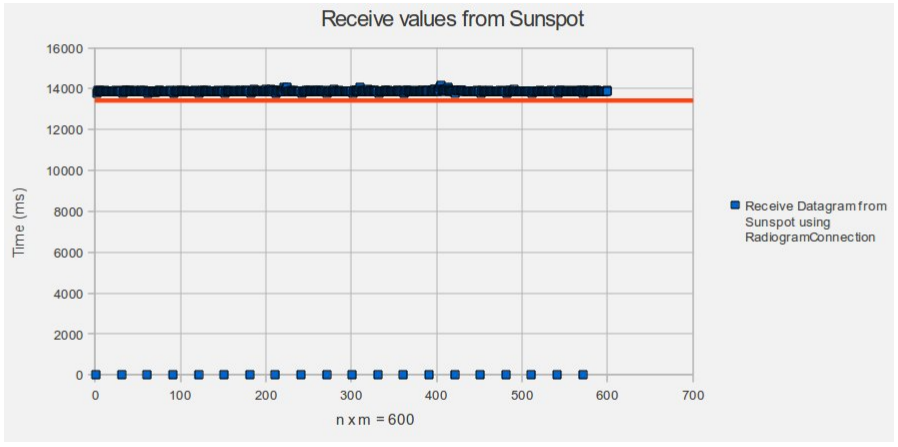


Figure 47: Host application receives the values from Sunspot.

seems that a message could have greater data content that had no impact on the time consumption, as it was for the state of receiving values.

Figure 47 also shows that the time consumption was no longer than 14.0 milliseconds. The tasks that were done during this time were receiving, reading and continuously storing to local variables.
The blue dots occurred in the Figure 47 at zero level was because the connection had been waiting for the reception where there was nothing to receive. Then, the *while()* loop forced the operation to run for receiving of new incoming message. If there was data to receive from the connection, the *read()* methods would be executed for reading.

The method *add()* that was used to add the object of type *Sunspot* to an *ArrayList* was also measured. Here, the time was taken on the tasks when comparing of object type and continuously add the item into the list. Figure 48 shows the

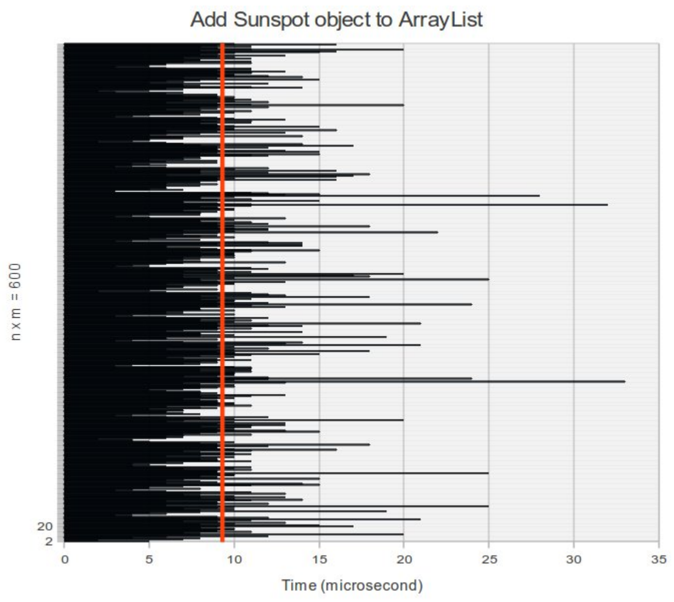


Figure 48: Add an object to list.

repetitions that had no more cost of time with the average of 9.3 microseconds. This should be a reasonable time.

Communications with Shepherd

The communication between the Host application and the Shepherd® platform was the Internet connection. Therefore, the bandwidth for *Uplink* and *Downlink* impacted the measurements. But, it also depended on the size of the message, which mean that the method chosen for the compression of message decided the size of a message. Which again, further determined the response time of delivering on the *Uplink*.

In this test, the time was measured from just before the opening of a *HttpsURLConnection* with a specified *URL*, and stopped right after all messages were sent out. The following was done during this time:

(i) Open *URL* connection

(ii) Flag the *URL* connection ready for sending

(iii) Initiate a *URL* connection ready to accept incoming input

(iv) Set caching to false

(v) Set the method to use the POST

(vi) Set several properties of the method

(vii) Prepare the output stream for the connection

(viii) Encoding the string to be sent out

(ix) Write to the output stream of encoded bytes

(x) Send out the output stream

Then, the time was started again for the following actions:

(a) Initializing the input stream from the same *URL* connection

(b) Read response code from this connection

(c) Compare the response code, store this value in a variable and print out to the Terminal.

The time was then stopped again. The total time spent on the method *flush()* for sending and the methods *getInputStream()* and *getResponseCode()* for receiving is shown in Figure 49.

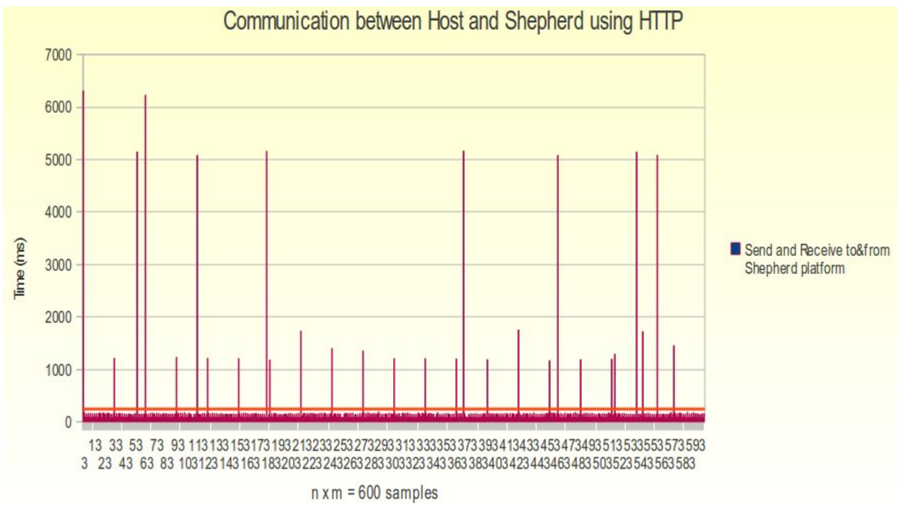


Figure 49: Communications with Shepherd.

Also here, the values were high at initialization of the connection, but decreased rapidly and ranged from 80 to 180 milliseconds. It was observed that the time for sending out took longer than the time for receiving. In addition, Figure 49 also shows some peaks. This occurred between m deliveries of one measurement. But, happended as much as 30% of the total 20 measurements.

Summary

Table 4 shows a summary of all measurements. The averages of each execution

Table 4: Measurements developing process.

Codes execution (A)	**# of surveys** (m)	**# of measurements** (n)	**Total Average** ($\overline{X_A}$)
Base station broadcasting	119 - 122	20	20,02μs.
Receive Datagram message from Sunspot	1	20	244,70ms.
Send reply using RadiogramConnection	1	20	54,75μs.
Receive values from Sunspot	30	20	13421,93ms.
Add object to ArrayList	30	20	9,31μs.
Send and Receive message to&from Shepherd	30	20	244,70ms.

are also calculated.

Figure 50 shows the average time for each code execution, and expansion of the graph for the lower values shown in Figure 51.

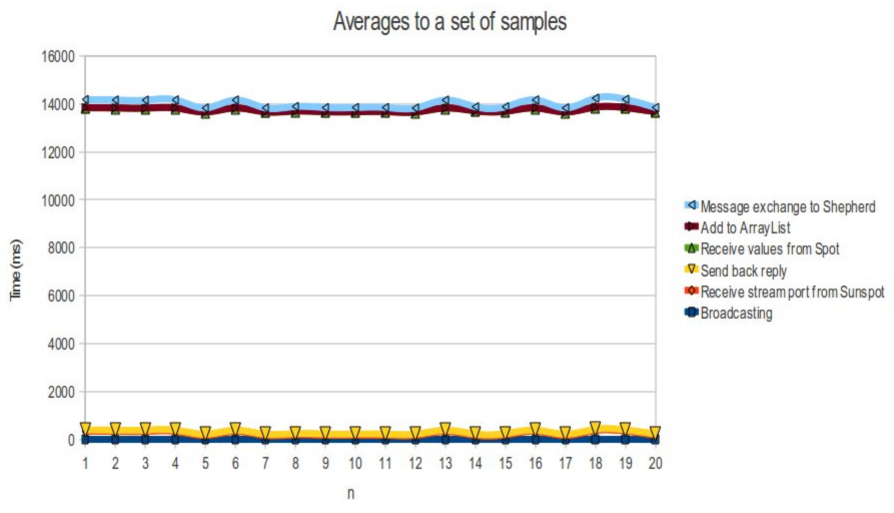


Figure 50: The averages of all measurements #1.

Figure 51 also show the linear regression lines for each execution code. For communication with the Shepherd, it appeared to decrease in time if one could have carried out several measurements ($n > 20$). The same would be applied for the state when the base station received values from Sunspot.

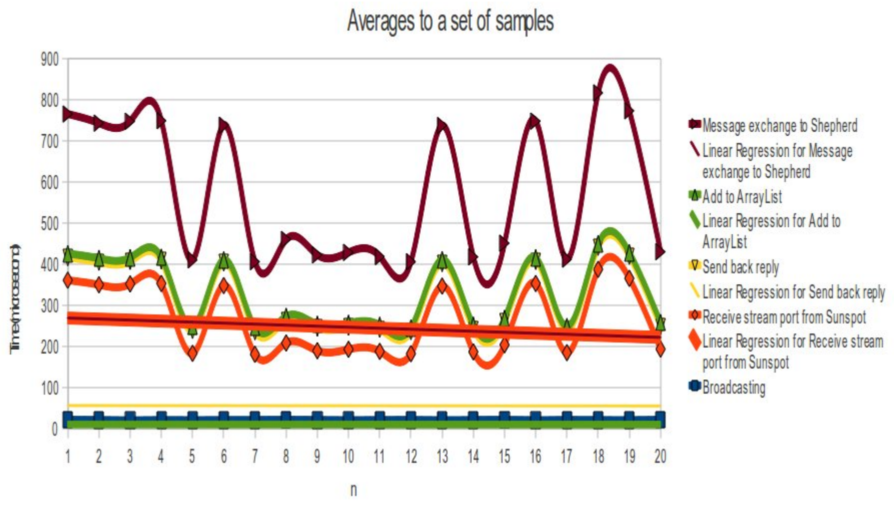


Figure 51: The averages of all measurement #2.

These should be the basic lines to be applied, when performing a performance analysis for optimization in the improvement of time-consumption.
As regards the improvement in connection with the Shepherd, a suggestion for optimalization here was for instance, by adding more nodes in the nodes infrastructure that had been set for Shepherd. A better load balancing would then, result in a significant improvement of response time.

The test also showed that shaking and movement of the device did not affect the wireless communication between the Sunspot device and the base station. Nor the plaster wall between the Sunspot devices affected communication between them. Apart from that, the distance between them would have an impact on the communication link. This meant that communication between them was broken.

6.5 Maintenance

Sunspot devices are very easy to handle and maintain. There is no need for battery replacement, moreover, the batteries may have longer life depending on how often one wants the device to monitor, measure and deliver value. The program in this thesis is designed in the way that, if there is low battery, the wireless communication will also cease.
One should therefore, have multiple devices that can perform the same tasks, so that when the device' battery is re-charged, it should not lead to deficiencies of the system in general.

Besides, it is possible to add other types of sensors such as GPS or RFID in the Sunspot device as an extension. Thus, it will be much easier to maintain the application that is designed for Sunspot devices.

When it comes to network maintenance for Norwegian National Rail Administration (JBV), it is dependent on whether the National Rail Administration chooses to use a *private* or *public* networks. This has been discussed in Chapter 2 and Chapter 3. But again, this should also be considered with regard to the choice of equipment to be used, as well as software for these devices.

However, to support M2M infrastructure in the network, and that information is not confidential, but may still need higher security level. The JBV should have an assessment, if it is less demanding and less costly for them to allow a telecommunication to take over the role of system maintenance. Where, there is also opportunity for software maintenance. The JBV should carry out a risk assessment to decide whether it is less demanding and less costly for them to allow a telecommunication provider to take care of network maintenance, including maintenance of software. This is partly dependent on how sensitive the JBV judges the information in the M2M network to be.

6.6 Future Works

The prototype developed by this thesis has been tested on Sunspot base station with one Sunspot device. The SPOT program has not been tested on how to let two Sunspot devices communicate with the base station simultaneously. Code for this purpose has been developed, but has not yet been tested. Further work is required in order to allow communication involving more than one Sunspot device.

In addition, there is also need for further work on establishing an algorithm for the application, in order to enhance the communications between the Sunspot devices in the delivery of the message to the base station. This may solve the some problems mentioned in Chapter 3.

It should also conducted a further examination of the sensors that are accessible and compatible with Sunspot devices. So that these can be integrated into the device. Besides, it also requires the changes of the software to take advantage of the additional sensors.

This thesis has introduced a M2M nettwork infrastructure with built-in security mechanism based on the standard ETSI TS 102.690. It has not implemented the security mechanisms for the prototype developed in this thesis. However, it has made use of the Secure Socket Layer (SSL) on HTTP protocol to establish a

HTTPS connection with the Shepherd® platform. It should therefore implement a method to encrypt the message, as well as a method for compression of the message, which will be forwarded to the Shepherd.

The prototype that has been developed in this thesis, can be used for further work on implementation of COOS instance. This is to achieve, among other things, reliable and secure communications between the device and Shepherd, and possibly better performance.

Finally, a web-based application that can retrieve data from Shepherd and show this in any browser should be developed as well. Moreover, it should also be possible to retrieve data that can be displayed in a mobile device with M2M nettwork infrastructure.

Chapter 7

7 Conclusion

This thesis has carried out two main objectives. The first is a deeper theoretical study of networking standards and infrastructure, relating them to the needs of the National Rail Administration. This may improve the systems and enhance the co-ordination toward a common system throughout Europe.

The second objective of the thesis is to develop a prototype that can be used in an M2M infrastructure. This M2M infrastructure makes use of existing networks. Here it is implemented in the railway domain, but the technology can be used in other areas as well, for example healthcare, telecommunications etc.

This thesis also presents the problems that JBV meets today, where there is a need for a system that meets and satisfies the goals of National Rail Administration, such as safety on board, business interests. As well as providing the services that increase the interest for the use of public transport in the population.

But first, it must establish a GSM-R network using equipment that is suitable for this network. This provides data security, not to mention that stolen equipment cannot be used without the GSM-R network coverage. If it is used within the network, the equipment will be tracked by the system.

In order to detect and trace an object, there is a need for sensors that are integrated into the object, such a useful sensor is the GPS sensor. But, a need for other sensors can also be taken into consideration.
In this thesis, sensors for measuring heat and movement have been used to illustrate this. There is a need for further development of applications for these sensors into a platform of heterogenous services.

8 Appendixes

There also included a CD in this thesis. You will find the software that runs on the Sunspot devices in the CD. There are also readme files that should be read before starting the program.

(I) Refer to Appendix 8.1 for the checklist before running any SPOT- and Host applications.

(II) To run an Java application with SPOT Emulator, see the description in Appendix 8.2.

(III) Appendix 8.3 described how to run the applications on real Sunspot devices.

(IV) Review the pictures that had been taken during the runs of the program in this thesis.

8.1 Appendix A. Checklist

(I) When running the SPOTManager, make sure that the host machine has the Internet connection.

(II) Aslo make sure that the host machine has the same SDK version as the applications. If not, a message of SPOT's SDK incompatible appeared as shows in Figure 52.

(III) If needed, upgrade the SPOT either via SPOTManager or via the command "*ant upgrade*".

(IV) Making changes in the *"userhome/.sunspot.properties"* file

- ◇ To change the default mode setting for basestation from "dedicate mode" to operate on the "Shared mode". Put this line *basestation.shared=true* in the file.
- ◇ Activate automatic port detecting: *spotselector.basestation.lastport=/dev/ttyACM0* (for Linux).
- ◇
  - *sunspot.home=*
  - *sunspot.lib=${sunspot.home}/lib*
  - *spot.library.name=transducerlib*
  - *spot.library.addin.jars=${sunspot.lib}/multihop_common.jar ${path.separator}${sunspot.lib}/transducer_device.jar*
  - *${path.separator}${sunspot.lib}/spotlib_device.jar${path.separator} ${sunspot.lib}/spotlib_common.jar${path.separator}*

- *${sunspot.lib}/squawk_device.jar*

(V) The USB can also be determined in the "build.properties" file:

- SPOT project: "*spotport=/dev/ttyACM0*" (Linux)
  "*-DremoteId=0014.4F01.0000.493B*"
- Host project: "*spotport=/dev/ttyACM0*", (Linux)
  "*main.class=org.sunspotworld.host.Host*"
  "*main.args=*
  user.classpath=/home/user/netbeans-6.9.1/platform/modules/ext/swing-layout-1.0.4.jar:
  /home/user/netbeans-6.9.1/java/modules/ext/AbsoluteLayout.jar"

(VI) If more than one MIDlet, define these in the "manifest.mf":
MIDlet-2: midletname2, , org.sunspotworld.sensors.midletname2

(VII) Turn on the "OTA".

8.2 Appendix B. Run Java application in Emulator

(i) In order to detect Emulator Spot, provide the Emulator Spot' IEEE MAC address in the *whitelist*:

- *Spot.getInstance().setProperty("radio.filterlist","true");*
- *Spot.getInstance().setProperty("radio.whitelist", "7F00.0101.0000.1001, 7F00.0101.0000.1002, 7F00.0101.0000.1003");*

(ii) Choose *"refresh"* to identify the base station' IEEE MAC address.

(iii) Start the base station. Activate the Shared mode if needed.

(iv) Clean and Compile the SPOT application and the Host applicatios. Open two Terminals:

a. Terminal 1: navigate the folder where the Host application is located. Type *"ant host-compile"* or *"ant -Dsportport=/dev/ttyACM0 host-run"*.

b. Termnial 2: navigate the folder where the SPOT application is located. Type *"ant solarium"*. This starts a new Solarium window for creating virtual Spot.

(v) From the menu, choose *"New Virtual SPOT"*.

(vi) Right click on virtual Spot, and choose *"Deploy MIDlet bundle..."*. The files "build.xml" and "*Sensors_1.0.0*.jar" (in folder "suit") must be deployed.

(vii) In Terminal for Host applicaiton, type *"ant host-run"*. Right click on the Emulator Spot, choose *"Run MIDlet"*.

Figure 53 shows the Emulator Spot is running the SPOT application.

8.3 Appendix C. Run Java applications on SunSPOTs

Run SPOT application on real Spot:

(a) Provide the Spot' IEEE MAC address in *whitelist* in the SPOT application:

- *Spot.getInstance().setProperty("radio.filterlist","true");*
- *Spot.getInstance().setProperty("radio.whitelist","493B, 52C3");*

(b) Clean and Build the application.

(c) Deploy to Sunspot device via USB cable.

(d) Run application. Unplug the device and stop the application. The real Sunspot device can also be viewed in the Solarium as shows in Figure 54. It can also see the wireless connection between devices as shows in Figure 56.

Run Host application:

(i) Open a Terminal

(ii) Connect the base station to the host machine via USB cabel.

(iii) Type "*ant host-run*".

Figure 55 shows it is possible to run the SPOT application on real device and on Emulator.

8.4 Appendix D. Pictures of Emulator SunSPOT devices

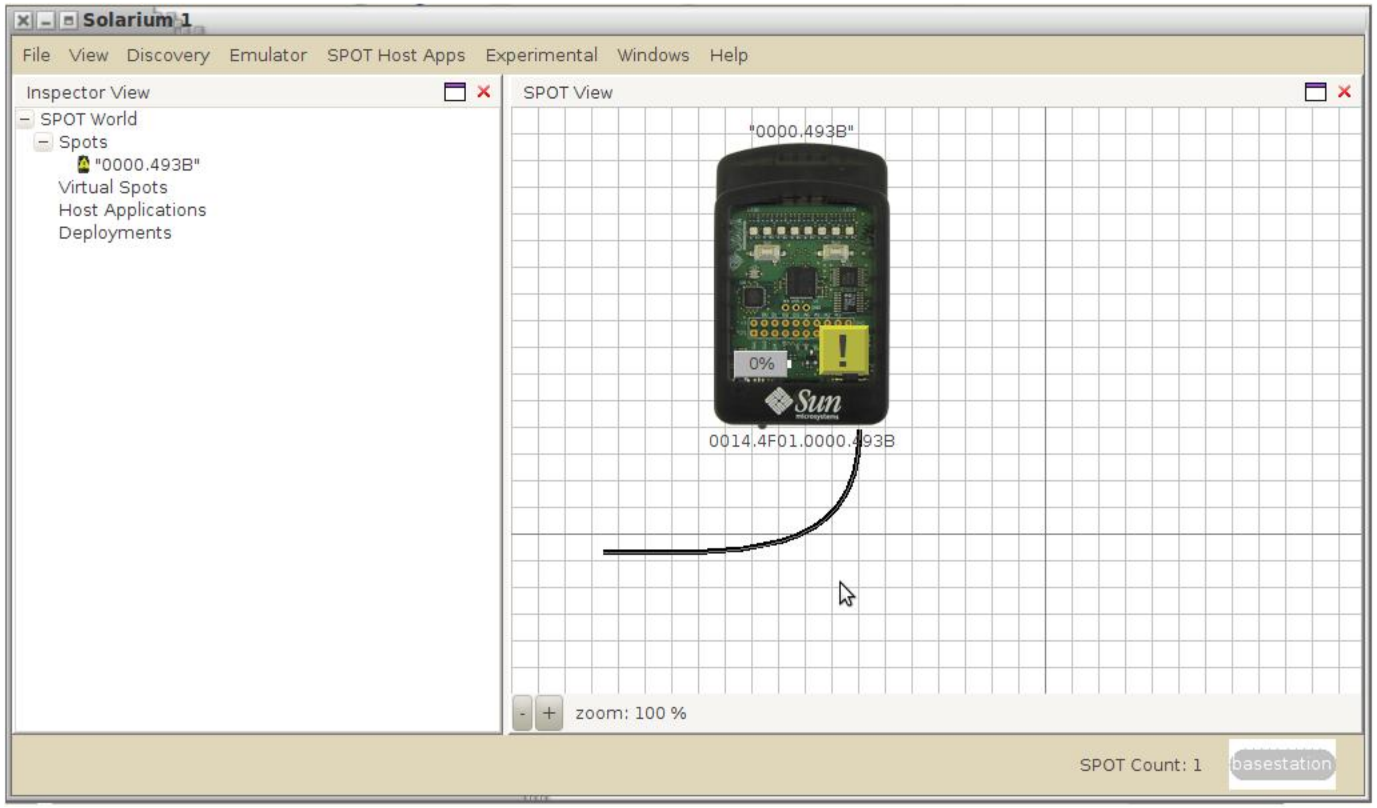
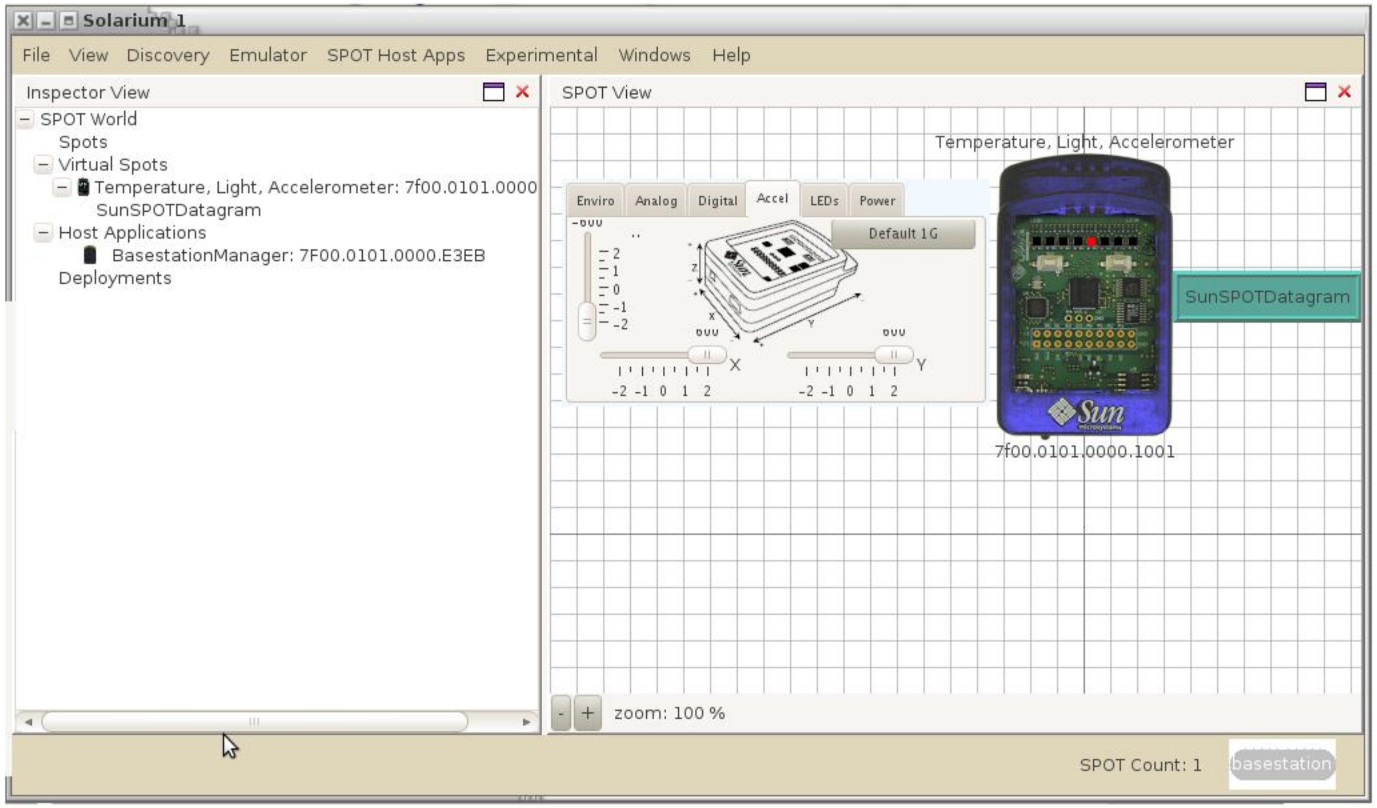


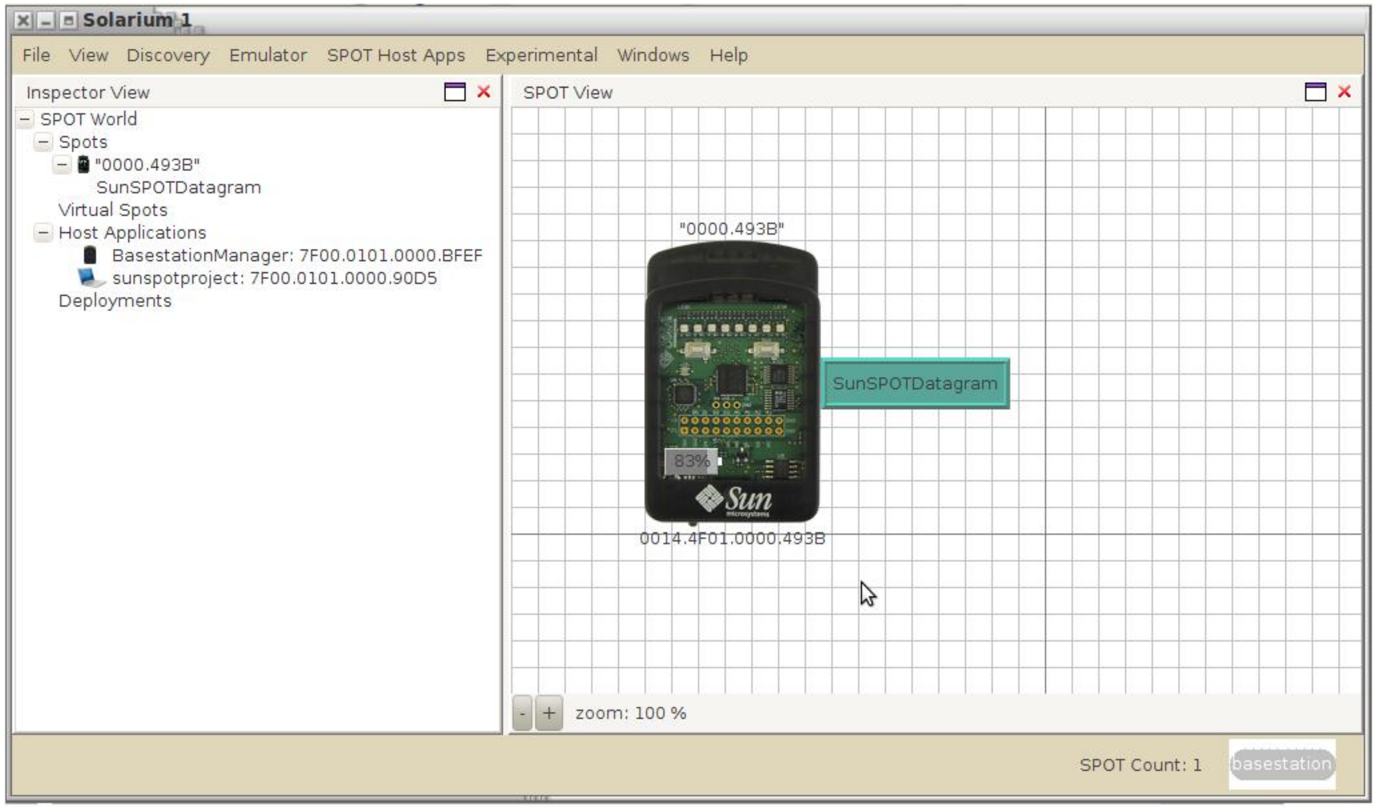
Figure 52: Warning message when incompatible SDK.


Solarium 1
File View Discovery Emulator SPOT Host Apps Experimental Windows Help
Inspector View
SPOT View
Enviro Analog Digital Accel LEDs Power
Default 1G
zoom: 100 %
SPOT Count: 1
basestation

SPOT World
Spots
Virtual Spots
Temperature, Light, Accelerometer: 7f00.0101.0000
SunSPOTDatagram
Host Applications
BasestationManager: 7F00.0101.0000.E3EB
Deployments

Temperature, Light, Accelerometer
SunSPOTDatagram
7f00.0101.0000.1001

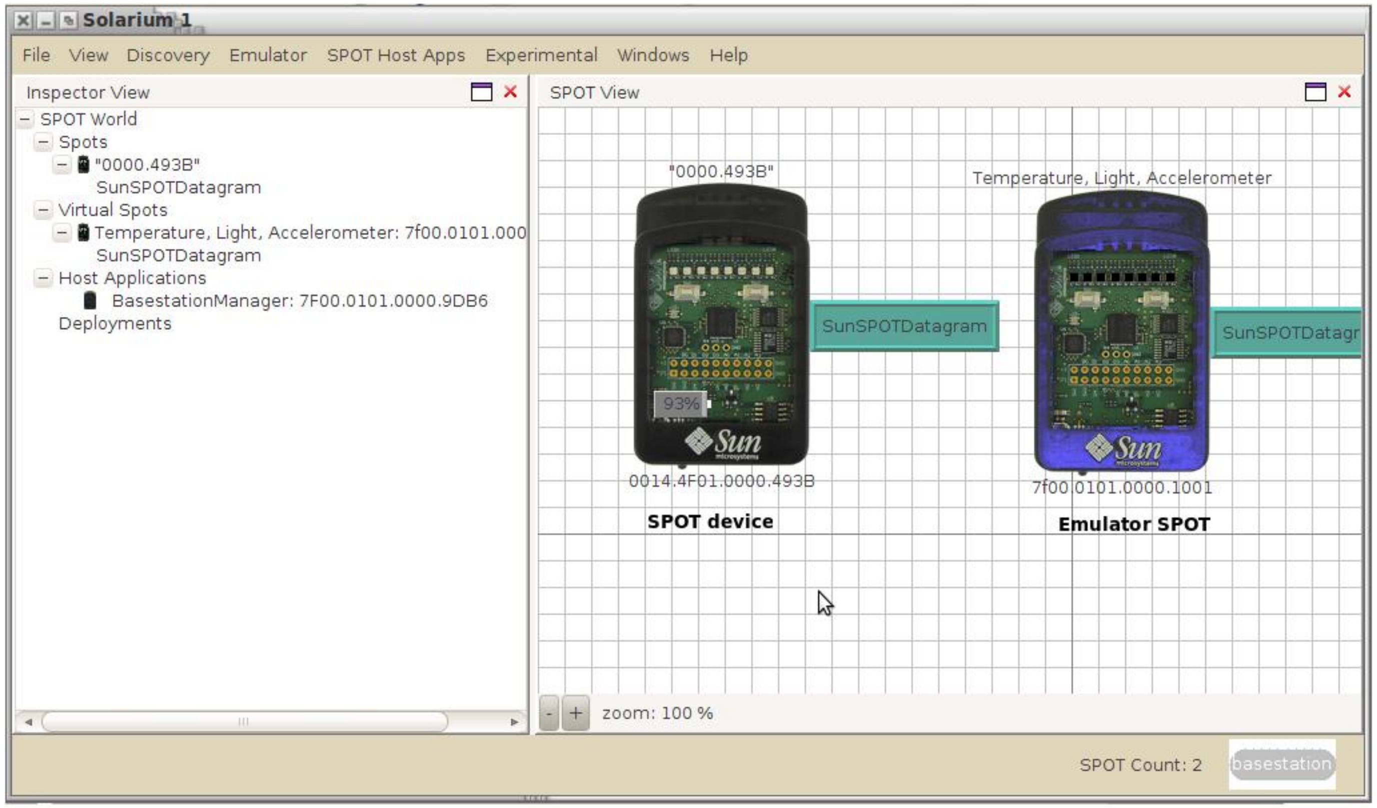
Figure 53: Test of SPOT application on Emulator SPOT.



SPOT World
Spots
"0000.493B"
SunSPOTDatagram
Virtual Spots
Host Applications
BasestationManager: 7F00.0101.0000.BFEF
sunspotproject: 7F00.0101.0000.90D5
Deployments

SPOT Count: 1

Figure 54: Using SPOTManager to discover real Sunspot device.



- SPOT World
  - Spots
    - "0000.493B"
      - SunSPOTDatagram
  - Virtual Spots
    - Temperature, Light, Accelerometer: 7f00.0101.000
      - SunSPOTDatagram
  - Host Applications
    - BasestationManager: 7F00.0101.0000.9DB6
  - Deployments

Figure 55: SPOTManager discovered both real Sunspot and Emulator spot.

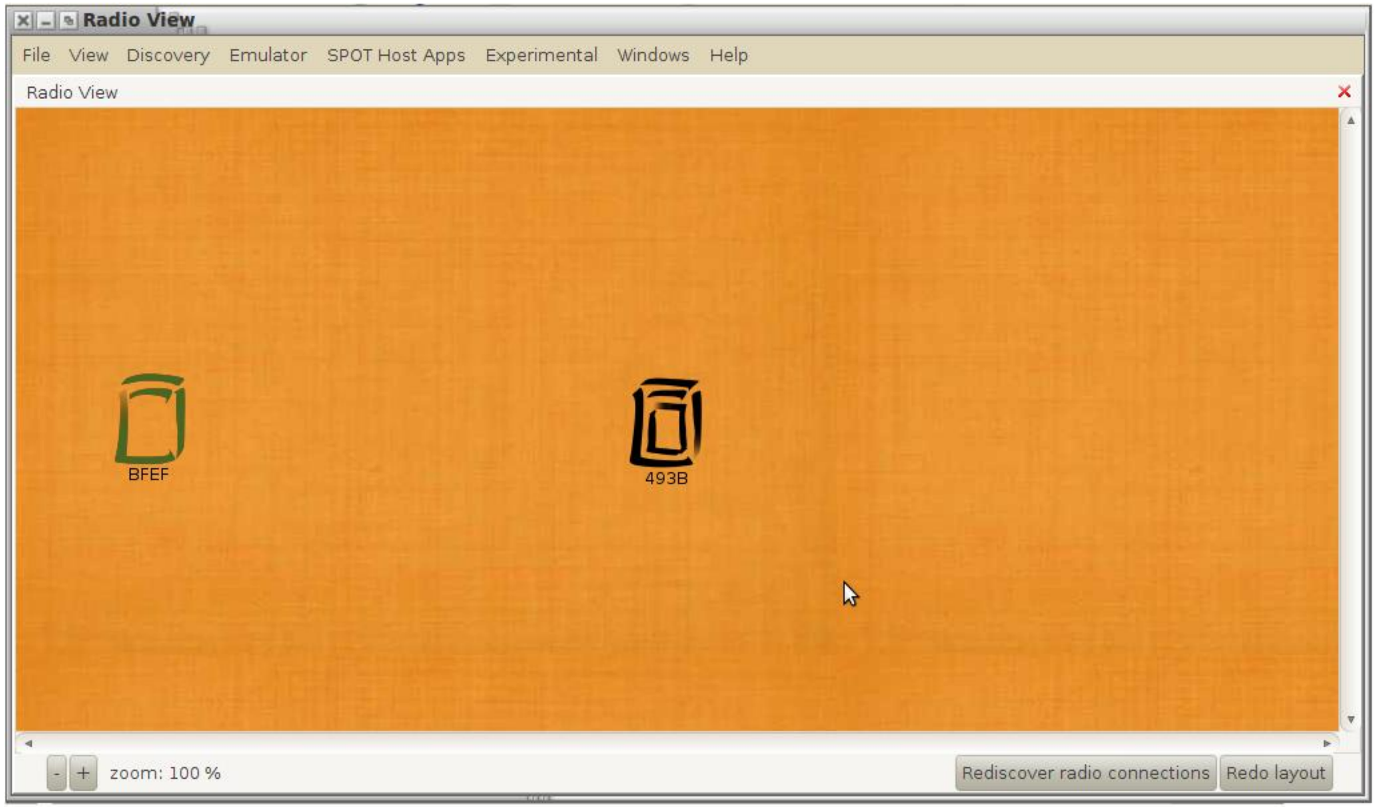

Radio View
File View Discovery Emulator SPOT Host Apps Experimental Windows Help
Radio View
BFEF
493B
- +
zoom: 100 %
Rediscover radio connections
Redo layout

Figure 56: Discovering of Sunspots within radio range.

8.5 Programming Outputs

8.5.1 Host Application

The outputs were produced to the Terminal when run the Host application:

```
/Thesis/sunspotproject/Host$ ant host-run
Buildfile: build.xml
-pre-init:
-do-init:
-post-init:
-warn-jar-file:
init:
-set-selector-for-host-run:
-override-warning-find-spots:
-prepare-conditions-for-find-spots:
-find-shared-basestation:
[echo] Checking for shared basestation...
[echo] Shared basestation result: false
-run-spotfinder:
[exec] Using Hardware Abstraction Layer (HAL) to probe Sun SPOTS...
-check-spotfinder-result:
-decide-whether-to-run-spotselector:
-run-spotselector:
[java] Please wait while connected Sun SPOTs are examined...
[java] RXTX Warning: Removing stale lock file. /var/lock/LCK..ttyACM0
[java] Experimental: JNI_OnLoad called.
-collect-spotselector-result:
[echo]
[echo] Using Sun SPOT basestation on port /dev/ttyACM0
-clean-up-spotselector-output-file:
[delete] Deleting: /home/user/SunSPOT/sdk/temp/spotselector-211732954
-spotselector-fail:
-decide-whether-to-start-basestation-manager:
-start-new-basestation-manager:
[echo] Starting new shared basestation...
-do-find-spots:
-pre-host-compile:
-do-host-compile:
-post-host-compile:
host-compile:
```

```
-pre-host-run:
-do-host-run:
[java] [radiogram] Adding: Server on port 8
[java] [radiogram] Adding: Server on port 100
[java] Etablished RadiogramConnection!
[java] Trying to read incoming message...
[java] [radiogram] Adding: Broadcast on port 100
[java] Total Broadcasting time: 15090
[java] 2010/11/30 17:46:46 - Received from Client: 0014.4F01.0000.493B, 7F00.0101.0000.D746, 60
[java] Total time to write and send a datagram: 55ms.
[java] 2010/11/30 17:46:49 - Sent reply to 0014.4F01.0000.493B
[java] open new connection on streamport 0014.4F01.0000.493B:60
[java] [radiogram] Adding: Server on port 60
[java] Etablished new RadiogramConnection to: 0014.4F01.0000.493B:60
[java]
[java]
[java] —– Trying to read values... —–
[java] 1ms. for receiving and read from a RadiogramConnection.
[java] —– End receiving values —–
[java] ********************************************
[java] Spot in list: org.sunspotworld.host.SunSPOT@a3bcc1
[java] Used time:3ms.
[java] 1. 2010/11/30 17:46:58
[java] Tue Nov 30 08:46:54 PST 2010, SunSPOT Id: 0014.4F01.0000.493B
[java] The current temperature is: 31.25 °C
[java] The accelerometer is: X=-0.03490870032223416, Y=-0.040279269602577876, Z=0.9693877551020409.
Tilt: -1.0
[java] Light is: Bright = 154
[java] ********************************************
[java] Total time for adding a SPOT to list: 8ms.
[java] Prepare to send values to Shepherd
[java] HttpsURLConnection is established.
[java] Setting properties...
[java] Teady to send..
[java] Sent to Shepherd!
[java] Response: OK = 200
[java] Closed all connections to Shepherd. [java]
[java]
[java] —– Trying to read values... —–
[java] 13762ms. for receiving and read from a RadiogramConnection.
```

```
[java] —– End receiving values —–
[java] **********************************************
[java] Spot in list: org.sunspotworld.host.SunSPOT@edc3a2
[java] 2. 2010/11/30 17:47:59
[java] Tue Nov 30 08:47:57 PST 2010, SunSPOT Id: 0014.4F01.0000.493B
[java] The current temperature is: 31.75 °C
[java] The accelerometer is: X=-0.056390977443609026, Y=-0.045649838882921595, Z=0.9962406015037595.
Tilt: -2.0
[java] Light is: Bright = 160
[java] **********************************************
[java] Total time for adding a SPOT to list: 4ms.
[java] Prepare to send values to Shepherd
[java] HttpsURLConnection is established.
[java] Setting properties...
[java] Teady to send..
[java] Sent to Shepherd!
[java] Response: OK = 200
[java] Closed all connections to Shepherd. [java]
[java]
[java] —– Trying to read values... —–
[java] 13895ms. for receiving and read from a RadiogramConnection.
[java] —– End receiving values —–
[java] **********************************************
[java] Spot in list: org.sunspotworld.host.SunSPOT@27391d
[java] 3. 2010/11/30 17:49:00
[java] Tue Nov 30 08:48:58 PST 2010, SunSPOT Id: 0014.4F01.0000.493B
[java] The current temperature is: 31.75 °C
[java] The accelerometer is: X=-0.040279269602577876, Y=-0.034908700322223416, Z=0.9747583243823846.
Tilt: -1.0
[java] Light is: Bright = 159
[java] **********************************************
[java] Total time for adding a SPOT to list: 10ms.
[java] Prepare to send values to Shepherd
[java] HttpsURLConnection is established.
[java] Setting properties...
[java] Teady to send..
[java] Sent to Shepherd!
[java] Response: OK = 200
[java] Closed all connections to Shepherd. [java]
[java]
```

```
[java] —– Trying to read values... —–
[java] 13882ms. for receiving and read from a RadiogramConnection.
[java] —– End receiving values —–
[java] *********************************************
[java] Spot in list: org.sunspotworld.host.SunSPOT@af8358
[java] 4. 2010/11/30 17:50:00
[java] Tue Nov 30 08:49:58 PST 2010, SunSPOT Id: 0014.4F01.0000.493B
[java] The current temperature is: 31.25 °C
[java] The accelerometer is: X=-0.05102040816326531, Y=-0.07250268528464017, Z=0.9801288936627283. Tilt:
-2.0
[java] Light is: Bright = 152
[java] *********************************************
[java] Total time for adding a SPOT to list: 18ms.
[java] Prepare to send values to Shepherd
[java] HttpsURLConnection is established.
[java] Setting properties...
[java] Teady to send..
[java] Sent to Shepherd!
[java] Response: OK = 200
[java] Closed all connections to Shepherd.
```

8.5.2 SPOT Application

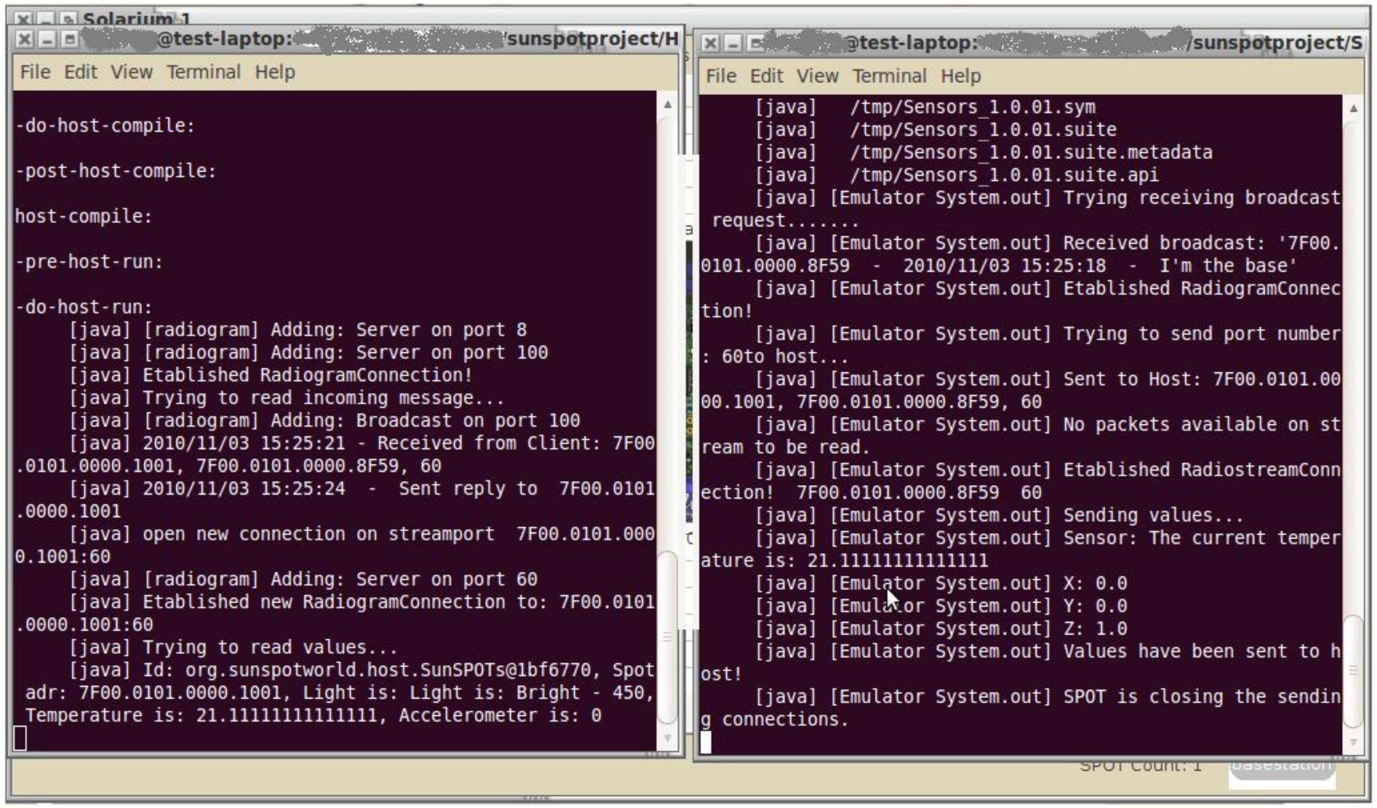
The outputs were produced to the Terminal when run the SPOT application on Emulator

```
/Thesis/sunspotproject/Sensors$ ant solarium
-do-run-solarium:
[echo] Java Runtime Environment version: 1.6.0_07
[java] CompilerOracle: exclude com/sun/squawk/Method.getParameterTypes
[java] CompilerOracle: exclude com/sun/squawk/SymbolParser.getSignatureTypeAt
[java] CompilerOracle: exclude com/sun/squawk/SymbolParser.stripMethods
...
...
...
[java] [Emulator System.out] Trying receiving broadcast request.......
[java] [Emulator System.out] 11464ms of elapse time for receiving broadcasting.
[java] [Emulator System.out] Received broadcast: '7F00.0101.0000.E292 - 2010/11/22 16:53:41 - I'm the base'
[java] [Emulator System.out] Etablished RadiogramConnection!
[java] [Emulator System.out] Trying to send port number: 60to host...
[java] [Emulator System.out] Sent to Host: 7F00.0101.0000.1001, 7F00.0101.0000.E292, 60
```

```
[java] [Emulator System.out] No packets available on stream to be read.
[java] [Emulator System.out] Time used in opening connection: 17
[java] [Emulator System.out] Etablished RadiostreamConnection! 7F00.0101.0000.E292 60
[java] [Emulator System.out] ————————————————————————
[java] [Emulator System.out] Sending values...
[java] [Emulator System.out] Time being used for sending values: 1657
[java] [Emulator System.out] Values have been sent to host!
[java] [Emulator System.out] SPOT is sleeping
[java] [Emulator System.out] ————————————————————————
[java] [Emulator System.out] ————————————————————————
[java] [Emulator System.out] Sending values...
[java] [Emulator System.out] Time being used for sending values: 1659
[java] [Emulator System.out] Values have been sent to host!
[java] [Emulator System.out] SPOT is sleeping
[java] [Emulator System.out] ————————————————————————
[java] [Emulator System.out] ————————————————————————
[java] [Emulator System.out] Sending values...
[java] [Emulator System.out] Time being used for sending values: 1664
[java] [Emulator System.out] Values have been sent to host!
[java] [Emulator System.out] SPOT is sleeping
[java] [Emulator System.out] ————————————————————————
```

8.5.3 Output in the Terminal using Real SunSPOT devices

Figure 57 shows the outputs of Host application and SPOT application in the Terminals.


Solarium 1
[redacted]@test-laptop:[redacted]/sunspotproject/H
File Edit View Terminal Help
[redacted]@test-laptop:[redacted]/sunspotproject/S
File Edit View Terminal Help
SPOT Count: 1
basestation

```
-do-host-compile:

-post-host-compile:

host-compile:

-pre-host-run:

-do-host-run:
     [java] [radiogram] Adding: Server on port 8
     [java] [radiogram] Adding: Server on port 100
     [java] Etablished RadiogramConnection!
     [java] Trying to read incoming message...
     [java] [radiogram] Adding: Broadcast on port 100
     [java] 2010/11/03 15:25:21 - Received from Client: 7F00
.0101.0000.1001, 7F00.0101.0000.8F59, 60
     [java] 2010/11/03 15:25:24  -  Sent reply to  7F00.0101
.0000.1001
     [java] open new connection on streamport  7F00.0101.000
0.1001:60
     [java] [radiogram] Adding: Server on port 60
     [java] Etablished new RadiogramConnection to: 7F00.0101
.0000.1001:60
     [java] Trying to read values...
     [java] Id: org.sunspotworld.host.SunSPOTs@1bf6770, Spot
 adr: 7F00.0101.0000.1001, Light is: Light is: Bright - 450,
 Temperature is: 21.11111111111111, Accelerometer is: 0
```

```
     [java]    /tmp/Sensors_1.0.01.sym
     [java]    /tmp/Sensors_1.0.01.suite
     [java]    /tmp/Sensors_1.0.01.suite.metadata
     [java]    /tmp/Sensors_1.0.01.suite.api
     [java] [Emulator System.out] Trying receiving broadcast
 request.......
     [java] [Emulator System.out] Received broadcast: '7F00.
0101.0000.8F59  -  2010/11/03 15:25:18  -  I'm the base'
     [java] [Emulator System.out] Etablished RadiogramConnec
tion!
     [java] [Emulator System.out] Trying to send port number
: 60to host...
     [java] [Emulator System.out] Sent to Host: 7F00.0101.00
00.1001, 7F00.0101.0000.8F59, 60
     [java] [Emulator System.out] No packets available on st
ream to be read.
     [java] [Emulator System.out] Etablished RadiostreamConn
ection!  7F00.0101.0000.8F59  60
     [java] [Emulator System.out] Sending values...
     [java] [Emulator System.out] Sensor: The current temper
ature is: 21.11111111111111
     [java] [Emulator System.out] X: 0.0
     [java] [Emulator System.out] Y: 0.0
     [java] [Emulator System.out] Z: 1.0
     [java] [Emulator System.out] Values have been sent to h
ost!
     [java] [Emulator System.out] SPOT is closing the sendin
g connections.
```

Figure 57: Outputs when run the applications on real Sunspots.

9 Bibliography

References

[1] UIC. Global digital railways communication system based on gsm. *White paper*.

[2] SELEX Communications. The ertms memorandum of understanding a cross-sector agreement to ensure ertms success. *ERTMS Factsheets*, 2009.

[3] UNIFE. Rgg200, rog100, and rgs100, gsm-r family handhelds. *White paper*, Version 2, 2008.

[4] Jernbaneverket. Konseptvalgutredning for ertms/etcs. *White paper*, May 2010.

[5] Jernbaneverket. Map over norway with line covered by gsm-r. *White paper*, (See footnote).

[6] M. Obenaus. Ertms is becoming a worldwide standard, unife statistics, news release. March 2009.

[7] E. Paren de Curzon. Ertms - a contribution to the creation of tomorrow's railway. 1999.

[8] S.T.G. Wooton K. Konrad D. Mandoc G. Dalton A.F. Neele. Gsm-r procedure guide. *UIC*, Version 5.0, February 2007.

[9] Unife The European Rail Industry. Ertms levels - different ertms/etcs application levels to match customers's needs. *ERTMS Factsheets*, 2009.

[10] R. Roman J. Lopez and C. Alcaraz. Analysis of security threats, requirements, technologies and standards in wireless sensor networks. *University of Malaga - Spain*, Springer-Verlag Berlin Heidelberg:Section 2, 2009.

[11] J. Søberg E. Munthe-Kaas. Data management in sensor networks. *University of Oslo*, INF5100, Autumn 2009.

[12] R. Ramachandran. Evolution to 3g mobile communication - 1. second generation cellular systems. *Resonance*, September 2003.

[13] J. Mounet B. Dalibard and D. Kaplan. Machine to machine stakes and prospects. *Orange Business Service, Syntec Informatique and Fing*, White paper, 2006.

[14] Klaus-Dieter Walter. Implementing m2m applications via gprs, edge and umts. *White paper - M2M Alliance, Germany.*

[15] K.-D. Walter. Solving m2m applications with linux. *Germany*, 2005.

[16] ©European Telecommunications Standards Institute. Machine- to- machine communications (m2m); functional architecture. Draft Specification of ETSI TS 102.690. Not for implementation, 2010-01.

[17] Consultant M.-B. PAUTET M. MOULY. The evolution of gsm. *France Télécom.*

[18] E. Jacob P. Síz J. J. Unzilla M. V. Higuero J. Matás M. Aguado. Railway signaling systems and new trends in wireless data communication. *University of the Basque Country Bilbao, Bizkaia (Spain)*, 0-7803-9152-7/05, 2005.

[19] Hans-Jörg Vögel Christian Bettstetter Christian Hartmann Jörg Ebergspächer. *GSM - Architecture, Protocols and Services.* A John Wiley and Sons, Ltd, Publication, third edition edition, 2009.

[20] UIC Project EIRENE. Eirene - european integrated railway radio enhanced network. system requirements specification. *GSM-R Operator Group*, Version 15, 17 May 2006.

[21] Telenor Objects - Norway. Shepherd® gsm connectivity. *White paper.*

[22] J. Mounet B. Dalibard, D. Kaplan, et al. Gps & selective availability q&a. *White paper.*

[23] A. K. Brown, and M. A. Sturza. Vehicle tracking system employing global positioning system (gps) satellites. *US Patent documents*, (US005225842A):page 1–2, 9 May 1991.

[24] G.T. Johnston G.J. Morgan-Owen. Differential gps positioning. *Electronics & Communication Engineering Journal*, February 1995.

[25] P. Deng. Introduction to spot - programming the real world. *CSSE University of Melbourne*, Presentation.

[26] A. Caicedo. The sun spot: Java^{TM} technology-based wireless sensor network. *Presentation*, Technology Evangelist, Sun Microsystems, 2006-2007.

[27] Sinem Coleri Ergen. Zigbee/ieee 802.15.4. Septemer 10 2004.

[28] K. Smolderen P. D. Cleyen B. Braem Ch. Blondia D.v.d. Akker. Tinyspot-comm: Facilitating communication over ieee 802.15.4 between sun spots and tinyos-based motes. Dept. of Mathematics and Computer Science, University of Antwerp - IBBT. Industrial Sciences and Technology, Belgium.

[29] J. A. Gutierrez I. Howitt. Ieee 802.15.4 low rate - wireless personal area network coexistence issues. *IEEE*, University of North Carolina at Charlotte, and RF/Communications Group - Innovation Center, Milkwaukee, 2003.

[30] Ray Rischpater. *Beginning Java™ME Platform.* Springer-Verlag New York, Inc, 2008.

[31] Telenor Objects - Norway. Shepherd® device integration. *White paper.*

[32] A. Herstad. Telenor - connected objects. *Telenor R&I.*

[33] Telenor Objects - Norway. Shepherd® managed services for m2m solution. *White paper.*

[34] A. Herstad. Telenor objects - shepherd platform introduction. *White paper.*

[35] Telenor Objects - Norway. Shepherd®api description. *White paper.*

[36] M. Cardei, J. Wu, and Sh. Yang. Low power hitch-hiking broadcast in ad hoc wireless networks. *Paper. Department of Computer Science and Engineering, Florida Atlantic University.*

[37] Inc. California Sun Microsystems. Sun™ small programmable object technology (sun spot) developer's guide. *White paper*, Red release version 5.0:page 19, 7. May 2009.

[38] Ph.D Paul Sanghere. *SCJP Exam for J2SE 5.* A Concise and Comprehensive Study Guide for The Sun Certified Java Programmer Exam. Springer-Verlag New York, Inc., 2006.

[39] Bill Shannon. *Java Platform, Enterprise Edition (Java EE) Specification.* Version 5. Sun Microsystems, Inc., May 2006.

[40] ARTEMIS Joint Undertaking - SP6. Artemis annex. *Internal document - ARTEMIS-2009-1*, (Proposal no. 100204), April 2010.

[41] Chief Technologist and Professor Josef Noll. Interoperable rail information system. *Presentation*, October 2010.