DSNAnalyzer: Backend for the Deployment Support Network

Master Thesis

September 22, 2006

Patrice Oehen

poehen@student.ethz.ch

Communication Systems Group

Department Of Information Technology And Electrical Engineering

Swiss Federal Institute of Technology (ETH) Zurich

Prof. Dr. Bernhard Plattner

Advisor: Dr. Philipp Blum, philipp.blum@siemens.com

Advisor: Dr. Martin May, may@tik.ee.ethz.ch

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Departement Informationstechnologie
und Elektrotechnik

Computer Engineering and
Networks Laboratory

Preface

This work was created within the scope of a master thesis at the Swiss Federal Institute of Technology (ETH) Zurich. The time of the work was 6 months which were spent at the R&D department of Siemens Building Technologies (SBT) international headquarters in Zug.

With this I certify that I have written this work myself and listed all used sources in the appendix.

I would like to thank my advisor, Philipp Blum very much for the great support and the inspiring talks. Additionally, I thank the whole SBT Communication and Wireless Group for the great working atmosphere.

Zurich, September 2006 Patrice Oehen

Abstract

Nowadays, wireless sensor network applications can be used mostly everywhere to help gathering detailed environmental information. But these systems also have very strong constraints such as low battery power and few computing resources. In the case of wireless systems for fire detection, even harder problems like high dependability, fault-tolerance and real-time constraints appear.

Because it is often difficult to have a testbed which is large enough to test a protocol in its full diversity, simulators are used to shorten the research and development phase. The problem that arises when using simulators is to have a radio model that is not only simple enough to be efficient but also complex enough to model the real-world behavior accurately enough. Nevertheless, experimental verification is needed to confirm the results gained from simulation.

The main goal of this thesis is providing a tool which helps to understand wireless communication in non-deterministic environments such as office buildings, where one key aspect is bringing simulation and reality closer together. Information which comes from the measurements is analyzed and serves as input for the simulation model. The tool is based on a second independent network connected to the network in production and provides direct access to the sensor nodes using Bluetooth technology, as if we had a cable to each of them.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ARQ	Automatic Repeat Request
BIC	Business Innovation Center
BER	Bit Error Rate
CSEM	Centre Suisse d'Electronique et de Microtechnique
CTI	Commission for Technology and Innovation, Federal Funding Agency
DSN	Deployment Support Network
EMC	Electromagnetic Compatibility
FEC	Forward Error Correction
FS	Fire Safety
FSK	Frequency Shift Keying
GloMoSim	Global Mobile Information System Simulator
GUI	Graphical User Interface
HTTP	Hyper Text Transfer Protocol
ISM	Industrial, Scientific and Medical
LED	Light-Emitting Diode
MAC	Media Access Control
MANET	Mobile AdHoc Network
PCB	Printed Circuit Board
PER	Packet Error Rate
PRR	Packet Reception Rate
RNP	Required Number of Packets
RPC	Remote Procedure Call
RUP	Rational Unified Process
RSSI	Received Signal Strength Indication
SBT	Siemens Building Technologies
SFD	Start Frame Delimiter
SINR	Signal-to-Interference-plus-Noise Ratio
SNR	Signal-to-Noise Ratio
UML	Unified Modeling Language
WSN	Wireless Sensor Network

1 Introduction

In this chapter we give an overview of what the master thesis is about and describe the basic challenges we had to solve.

1.1 Safety Critical Sensor Networks for Building Applications

In this section we define the scope by first introducing Wireless Sensor Networks (WSNs), then starting from the point of view of SBT which is building fire detection systems, explaining then the special topics of prototyping WSN and defining afterwards the DSN which should help to overcome the special challenges appearing in this kind of development area.

1.1.1 WSN

Moore's law states that every one and a half years the storage capacity on a chip is doubled. This has consequences for a lot of different application areas [Culler/Estrin 2004, 1]. Because computers are getting smaller and cheaper, they can also be used as sensor nodes which measure data about their environment. In order to collect this data, the information is usually sent to the sink[1] of the network. Sensor nodes which communicate by using radio are therefore called WSN.

One special constraint of WSN is the fact of limited resources. Sensor nodes usually have small batteries, limited processing speed, small storage capacity and only little communication bandwidth [Culler/Estrin 2004, 1]. These limitations make the design of protocols and applications for WSN very difficult.

Nevertheless, WSN are very popular nowadays and already used in a large variety of applications such as medical monitoring, animal monitoring, vehicle monitoring, building technology and many more [Lewis 2004, 1].

1.1.2 WSN at SBT

One activity of the Fire Safety (FS) division at SBT is developing fire detection devices which are based on current WSN technology. Up to now, WSN systems for fire detection are a niche product mainly used for places like museums and churches, where cable installations should be avoided. SBT has been selling WSN fire detection systems which work with a star topology[2] for several years [Siemens 2006].

This thesis is part of a Project (see 1.1.2.2) whose main goal is to provide an enhanced WSN fire detection system with mesh networking[3] functionality.

[1] A "sink" is a sensor node which collects the data from the WSN and delivers it to some kind of backbone network.

[2] In a star topology, a message is only sent from one target to another and cannot be routed over several Targets.

[3] Mesh Networking allows sending messages over several Targets using different paths.

1.1.2.1 Fire Detection

A *fire detector* (FD) belongs to a *fire detection network* (FDN) which represents a multi-hop WSN. A FD uses fire sensors to measure its environment and forwards a fire alarm on sensing fire conditions. This alarm is sent to the sink of the WSN which then, for example, can inform the fire brigade and enable sounders as well as special lights which are nodes of the WSN.

A FDN usually works in two phases: In a first phase, all FDs try to get information about the other FD in the system and initialize their internal data structures. In a second phase, the system starts to measure the environment and triggers alarms in case of a fire detection.

In this special kind of WSN application, there are new requirements such as a maximum latency for a message over multiple hops[4] and maximum energy consumption in order to design devices which can work for several years.

1.1.2.2 CTI Project

The multi-hop WSN for fire detection is developed with the support of the *Commission for Technology and Innovation (CTI)*[5]. The project runs under the name *Safety Critical Sensor Networks for Building Applications* and has a duration of 36 months. In contrast to traditional WSN technology, this CTI-Project is mainly about real-time constraints in WSN, which are needed in order to use WSN for fire detection systems as well as reliability and fault-tolerance. Additional goals of the project are a high dependability, small energy consumption of the Targets and performance guarantees. [Siemens 2006].

In this project, SBT works together with the following partners:

- *Siemens Switzerland, Business Innovation Center* (BIC)
- *Centre Suisse d'Electronique et de Microtechnique* (CSEM)
- *Swiss Federal Institute of Technology* (ETH)

1.1.3 WSN Prototyping

The classic approach in prototyping WSN is to specify the system, test it by using formal verification, simulate it and run the application on a small set of Targets.[6] The Targets are often connected to a computer over serial cables which enable the system developer to check if the Target software[7] works as intended and provides services such as reprogramming.[8] An easier way to check the system status is by using Light-Emitting Diodes (LED) connected to the Targets which indicate the system condition using different colors but provide only very limited debugging possibilities [Beutel et al. 2006, 1].

[4] One hop means sending a packet from one node to another.

[5] The "CTI" is a Swiss Federal Founding Agency. See http://www.bbt.admin.ch/kti/index.html?lang=de

[6] A "Target" is a sensor node which runs most often using battery power and has limited system resources (see 2.1.1.3).

[7] An alternative term is "firmware".

[8] An Example of this configuration is [MoteLab 2006].

1.1.3.1 Simulation

Because real WSN are considered to consist of several hundred Targets, experiments would be difficult at that scale. Therefore simulations are often used to gain information about the developed software [Reijers et al. 2004, 1].

One of the fundamental goals in designing a simulator is to create a radio model that is simple enough to be fast but also complex enough to model reality accurately.

SBT has decided to use GloMoSim[9] (see [GloMoSim 2006] for details) which is a simulation environment tailored for large wireless communication networks. GloMoSim allows simulating networks which consist of thousands of Targets and also supports multi-hop wireless communication [Nuevo 2004]. It also combines simulation of individual node behavior and sophisticated propagation models.

1.1.3.2 Measurement

A second way to prototype WSN is to run the developed software directly on the Targets. The results which can be retrieved by measurements are accurate and reflect the real-world properties of a WSN.

But measurements are not only used for prototyping. A further usage is when the system is finished and the developers and testers want to be sure that the system works as expected; therefore, long-term tests under real-conditions have to be performed. Another reason for doing measurements comes from the fact that most protocols rely on some assumptions about the link quality and its temporal properties. In order to provide this very important information to the system designers, initial tests have to be executed to study the behavior of the Targets in the specified environment.

1.1.3.3 Conclusion

Simulators are a good and cheap way to test protocols and applications for WSN, but they cannot substitute measurements in the real world. An explanation is the fact that radio wave propagation cannot be modeled simply and accurately enough to gain the same results as with a real-world system. Another explanation is the fact that in most simulations, only the protocol is changed at different runs while other factors are kept constant which can lead to inaccuracies of the test outcomes [Andel/Yasinsac 2006, 48]. Therefore, it is indispensable to run software on a sufficient number of Targets over a time which is long enough to detect all major software bugs.

Because of the advantages and drawbacks of the two prototyping methods described above, they have to be combined. We need simulation to overcome the complexity of setting up very large testbeds[10] for basic measurements, but we also need real-world measurements to prove the simulation results on the one hand, and to get detailed link quality information which can be used as input for the simulator on the other hand. I.e. the simulation can be used to improve the implementation and the measurements help to improve the simulation models.

[9] GloMoSim stands for *Global Mobile Information System Simulator*.

[10] A testbed is an experimentation platform for large projects. See http://en.wikipedia.org/wiki/Testbed

In order to gain the environmental information needed for the simulation as well as performing extensive measurements, we need a toolkit which enables easy access and control of all Targets of a WSN which is provided by the DSN discussed in 1.1.4.

1.1.4 Deployment Support Network

The *Deployment Support Network* (DSN) is a tool for prototyping WSN and was developed at the ETH Zurich to overcome the limitations in classical prototyping approaches.

When a WSN is getting larger than just a few nodes, the prototyping methods using serial cables described above reach their limits. It would not be possible to place a cable from a single PC to every Target, especially if the tests should be made in an office building or harsh outdoor environments. But not only the monitoring of the WSN is getting more complicated with an increasing number of Targets, also controlling and reprogramming would be infeasible. Nevertheless, it is very important for the system engineer to be aware of what exactly happens in the network; therefore, a kind of virtual connection to every Target is needed.

An approach for these issues would be to integrate the controlling and code dissemination[11] directly into the WSN currently being developed (see [Levis et al. 2004; Chlipala et al. 2003]). However, this technique has the drawback that a stable network is needed for code dissemination and monitoring purposes, which is usually not the case. A second drawback of this approach is the fact that the controlling and monitoring network would influence the "real" network because maintenance data would also be sent through the network [Hobi/Winterhalter 2005, 3].

In order to solve the problems described, the Deployment Support Network was developed. The DSN is a tool which helps to deploy large WSN and consists of a wireless multi-hop network of DSN-Nodes[12]. The network creates a virtual connection to the Targets such that each Target can be controlled from a single computer. The DSN-Network itself is a network which runs independently of the Target network, i.e. it does not influence the Target network[13].

The system developer can use the DSN to work as if every node is still connected to its computer using a cable and can therefore use the same controlling and monitoring tools. The DSN can be used over the whole production cycle (see Figure 1-1) of a WSN starting from initial code distribution, control the Targets, run exhaustive validation tests and monitor the system in its final state [Beutel et al. 2004].

[11] Code dissemination or code distribution describes the task of distributing new software to the Targets which afterwards can be used to reprogram them.

[12] A *DSN-Node* is a WSN-Node itself and small enough to be attached to any Target of the WSN which is to be implemented.

[13] This statement is analyzed in 2.5.1.8.

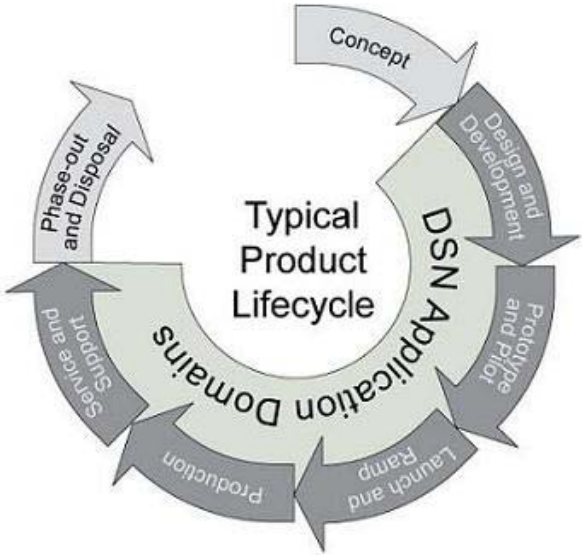

Figure 1-1: DSN Application Domain [BTnode 2006]

1.1.4.1 Hardware Platform

In order to provide the services described above, the DSN needs an underlying hardware platform for the DSN-Nodes which can build multi-hop networks, allows efficient data transfer and has an interface for being easy connectable to Targets of all kinds.

For these requirements, Bluetooth was chosen because of its reliable link layer, bandwidth and reliability. The fact of saving power is not that important because the DSN-Nodes do not have to live as long as the Target, which is designed to be energy-efficient. Another requirement which a DSN-Node should fulfill is to be easily adaptable to different kind of Targets.

The developers of the DSN have chosen BTnodes [Beutel et al. 2003; BTnode 2006] as the hardware platform for the DSN-Nodes (see Figure 1-2). The BTnode is a Bluetooth based device which is small and runs on an Atmel ATMega128L microcontroller and was designed for fast prototyping WSN [Beutel et al. 2004].[14]

Figure 1-2: BTnode Hardware [BTnode 2006]

[14] More Details of the BTnode are shown in 2.1.2.2.

1.1.5 Other Concurrent Work

This thesis is concurrently done with other subprojects:

- ETH:
    - Development of the DSN-Server (Master Thesis) (see 1.1.2)
    - Enhancement of the DSN-Node software (see 1.1.2)
    - Development of the Dwarf protocol
    - Development of the ClusterMAC protocol
- CSEM:
    - Development of the WiseMAC protocol
- SBT:
    - Implementation of the physical layer of the A80 Target
    - Analysis of the battery usage of the A80

The software developed in this thesis should be used to visualize the results of the simulations of the Dwarf, ClusterMAC and WiseMAC protocol. Additionally, it can be used for performing measurements with the A80 Targets from SBT.

1.2 Problem Specification

After having defined the scope of the thesis, we describe the specific problems to solve as well as the goals of the thesis.

In 1.2.2.4 we specify the tool which is developed in this thesis and discuss why such a tool is necessary when analyzing WSN.

1.2.1 Goals / Motivation

The main goals of this thesis are:

1. Integration of the existing Deployment Support Network (DSN) with the existing radio module by Siemens Building Technologies (SBT).
2. Design and implementation of a tool for analyzing, controlling and testing WSN-Applications. This tool has to handle both the output of a WSN simulator as well as the output of the DSN.
3. Design, planning and execution of tests using the developed tool, which should help to understand the behavior of radio waves in special environments such as office buildings and should prove the functionality of the tool. These test results should also help to improve the communication protocols introduced in 1.1.5.

1.2.2 System Overview

In Figure 1-3 we see the whole DSN System. The system consists of various components which are defined and introduced in the following sections. The work on the DSN Layer is done by people at the ETH, while the WSN-Layer is a part of this thesis.

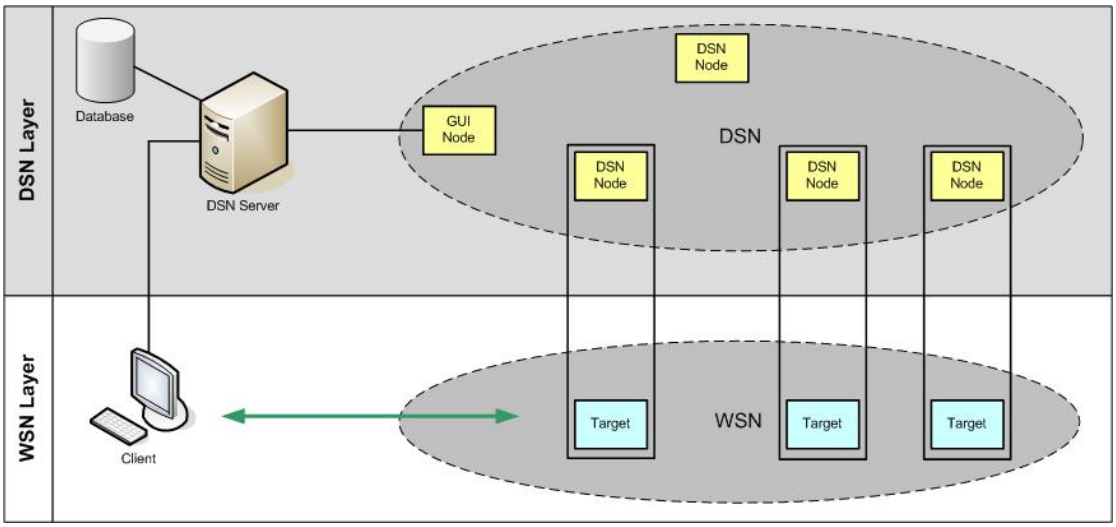

Figure 1-3: DSN System Overview

1.2.2.1 DSN-Node

The DSN-Node is a BTnode which runs the DSN software. It autonomously builds up a multi-hop network which is connected to the DSN-Server (see 1.2.2.2). The DSN-Nodes are usually placed in a box together with a Target (see 1.2.2.3), which allows controlling and monitoring services.

1.2.2.2 DSN-Server

The DSN-Server is the software which runs on a PC and allows clients to interact with the DSN. This server is mainly responsible for transforming user requests to DSN commands and handling the results. Another task is to collect logs[15] from the DSN-Nodes and store them correctly in a database as well as providing access functions to this database. The DSN-Server is used by clients where the *DSNAnalyzer* (see 1.2.2.4) is the client software in this thesis. This software is concurrently developed as a master thesis at the ETH.

1.2.2.3 Target

The Target is the wireless sensor node which is under development and is controlled and monitored and reprogrammed by the DSN-Node (see 1.2.2.1). In our case the Target is a Siemens A80 radio module (see 2.1.2.1) which is being developed with the software of this thesis by people at SBT.

1.2.2.4 Client (DSNAnalyzer)

The DSNAnalyzer is a backend software application which communicates with the DSN-Server (see 1.2.2.2) and allows the user to control, monitor and perform measurements on the Targets connected to the DSN-Nodes. A design decision was to be as independent of the DSN itself as possible, i.e. the DSNAnalyzer just takes advantage of the DSN but is not the backend for the DSN-Nodes.

[15] A *log* or a *log entry* is a string which is generated at the DSN-Node. Events which are generated at Targets are stored as log entries at the DSN-Node.

The DSNAnalyzer is the intended tool for bringing simulation and real-world measurements closer together and is the result of the second goal described in 1.2.1. Therefore, it has the possibility of not only getting input from the DSN but also from performed simulations.

The functionality of the DSNAnalyzer can be divided into the following three main parts:

- The *Control View* is a place where the actor[16] can completely control the Targets of the DSN as well as reprogramming them. There is a possibility to open a terminal to every single Target as well as controlling its state.
- The *Analyze View* allows the user to analyze *Traces* which are a set of events[17] that occurred either on the Targets or during a simulation. Events are displayed on three types of sub-views including a sequence chart where events are shown on timelines, a 3D location map where the user can see the position of the Targets and a log window where the textual representation of the events are displayed.
- In the *Analyze Test View*, the user can analyze results of previous test-runs[18] in the Control View in the form of charts. This view is very easy extendable and allows the advanced user to draw exactly the charts which are important for the given WSN.

With all the functionality mentioned above the DSNAnalyzer allows the user to gain a deep knowledge about system and radio propagation characteristics of the WSN and its radio hardware which is in development. Additionally, this tool could be used to do intensive validation checks of a final WSN as well as taking full control of all the Targets in the WSN.

1.2.3 Channel Measurements

WSN communicate over radio waves which have a well-defined propagation in the blank space. However, if we are in difficult environments such as office buildings, radio wave propagation becomes rather unpredictable due to reflection, diffraction and attenuation. Additionally, making predictions about link quality using low-power radio is much more complicated than with e.g. Wireless LAN [Woo/Culler 2002, 1].

In order to provide relevant information about the radio wave propagation in this special environment to the designers of wireless protocols, we need to perform channel measurements what refers to the third goal mentioned in 1.2.1. These measurements should give an insight into link quality, its symmetry, resistance to interferences and temporal behavior.

For performing these measurements, the DSN system described in 1.2.2 extended with a new backend application should be used.

1.3 Contributions

The following items are my specific contributions to the subject matter of this master thesis:

- Put the Adapter Board (see 2.1.2.3) into operation. Define, implement and execute initial tests on the A80 to check if the hardware works properly.

[16] In this thesis, the term *actor* is used as a synonym for *user*

[17] An event is a description of what, where and when something happened (e.g. A message sent by Target A at time t).

[18] In our case, these tests are link quality tests.

- Find an appropriate way to prototype the A80 radio module and to integrate simulation output with the output of measurements using the DSN. This consists of requirement analysis, design and implementation of the DSNAnalyzer which is the main contribution of this thesis.
- Integrate the DSN-Server and DSN-Node software which is being designed and developed by people at the ETH (TIK).
- Design and implement test scenarios which help to understand the radio wave propagation in office buildings.
- Take measurements of the DSN system installed at SBT and assess its current performance.
- A method and a schedule to achieve the contributions mentioned above within the given time.
- Perform and discuss link quality measurements with the A80 Target and the developed software.

1.4 Document Overview

This master thesis document is structured in three parts: Chapter 2 describes the developed tool as well as its evaluation. Chapter 3 describes the tests performed using the tool and analyzes the results.

2 DSNAnalyzer

The DSNAnalyzer is the main contribution of this thesis and is a backend software application which allows the user to control and monitor the Targets connected to the DSN-Nodes. In this chapter, we first give an overview of the whole DSN installed and configured at SBT. Afterwards, we mention the methodical processing of this thesis which is followed by a section about requirements and Use Cases. After that, we describe the design and implementation as well as the detailed evaluation of the developed system.

2.1 DSN System Overview

This chapter provides an overview about the whole DSN System with all its components and not only about the DSNAnalyzer implemented in this thesis. The DSN is a network which consists of DSN-Nodes connected to Targets. These DSN-Nodes can be controlled by the DSN-Server and are able to interact with the Targets. The DSN-Server is controlled by a client application which is a visible backend to the system developer (see Figure 1-3).

2.1.1 System Components

The DSN could be split up into two main parts (see Figure 1-3):

- The *DSN Layer* which already exists and consists of the DSN-Nodes and the DSN-Server.
- The *WSN Layer* which is specific to the application and consists of the Target Node software as well as the client which is in this thesis a Graphical User Interface (GUI) Application.

2.1.1.1 DSN-Node

The BTnodes (see 2.1.2.2) which are usually directly connected to a Target (see 1.2.2.3) are called *DSN-Nodes* and are used for controlling and monitoring the Targets. A DSN-Node is running the open source software *JAWS*[19] which is basically used to build up a multi-hop network between all available DSN-Nodes.

The DSN software, which is written in C, supports on the one hand functions on the DSN-Node itself (e.g. reset, get log entries) and on the other hand functions for the Target (e.g. reset, reprogramming, toggle pins, send commands, receive events).

There is a special kind of DSN-Nodes called *GUI-Node*[20] which runs the same software as the other nodes but in a different mode. The GUI-Node is connected to the DSN-Server (see 2.1.1.2) over USB and is therefore the connection between the server and the multi-hop network. The GUI-Node can communicate with all other DSN-Nodes in the system by using a Remote Procedure Call (RPC) command which allows executing an arbitrary command on one or multiple DSN-Nodes.

[19] See also http://www.btnode.ethz.ch/Projects/JAWS-DSNSpecification

[20] This node is called GUI-Node because of historical reasons when it was directly controlled by a GUI-Application. The GUI-Node could also be referred to as *Sink* of the DSN.

The software release we use for the DSN-Node uses 50Kbytes out of the 180Kbyte memory which are totally available for storing log entries.[21]

2.1.1.2 DSN-Server

The DSN-Server is the software that connects the client to the DSN-Nodes. This piece of Java software was developed concurrently with the client by another master-student of the ETH and is an improved version of old DSN server application (see [Zimmermann 2005]).

The DSN-Server interconnects the clients to the DSN-Nodes over the GUI-Node. It provides an XML-RPC interface for the following services:

- Executing commands on Targets/DSN-Nodes
- Programming Targets/DSN-Nodes
- Reset Targets/DSN-Nodes
- Get connection information for the DSN-Nodes
- Get program version of the DSN-Nodes
- Get events from the database

2.1.1.3 Target

The Target is the sensor node which is under development and is therefore monitored and controlled by a DSN-Node (see 1.2.2.1). In this thesis, the Target is the Siemens A80 radio module (see 2.1.2.1). The interface between the Target and the DSN-Node is described in 2.1.3.4.

2.1.1.4 Client (DSNAnalyzer)

The client component (introduced in 1.2.2.4) is the main contribution of this thesis and called *DSNAnalyzer*. It is a Java backend application for the DSN and used for controlling, monitoring and reprogramming the Targets as well as to perform measurements. The design of the client and its functions are described in more detail in 2.4.

2.1.2 Hardware

In the following subsections we describe the hardware of the DSN-Node, the Target and the Adapter Board which is the device that combines them.

2.1.2.1 Siemens A80

The Siemens A80 (see Figure 2-1) is a Target based on a MSP430[22] microprocessor and has a Texas Instruments Chipcon CC1020 UHF Radio Transceiver. This hardware is designed for low power applications in the Industrial, Scientific and Medical (ISM) domain.

[21] Each log entry takes 14 bytes overhead for the data structure

[22] See also http://focus.ti.com/mcu/docs/mcuprodoverview.tsp?sectionId=95&tabId=140&familyId=342

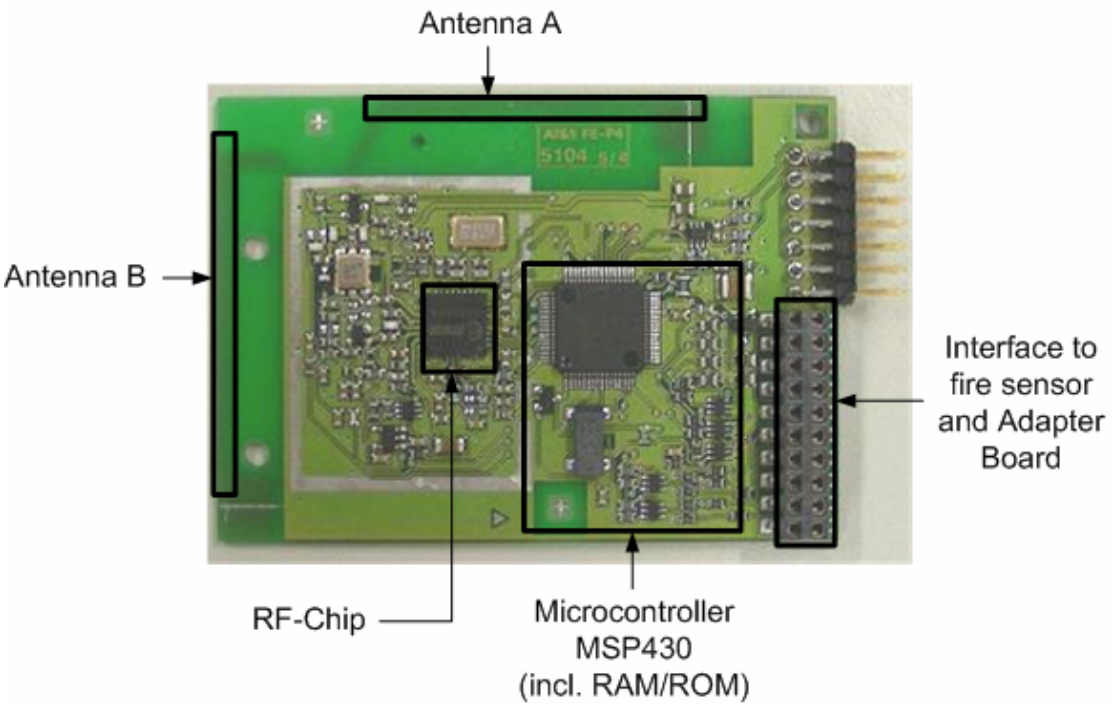

Figure 2-1: Siemens A80

The CC1020 has a frequency range from 402 MHz – 470 MHz and 804 MHz – 940MHz, a sensitivity of -118 dBm and includes an RSSI indicator [Chipcon 2006]. The CC1020 has a data rate of 4.8Kbit/s and supports (among other) *Frequency Shift Keying* (FSK) for modulation.

In the A80 Software, antenna A is called antenna 0 and antenna B is called antenna 1.

The CC1020 supports antenna diversity which makes it possible to change between two antennas. Because we are sending on 4.8Kbits/s and have a channel spacing of 25kHz we have a receiver sensitivity of -112 dBm.

2.1.2.2 BTnode rev3

The BTnode rev3 is the hardware platform of the DSN-Nodes. It is a versatile computing platform based on a Bluetooth radio module. Additionally, it includes a second low-power radio module (which is also used on the Berkeley Mica2 Motes[23]) as well as a microcontroller (see Figure 2-2). These two Radio modules can be turned on and off independently but can also be used simultaneously.

The BTnode is primarily used for prototyping WSN and Mobile AdHoc Networks (MANETs).

The System-Software of the BTnode in this setup is called *BTnut*[24] which consists of the Nut/OS operating system, the Bluetooth stack and BTnode specific drivers.

[23] See also http://www.xbow.com/products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf
[24] See also http://sourceforge.net/projects/btnode

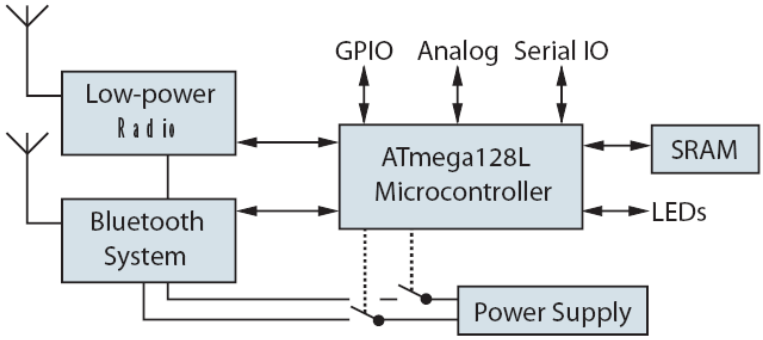

Figure 2-2: BTnode rev3 System Overview [BTnode 2006]

The BTnode contains 64+180 Kbyte SRAM, 128Kbyte Flash ROM and 4 Kbyte EEPROM. The size is 58.15 x 33 mm [BTnode2006].

2.1.2.3 Siemens Adapter Board

The *Adapter Board* is the hardware platform where the DSN-Node (see 2.1.1.1) and the Target (see 1.2.2.3) are connected together (see Figure 2-3). The Adapter Board is a *Printed Circuit Board* (PCB) which mainly consists of the connector for the BTnode (DSN-Node), the connector for the A80 (Target) and a plug for power.

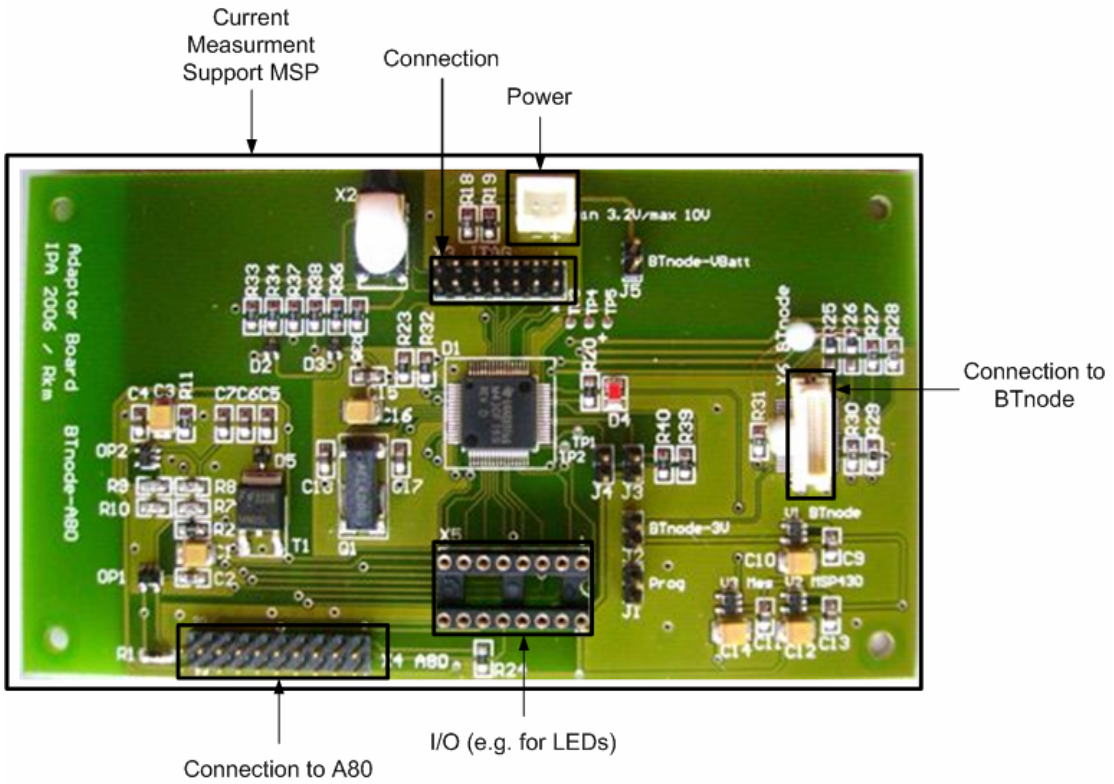

Figure 2-3: Siemens Adapter Board

Additionally, it includes the *Support MSP* which is an MSP430[25] microprocessor used for the following features:

- Current measurements (Up to now, the current can only be measured manually but it is planed to extend the BTnode software such that it can communicate with the Support MSP over I2C serial bus and retrieve current measurements.)
- Spy feature (Allows the user to connect a cable to the onboard cinch plug and follow the communication between the BTnode and the A80. This feature can also be used to work with the A80 and the Adapter Board without a DSN.)
- Possibility for externally influence the Target.

The communication between the DSN-Node and the Targets takes place over a serial interface (see 2.1.3.4).

The Adapter Board is placed into a box (see Figure 2-4) which includes a battery pack and protects the hardware. The battery back allows the following battery configuration:

- 1 x 9V
- 1 x AA 3.6V
- 4 x AA 1.5V

Figure 2-4: Siemens "Blue Box"

On the bottom of the box there are magnets which are used to put the box onto a magnetic surface (e.g. the ceiling of the office at SBT as shown in Figure 2-5).

[25] See also http://focus.ti.com/mcu/docs/mcuprodoverview.tsp?sectionId=95&tabId=140&familyId=342

Figure 2-5: "Blue Box" at the ceiling close to a fire detector

2.1.3 Protocols / Interfaces

As already mentioned, we have four different components in the DSN:

- Client
- DSN-Server
- DSN-Node
- Target

In the following section, the communication protocols among these subparts are described. Whenever possible we try to use standardized and generic protocols which we can use as much as possible out of the box and had only to adapt them to our specific system needs. This was done to make the system easy accessible by other software.

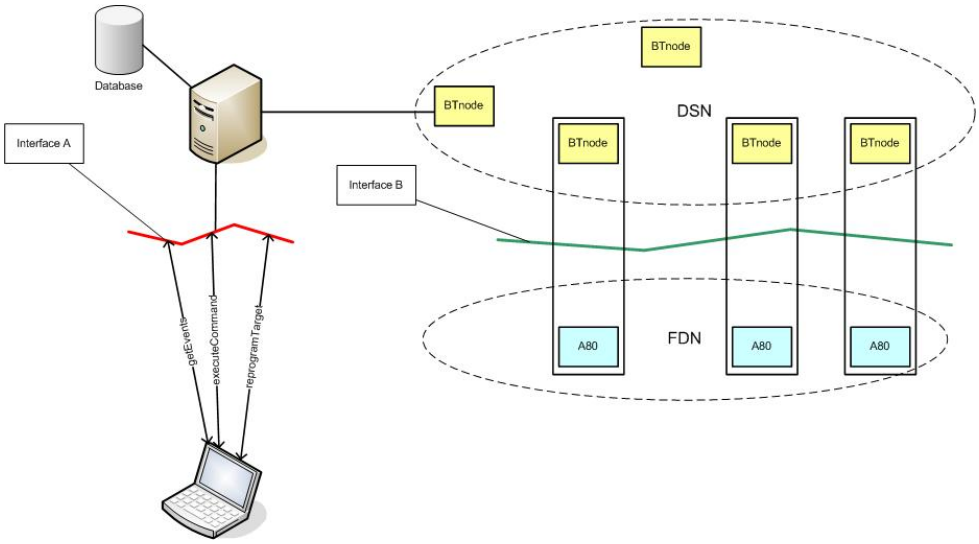

Figure 2-6: DSN Interfaces

The most important interfaces are shown Figure 2-6 in where Interface A is described in 2.1.3.1 and interface B in 2.1.3.4.

An overview of the protocols used between the DSN interfaces can be seen in Figure 2-7.

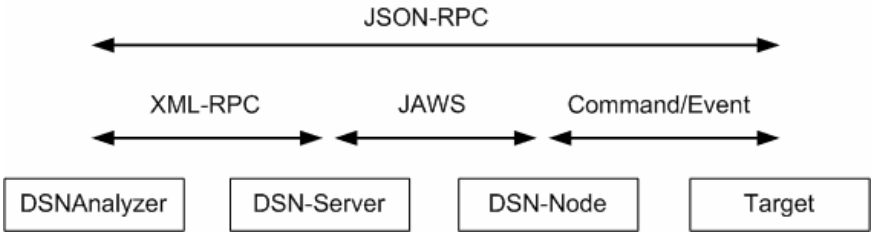

Figure 2-7: RPC Schema

2.1.3.1 Client – DSN-Server

The DSN-Server provides a bunch of methods which could either be used to control the DSN or to get information out of the server database (see interface A in Figure 2-6).

For the communication between the Client (the DSNAnalyzer) and the DSN-Server, we have chosen to use XML-RPC for the protocol.

XML-RPC is a specification about doing Remote Procedure Calls (RPC) using the Hyper Text Transfer Protocol (HTTP) as transport protocol. XML-RPC is able to run on different platforms, is designed to be very simple and easy to use and allows exchanging complex data structures [XML-RPC 2006].

The implementation we used was Apache XML-RPC[26] which is a java implementation of the XML-RPC standard.

2.1.3.2 Client – Target

For the communication between the Client (DSNAnalyzer) and the Target (A80), we have chosen to use a kind of an RPC interaction.

A simple lightweight RPC protocol is *JSON-RPC*[27] which provides a common representation of data transferred from the client to the server. In our case, the client is the DSNAnalyzer and the server the Target (A80). The reason why we do not take XML-RPC is, that the JSON-RPC standard is easier to implement on the Target because it is easier to write a parser for JSON-RPC than for XML-RPC. Additional advantages are that a parser for JSON-RPC is smaller and that the protocol includes less overhead.

JSON-RPC allows the data types String, Number, Boolean, Array and Dictionary[28] and the composition of these.

JSON-RPC uses *Requests* from the clients which are answered by *Responses* from the server (A simple message flow is shown in Figure 2-8). A Request includes the name of the method, the parameters and an ID. A Response includes the result, an error message and the ID of the Request. The exact data flow from the Client to the Target it shown in Figure 2-8.

[26] See also: http://ws.apache.org/xmlrpc/
[27] See also http://json-rpc.org
[28] A dictionary is a set of key-value pairs.

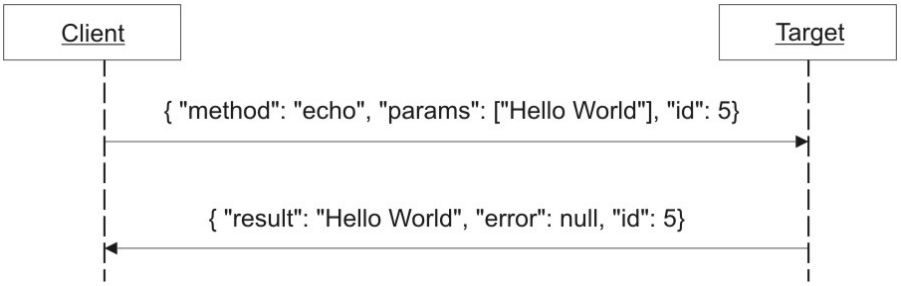

Figure 2-8: Client/Target Sequence Diagram

2.1.3.3 DSN-Server – DSN-Node

The DSN-Server communicates with the DSN-Node over the GUI-Node (see 2.1.1.1). Therefore, it uses the commands provided by the *JAWS* software on the DSN-Nodes.

The DSN-Server uses the following function to execute a command on the Target: "tg cmd <cmd>", where <cmd> refers to the command which should be executed on the Target. This command is in the JSON-RPC format (see 2.1.3.2). The DSN-Node immediately sends back an acknowledgement which states that the command was executed (see Figure 2-9).

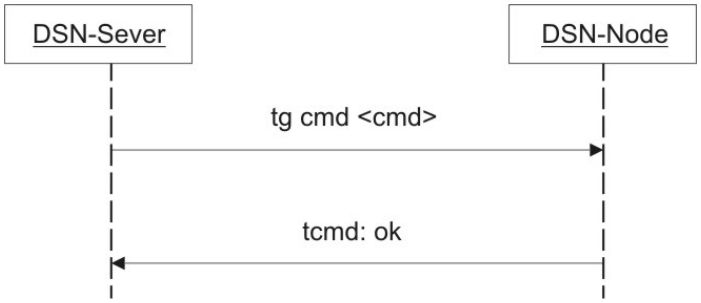

Figure 2-9: DSN-Server/DSN-Node Sequence Diagram

The *JAWS* software supports a lot of other commands which allow the DSN-Server to distribute software in the DSN, get log entries, reset DSN-Nodes, retrieve the battery status and many more[29].

2.1.3.4 DSN-Node – Target

The interface between the DSN-Node and the Target (shown as interface B in Figure 2-6) consists of two components, the hardware and the software component.

Hardware Component [30]

The hardware component defines the wiring and electrical specification. The pin connections are described in Table 2-1.

[29] A detailed list of all available commands can be found at http://www.btnode.ethz.ch/Projects/JAWS-TerminalCommands

[30] For a detailed description of the hardware interfaces see http://www.btnode.ethz.ch/pub/uploads/Projects/a80_pinout.txt

Adapter Logic	AVR Pin	BTnode Pin	Function
UART0_RXD	PE1	4	UART0_TXD
UART0_TXD	PE0	5	UART0_RXD
RESET	PB0	27	SS
PROG	PB4	14	PB4
IRQ	PE6	29	PE6
TG_BAT	PF0	10	PF0
TRIGGER	PE3	31	PE3
SDA	PD1	14	SDA
SCL	PD0	13	SCL

Table 2-1: A80 Target Adapter [BTnode 2006]

The Target supports baud rates up to 38400 baud.

Software Component

The software component of the interface to the Target allows the client to do the following operations on the Target (see Figure 2-11) [BTnode 2006]:

- Target Programming (Flash the Target with new software)
- Power control of the Target (Power on/off and battery status)
- Control the pins of the Target
- Send commands[31] via UART (see)
- Receive events[32] via UART (see)

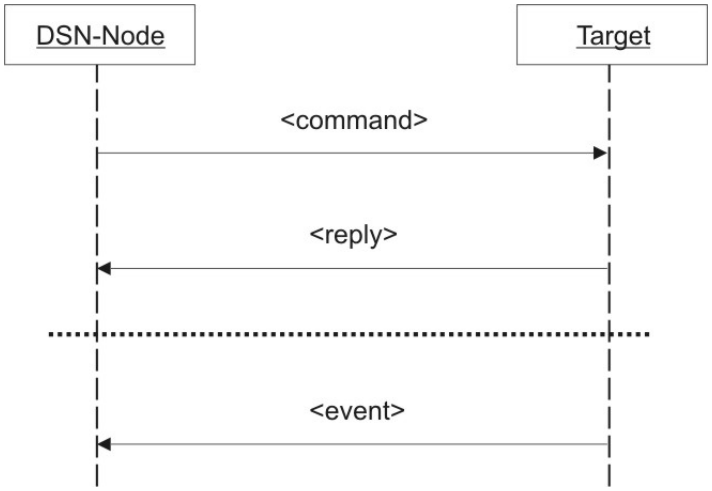

Figure 2-10: DSN-Node/Target Sequence Diagram

The interaction between the Target and the DSN-Node consists of messages (commands sent by the DSN-Node and replies and events sent by the Targets). These messages are shown in

[31] A command is an arbitrary null-terminated string
[32] An event is an arbitrary null-terminated string

Figure 2-10 and defined in Table 4-2 in the Appendix. Note that the prefix E, W, I or D and the whitespace is removed by the DSN-Node; the rest is sent to the DSN-Server.

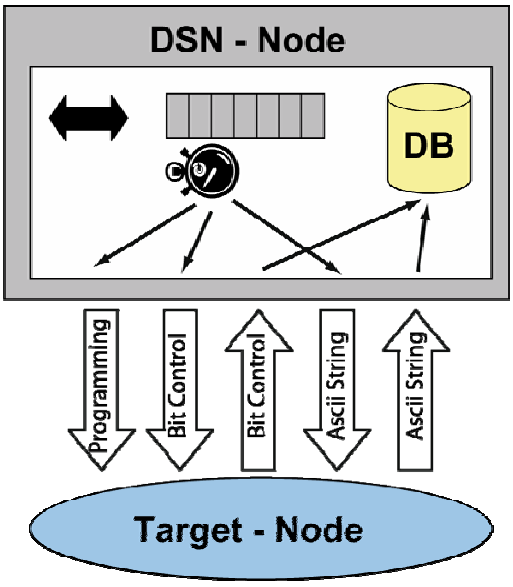

Figure 2-11: DSN-Node Target Adapter [BTnode 2006]

2.2 Methodical Proceeding

Because this thesis is part of a large project and had to deliver its results within a given time, we have chosen to use methodical engineering whenever possible. The methods described above provided a clear understanding about the problems which had to be solved and enabled the developer and the users to talk in the same language about problems and improvements.

A guideline during this thesis was to spend more time in design than in implementation. But whenever design decisions were not clear, we developed a prototype in order to make correct decisions.

The methodical procedure of the whole thesis was to develop use cases and system requirements (see 2.2.1), doing design and implementation which was followed by a testing phase (see 2.2.4). The whole process was supported by a lightweight version of the Unified Process and the help of the Unified Modeling Language (UML) (see 2.2.2) and documentation tools like a ticketing system and a Wiki website (see 2.2.6).

2.2.1 Requirements / Use Cases

Requirements are used to define the capabilities a system must have. The act of finding requirements, organize, track and write them down is referred to as *Requirements Management* [Leffingwell/Widrig 1999, 16].

Use Cases are used to describe an interaction between the user and the system. The whole system can be described by listing all these Use Cases. Use Cases are usually listed in a document where most important information such as a description and the main interaction flow are shown for each of them [Ariadne 2001, 39 ff.]. Use cases are principally used for

getting the functional requirements of the system which should be designed and implemented [Fowler 2003, 99].

Even if only one person is working fulltime on this thesis, there are reasons for doing a lightweight requirements management. There are a lot of developers involved which implement systems that later should be integrated with the software developed in this thesis. Additionally, this thesis has also its customers, namely SBT and ETH Zurich, which all have their own vision of what the system should be supposed to do.

Because of the reasons mentioned above we have chosen to start the project by defining the Use Cases of the system which should be developed and are described in 2.3.

2.2.2 Unified Process / UML

A secondary goal of this thesis is to use a lightweight version of a developments lifecycle model even if only one person is working 100% on the project in order to have a clear defined project structure. Therefore, we use a unified process with the help of the UML.

2.2.2.1 Unified Process / Project Lifecycle

When talking about software development processes the *Waterfall*[33] and the *Spiral Model*[34] often appear. As these processes lack in flexibility or are difficult to manage, we have chosen the *Iterative, Incremental Framework*[35] which is an extended version to the spiral model and consists of the following four phases:

1. *Inception* (In this phase the scope of the project is defined what results in a first vision document and an initial project plan.)
2. *Elaboration* (In this phase, most of the analysis work is done what ends in a fundamental understanding of the problem and the first Use Case- and Class-Diagrams (see 2.2.2.2)).
3. *Construction* (In this phase, there are several iterations which consist of the phases which we know from the waterfall model (see 2.2.2.1). These short iterations are repeated until there is a running system.)
4. *Transition* (At the end the final product is deployed. This phase includes beta-releases and large testing.)

In this thesis, we follow this process whenever possible and also used it for creating the project plan (see Figure 4-1 in the Appendix). The most work will be done in the construction phase where we try to do iterations with 2 week length. All the time it is important that we have a running system which can be tested by the developer as well as by first alpha users in order to detect bugs as early as possible. During the phases of the process we use, we try to apply the UML (see 2.2.2.2) whenever needed to have a common understanding of the problem [Ariadne 2001, 10 ff.].

2.2.2.2 UML

The UML stands for *Uniform Modeling Language* and consists of graphical notations for designing and describing software [Fowler 2003, 1]. The UML is used in the software

[33] In the waterfall model, each phase (e.g. design or implementation) has to be completed in order that the next phase can start.

[34] In the spiral model all phases (e.g. analysis, design, implementation, testing) are executed several times until the project is finished.

[35] The widely known *Rational Unified Process* (RUP) is an example of this process.

development process to support a common understanding of the problem in design questions. It allows different stakeholder (e.g. designers, testers, coders) to talk about the same problem by using diagrams in addition to textual specifications.

In the following, we list the UML-Diagrams we took advantage of in this thesis:

- *Use Case Diagram* (This diagram shows the behavior of the system from the viewpoint from the user.)
- *Class Diagram* (The Class Diagram consists of graphical representation for classes and their dependencies. The more detailed a Class Diagram is, the more attributes, operations and dependencies are drawn.)
- *Sequence Diagram* (This diagram shows the call flows of a set of objects. Each object in the diagram has a timeline and an arrow to the timeline of another object for a call to that object.)
- *Package Diagram* (The package diagram shows the packages of the software and its dependencies.)

Additionally to the diagrams described there exist many more like the *State Diagram*, *Component Diagram* and the *Collaboration Diagram* [Ariadne, 26 ff.].

2.2.3 Software Design Guidelines

At the beginning of a software project, the design is often clear and beautiful. But after the implementation and the first change requests, the design often begins to change, little hacks are introduced in order to be fast and as a consequence, the design becomes ugly [Martin 2000, 1].

In order to avoid the scenario described above, design guidelines can be used which help to leave the system in a clean and stable state, even if the requirements change or features should be added.

We can roughly distinguish between design on the high package level which is about the contents of packages and their dependencies, and the class level design which describes some fundamental principles of object orientation when designing classes.

2.2.3.1 Class Design Guidelines

For a system to be well designed it should be easy to understand, easy to reuse and easy to change. In order to enable good design, we need some simple basic guidelines [Martin 2000, 63].

The following principles from [Martin 2000, 64 ff.] for object oriented software help the developer to achieve a good design:

- *Single Responsibility Principle* (A class should not include every operation related to it, it should only have a single responsibility.)
- *Open Closed Principle* (Software should be extendable without the need for modification.)
- *Liskov Substitution Principle* (Users of a base class should also be able to use a derivative of the base class.)

- *Dependency Inversion Principle* (Classes should not depend on classes that are concrete and change often.)
- *Interface Segregation Principle* (Clients should not have to recompile classes just because methods have changed which are not used by them.)

2.2.3.2 Package Design Guidelines

In order to organize a design package level and not class level, principles have to be used. The following six principles which we tried to use in this thesis are mainly about the question, which classes should be in which packages [Martin 2000, 16 ff.]:

- *Release Reuse Equivalence Principle* (What should be reused together should also be together in a release.)
- *Common Closure Principle* (Group classes together about we think they change together.)
- Common Reuse Principle (Classes which are not reused together should not be placed in the same package.)
- *Acyclic Dependencies Principle* (There must not be any dependency cycles in the package diagram.)
- *Stable Dependencies Principle* (If package A changes more often than package B, than package A should depend on package B.)
- *Stable Abstractions Principle* (Stable packages should consist of abstract classes.)

2.2.3.3 Conclusion

The design guidelines mentioned above should be applied reactively, what means that the developer should not spend hours in just trying to design the code using the guidelines, but should use them when structural problems occur [Martin 2000, 82 ff.].

In this thesis, we try to agree on design guidelines whenever possible but we only apply them where they are useful, because otherwise we would create e.g. a lot of interfaces and abstractions which are not really needed. Because we do not very often use class inheritance the class design guidelines cannot be applied everywhere. As we have few structural problems, we do not spend a lot of time making our classes compatible with the guidelines mentioned above. However, we strictly try to use the Release Reuse Equivalence Principle, the Common Closure Principle and especially the Acyclic Dependencies Principle which is fully fulfilled (see Figure 2-14).

Because the classes in this thesis are very specific and a lot of them are not directly thought to be reused, not all classes follow the entire guidelines in the first release.

2.2.4 Testing and Deploying

As mentioned above, testing is very important in developing reliable systems. Therefore, we use JUnit[36] for performing unit tests on the classes which could be tested alone. Most of the classes which do not have an associated unit test are GUI-Related classes. There exist

[36] JUnit is an open source unit test tool for Java which could easily be integrated into the IDE Eclipse. See also http://www.junit.org/index.htm

approaches to create unit tests for GUI-Related classes (see [Hammel 2006]) but we do not use them because most of the critical behavior always has to do with several components.

A second method which we use is the distribution of alpha versions to the people related to this project. Because letting users, instead of computers, check the software reveals much more bugs, we should be able to create a final release which has an advanced stability and is ready to be used by WSN programmers. Additionally, we will do a lot of testing by using the application to perform link quality measurements (see chapter 3). The alpha users cannot only help in finding bugs, but also propose conceptual improvements.

The third way to help testing the system was the application of a ticketing system which is accessible to the people of SBT. This system allows the testers to create tickets with a detailed error description as well as information about the state and importance of the problem. These tickets are then available to the programmer of the software.

2.2.5 Documenting

When working on large projects which have later to be maintained by other people, clear code conventions are unavoidable. Using code conventions also helps finding bugs because about 80% of programming work lies in maintenance. Even if a software project is correctly implemented, uniform conventions help people to add new features because they need less time to go trough the code. Therefore, we used the Java code conventions from [Sun 1997]. Additionally, we used the software *CheckStyle*[37] which checks the source code for formats which are against the code conventions.

For the documentation of implementation details we use *Javadoc*[38] and do not include implementation details in the written thesis.

2.2.6 Wiki Website

In order to track the system development we used a Wiki system where all the new steps were listed and the interfaces were defined. Additionally, we used a ticket system where bugs and improvements related to the DSNAnalyzer could be stored in form of a ticket which provided information about the importance and a detailed description of the bug or improvement.

2.3 Use Cases

As already mentioned, Uses Cases are used for capturing the functional system requirements. In this section we show the fundamental Use Cases which we discovered in the Inception Phase (see 2.2.2.1) of the project in Table 2-2 and as a diagram in Figure 2-12.

[37] See http://checkstyle.sourceforge.net

[38] Javadoc is a tool which generates a HTML documentation of special tags in the java source code. See also http://java.sun.com/j2se/javadoc

Use Case	Short Description
Create Test	The actor creates a test for link quality measurements which is stored in a text file. This test includes parameters like the number of packets, the period, the sending power, the antenna to use and the participating Targets.
Execute Test Series	Execute several tests one after another by specifying the tests, their order and an optional delay before and after the test.
Execute Test	The actor executes a test that is specified in a text file.
Program Target	The actor loads a new binary file to a Target and flashes it.
Program Targets	Program several Targets with a new binary.
Control Target	Control a Target by setting its power state.
Execute Target Command	Execute an arbitrary command on one or a set of targets.
View Trace	The actor loads a trace of events from the DSN-Server of an external trace file. The events are displayed in different representation in several windows.
Choose Filter-Settings	Choose filter-settings according to the set of targets, log properties and time properties.
Choose Zoom-Setting	Choose the time range which should be visible without scrolling.
Export Trace	Export a trace into a file so that it can be imported later.
Analyze Test Result	Use charts to display the results of a test including the ranges of the signal strength of the correct received frames, the faulty received frames and the noise.
Analyze Link	Show the quality of a specified link.
Analyze Packet Errors	Show detailed information about the time and quantity of frame errors.
Analyze Bit Errors	Show detailed information about the time and quantity of bit errors.
Export Test Result	Export the result of a test for making in accessible in third-party software.

Table 2-2: Use Cases

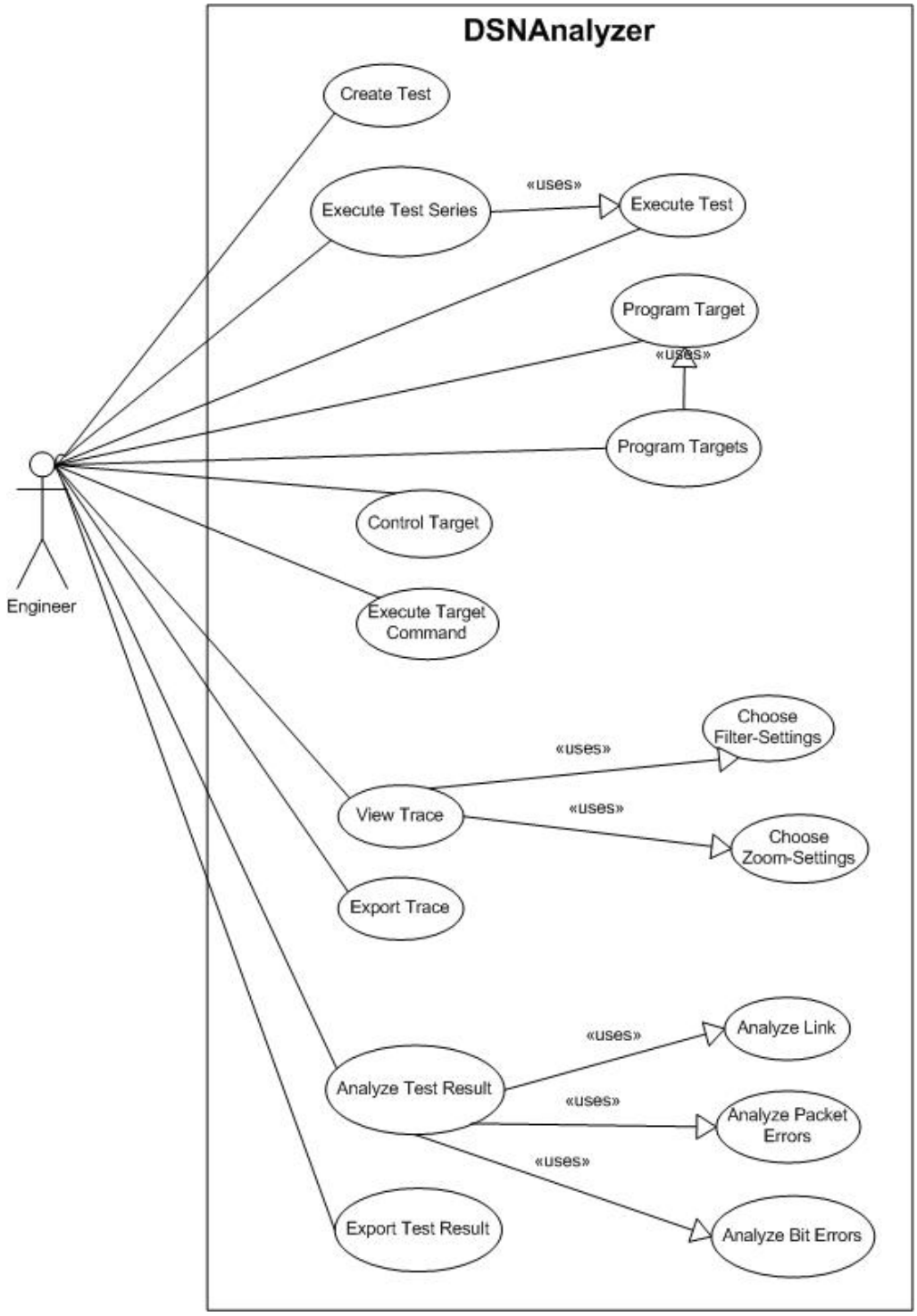

Figure 2-12: Use Case Diagram

2.4 Design and Implementation

In this section we describe the design decisions of the software implemented in this thesis. We start by describing the general design (see 2.4.1) followed by the design of the three different views: Control View (see 2.4.2), Analyze View (see 2.4.3) and Analyze Test View (see 2.4.4).

2.4.1 General GUI and System Design

In the following sections, the general design of the DSNAnalyzer and parts of the whole DSN is described.

2.4.1.1 DSN Modification

The existing DSN with DSN-Nodes, Server and Client (see [Hobi/Winterhalter 2005; Zimmermann 2005]) has to be extended in order to meet the requirements by SBT of having a prototyping system for controlling, monitoring and testing their Targets. Therefore, we need a clear separation between the DSN-Node and the Target (see 2.4.1.2).

Because the GUI application proposed by [Zimmermann 2005] is more a backend for the DSN itself than for the Targets, we decided to separate the DSN and the Target system by developing an application which is only responsible for the WSN Layer (see Figure 1-3). The tool should control the WSN Layer, analyze events from the Targets and analyze the output from a WSN simulator.

2.4.1.2 Interfaces

Because we require a client application that does not have to be at the same place as the DSN-Server, we need a clear interface at the DSN-Server which allows IP-based communication.

In order to use the DSN for various Target systems and different kinds of Client/Target interactions, we also need a clear interface between the DSN-Node and the Target.

We place the interfaces between the Client and the DSN-Server (See Interface A in Figure 2-13) as well as between the DSN-Node and the Target (see Interface B in Figure 2-13) because this is the best option for developing the system concurrently.

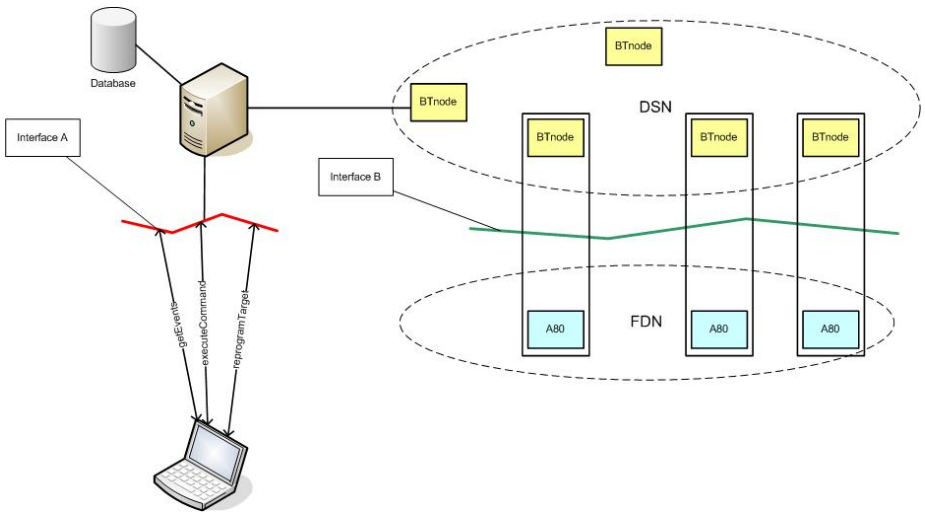

Figure 2-13: DSN Interfaces

The lower two parts of Figure 2-13 are implemented by SBT where the client in the left-lower part is the main contribution of this thesis.

2.4.1.3 Programming Language

For developing the client application, we use the object oriented programming language Java because it can be used to implement sophisticated GUI-Software and is nowadays known to a

lot of software developers. This allows an easy extension of the software. Additionally, there are a lot of Java development tools such as *Eclipse*[39], *Javadoc* and *Ant*[40] that are free, easy to use and provide the necessary functionality.

2.4.1.4 Application or Applet?

Another design decision is to choose between java applet and application. The two most important properties which have to be considered are security and distribution.

Applets are displayed in a web browser. They are easy to distribute because they are running within a web server. Users can use their web browsers for downloading and running the applet. Applets are more secure than applications because they have the limitation of being not allowed to read or write on the hard disk of the user because the user does often not know the source of the applet.

Applications run standalone on the platform of the user. For the distribution, the developer needs to ship the software to the end-user or get the user downloading the new software from a website or a network share. Applications are allowed to read and write on the hard disk of the user.

In our environment, software distribution is easy because we can simply copy the new software to a network share and we only have a limited number of users. A second important point is, that the software needs to import events from files and also has to support export functions which allow persistency of test information.

Because of the two points described above, we decided to use the application and not the applet.

2.4.1.5 Extendibility / Hardware Independence

An important design goal of the client software is to support different kinds of hardware and applications. Even though this tool is designed for prototyping fire sensor networks, it should also help to develop all other kind of sensor networks and therefore other types of hardware.

As the functions provided by the DSN-Server change during time, we put everything that depends directly on the DSN-Server into the class DSNCommunication.

Also the software on the Targets is frequently changing; therefore we only put calls to these functions into the classes which really need them (e.g. the classes in the Test package).

Whenever a method or variable fully depends on the hardware of the Target, it is marked in the code using the tag HARDWARE.

The fact that changes in the Target software affect some components of the system (e.g. Target Control) more than others (e.g. Analyze Events) is one reason for splitting the application into several mostly independent parts (see 2.4.1.6).

2.4.1.6 Views / MVC

The most important features of the system to develop are:

- Control the Targets (Commands, Reprogramming, Reset)

[39] See http://www.eclipse.org
[40] See http://ant.apache.org

- Analyze Events from Targets and Simulations
- Analyze Test Results

The DSNAnalyzer is structured into the three parts described above by using views that can be selected with tabs. Views also allow easy extendibility by just adding a new view when new features are requested. The separation of the application into views can also be seen in the structure of the packages (see 2.4.1.7).

By defining the classes of the application, we considered the Model-View-Controller (MVC) Pattern. The MVC Pattern is used to divide classes that model the problem domain (Model) from classes that control the model (Controller) and classes that display the model (View). The MVC Pattern also reflects the Observer Pattern where the Model notifies its observers (the Views) about its changes [Deacon 2005, 2 ff.].

We try to apply the MVC pattern as strong as possible when dividing models and their views and controllers, because keeping the models clean allows easy reusability of classes that work on the same problem but use a different user interface.

2.4.1.7 Packages

The system is structured into packages which, whenever possible, fulfill the package design guidelines proposed in 2.2.3.2. The package design is shown in Figure 2-14 where we can see the three main partitions controlview, analyzeview and analyzetestview. Additionally, there are common classes such as Target which are used by all GUI classes. The classes in the test package are used for channel measurements and util classes are used by all other packages.

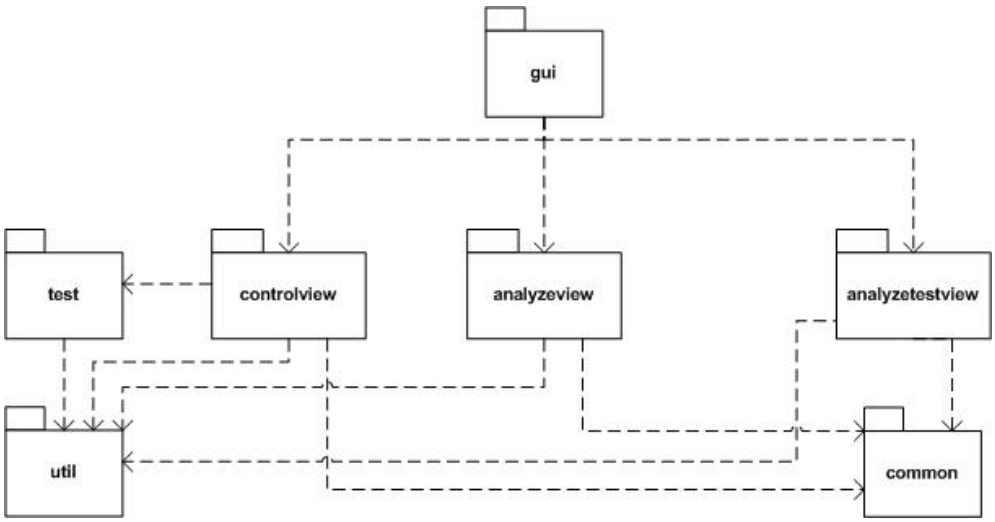

Figure 2-14: UML Package Diagram

2.4.1.8 ID Mapping

When accessing nodes, we need IDs to identify them. The DSN-Nodes have a 6 byte ID which is used by the DSN-Server to communicate with the DSN-Nodes. The A80 Targets have a 4 byte ID which is used by the client.

Because the DSNAnalyzer should communicate with the Targets over the DSN-Nodes, we have to find a mapping from the Target-IDs to the DSN-Node-IDs and vice versa. Additionally, we have the problem that we need the whole 4 bytes of the Target-ID because

the IDs of the Targets are random and not consecutive which would e.g. enable us to use only the lowest byte[41].

For solving this issue we have the following possibilities:

1. Change the ID of the Target to the ID of the DSN-Node which is connected to it and access the Target with the ID of the DSN-Node.
2. Use a mapping function on the DSNAnalyzer that maps DSN-Node-IDs to Target-IDs.
3. Proposal 2 with an additional mapping to a virtual ID which makes the ID more user-friendly.

Proposal 1 is not very flexible because we cannot easily recombine DSN-Nodes with Targets but would have to flash the Target at every change with the new DSN-Node-ID. Because of that and because proposal 3 would lead to a complex mapping process we took proposal 2 as it can be implemented using a special network-file which maps the IDs and can easily be edited.

2.4.1.9 Property File

For storing long-term properties of the DSNAnalyzer we use a property file. In this file we specify the following properties:

- The location of the network-file (see 2.4.1.8).
- Filter properties for log level, log layer, log aspect, time range and targets.
- A property which specifies if events are colored according to their log level or to their log aspect.
- The address of the DSN-Server.
- A property for enabling and disabling the 3D Map in the Analyze View.
- The default path.

2.4.2 Control View

The Control View (see Figure 2-15) should provide all functionality which is needed to control the Targets. This includes execution of commands, reprogramming, showing active Targets and performing tests.

[41] We cannot abbreviate “804404ab” with “ab” because there could be a second target with the ID “806404ab”.

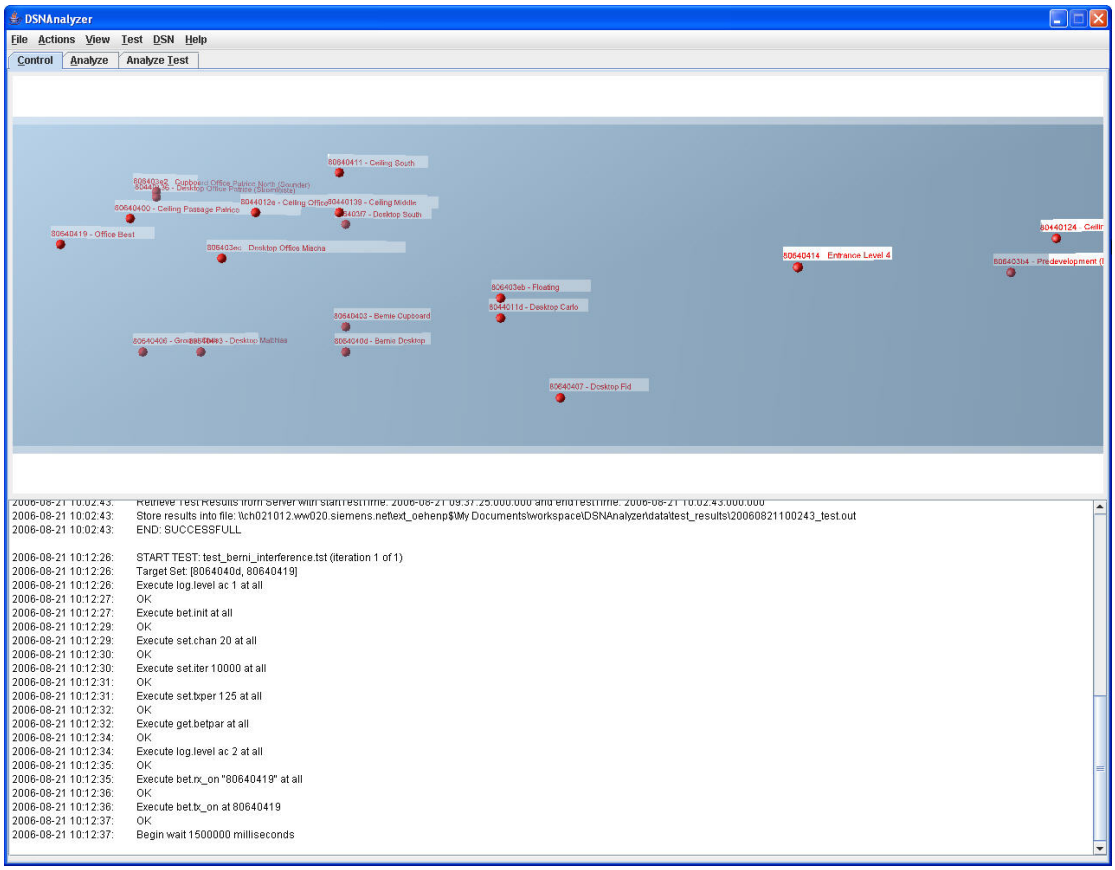

Figure 2-15: Control View

The structure of the Control View is shown in Figure 2-16.

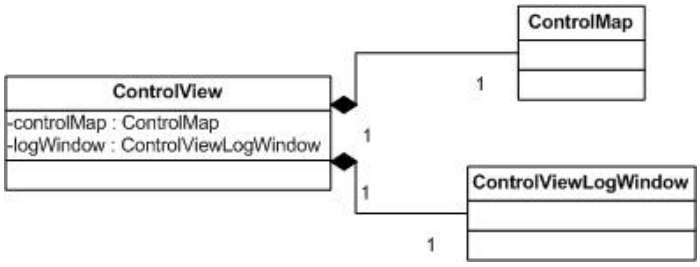

Figure 2-16: Control View Class Diagram

2.4.2.1 Command Execution

One of the basic features which are needed to prototype sensor nodes is to communicate with them directly. For this purpose, we use a terminal which allows the user to send commands to the Targets and retrieve replies. The control flow of a command/reply interaction can be seen in Figure 2-17. The reply of the Target is an event in the JSON-RPC format (see 2.1.3.2) which includes the same ID as the command. At the DSN-Node, this event is referred to as log entry. In the case shown in Figure 2-17, the DSN-Node is in the push mode whereas it automatically sends all log entries to the GUI-Node.

The communication with the DSN-Server is non-blocking. Because a reply is only an event (see 2.1.3.4), we use the same method to get the reply as if a set of events is requested (see

2.4.3.1). The DSNAnalyzer requests the reply-event of the command by requesting an event which includes the ID from the command. This request is done every 200 ms and for at most 10 s.

The format of the commands and replies is the JSON-RPC format (see 2.1.3.2).

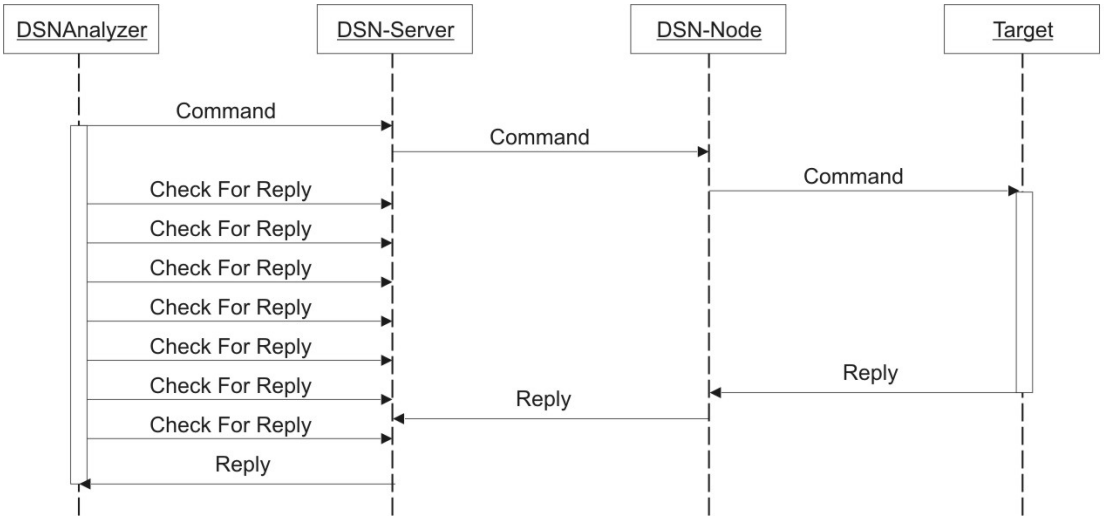

Figure 2-17: Command Sequence Diagram

2.4.2.2 Map

A requirement of the client software is to display the Targets which are currently running by showing them using icons in a map. For the map, there are two possibilities:

- A common 2D map which shows a Targets using an icon at a (X, Y)-position. This kind of map has the advantage of being very simple and supported by many open source libraries. The drawback is the lack of 3D drawing which would be helpful for Targets which are on different floors or on different height in a floor.
- A 3D map which shows the Targets using a (X, Y, Z)-position. The advantage of a 3D map is to be able to draw Targets on different floors or different height in a floor. The drawback is that for this kind of map, there are not many open source libraries.

For fire detection networks, it is very common that Targets are distributed on different floors. Therefore we decided to use a 3D map. For showing this map in the Control View, we use the *Wilmascope 3D graph visualization system*[42] since no working open source alternatives could be found which met our requirements. The map we use additionally provides easy view customization features like Zoom, Move and Rotate which enable the user to look at every possible perspective onto the set of Targets which is helpful if there are a lot of Targets distributed over several floors.

The map shows each Target which is connected to a running DSN-Node using the position information from a network file. This network file has the default ending .in and the format shown in Table 4-3.

[42] See http://wilma.sourceforge.net

Actions belonging to only one Target can be executed by choosing an option in the pull down menu when right-clicking on a Target icon. Actions belonging to all Targets can be chosen in the menu bar.

2.4.2.3 Target Programming

During prototyping processes, Target software changes occur very often. Therefore, we need a way to perform code distribution and Target flashing very fast and flexible. The DSNAnalyzer uses the programming functions of the DSN which work in the following manner:

1. The user chooses if one or multiple Targets should be programmed.
2. The user chooses the software (e.g. a hex file).
3. The selected binary is translated into a String and sent to the DSN-Server.
4. The DSN-Server translates the String into a binary and sends it to the GUI-Node.
5. The GUI-Node begins to distribute the binary to all DSN-Nodes.
6. The DSNAnalyzer repeatedly checks the current software versions of the Targets which should be reprogrammed.
7. When all the necessary Targets have the new binary, the DSNAnalyzer sends a command to flash them.
8. The DSN-Node returns an reply if the flashing was successful or not.
9. The DSN-Server redirects this reply to the DSNAnalyzer.

We think that the points 6 to 9 should be handled by the DSN-Server and added it therefore to the further work in 2.5.3.3.

2.4.2.4 Tests

Tests are a way of externally stimulating the Target software. There are two different types of tests:

- A set of commands using the test software running on the Targets which is used to do measurements in the WSN (see Figure 2-18). This kind of test, which is referred to as *Link Quality Test*, we use in Chapter 3. Link Quality Tests have always a software-counterpart which actually performs the measurements on the Targets.
- A set of commands which stimulate the default behavior of the Target software (see Figure 2-19). An example for this kind of test is a simulated fire alarm where we are interested in the reaction of the system.

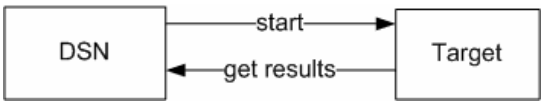

Figure 2-18: Link Quality Test

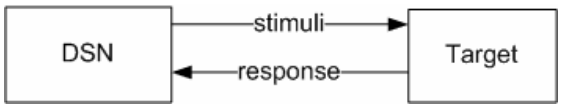

Figure 2-19: Stimuli Test

More formally, tests are a set of commands in the format described in Table 4-4 in the appendix.

A test in the DSNAnalyzer is represented by the class Test which consists of the properties of the test as well as the test elements (see Figure 2-20). A TestElement corresponds to one command for either one specific target or all targets. It optionally includes a time at which the command should be executed. Special types of test elements are wait commands which tell the test engine to wait for the specified amount of time or trace file commands which tell the system where to store the results of the test.

Tests that are executed in the DSNAnalyzer are parsed from a text file. Such test files can either be created by using a text editor or by using the class TestFactory. This class allows the user of the DSNAnalyzer to specify all parameters of a test such as participating Targets, number of frames to be sent, time between two frames, sending power and all other parameters which are supported by the test software running on the Targets. The TestFactory (see Figure 2-20) is used by the ChooseTestParametersDialog which serves as a graphical representation for the user to choose the test parameters.

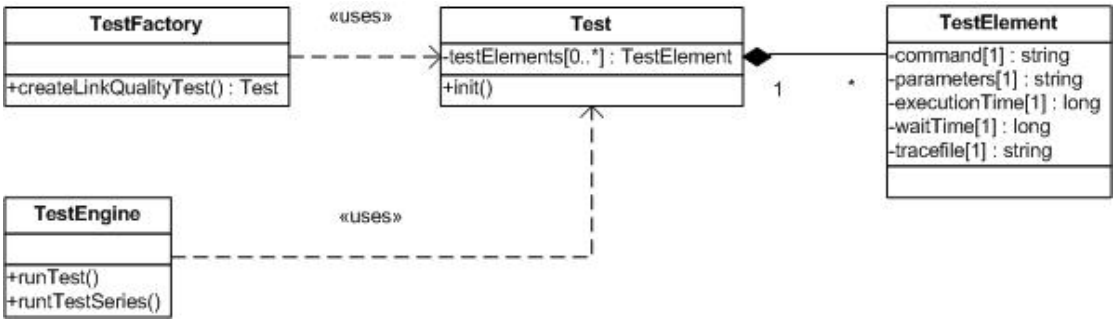

Figure 2-20: Test UML Diagram

Link quality tests consist of several subtests where in each subtest one Target is sending test-frames and all others are listening and recording errors. Up to now, there exist two test modes:

- *Rotation Mode* (In this mode, every participating Target is the sender only, i.e. a test containing n Targets consists of n subtests.)
- *One Sender Mode* (In this mode, tests consist of only one subtest where a set of listening Targets along with a sender is specified.)

Tests are executed by the class TestEngine (see Figure 2-20) which parses test files and executes specified tests one after another. The execution of a test consists of executing one test element after another. During execution, the TestEngine displays debug, warning and error output about the current state of the test.

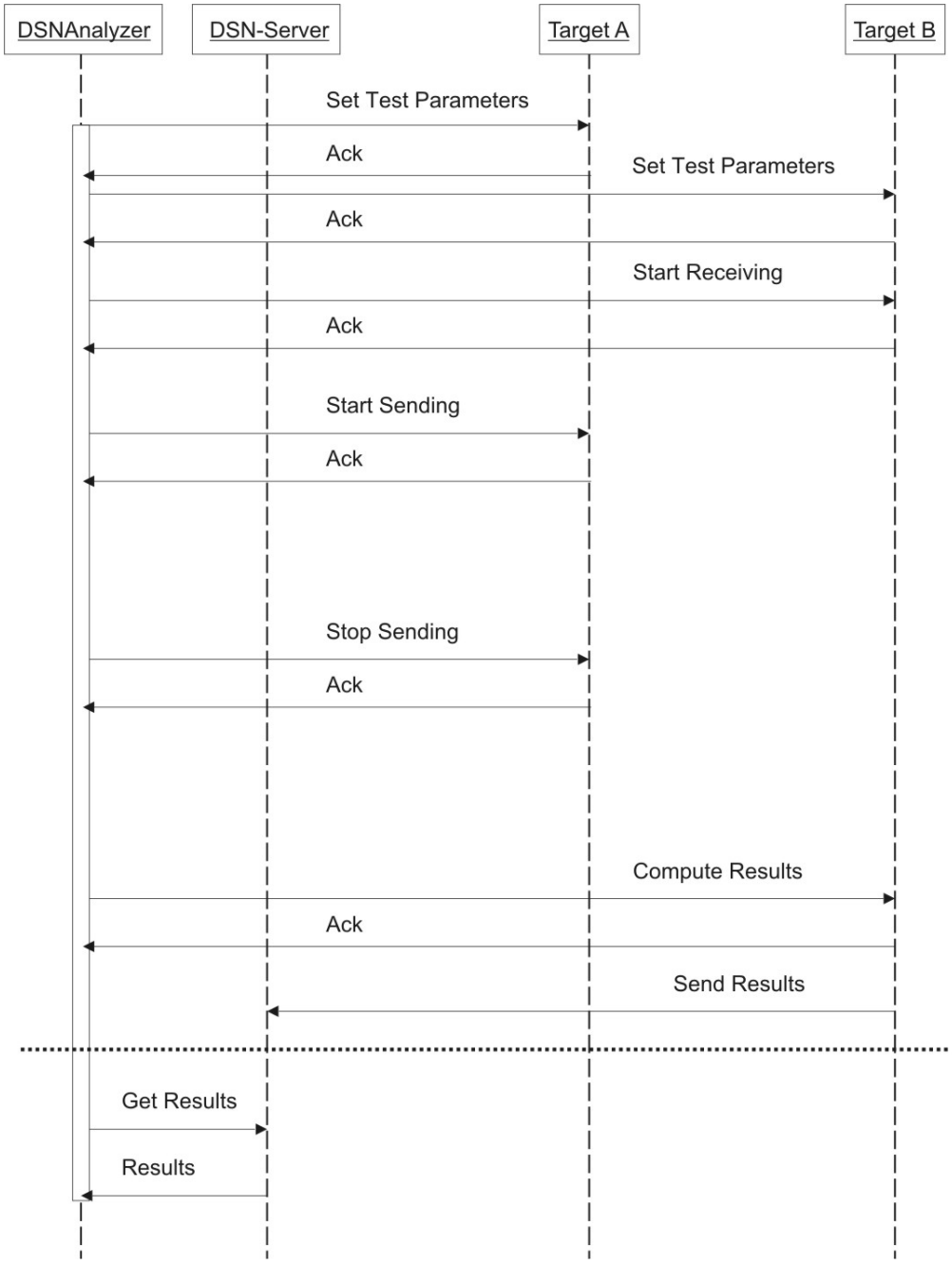

Figure 2-21: Test Sequence Chart (The DSN-Nodes are left out for a better presentation)

A part of a simple link quality test can be seen in Figure 2-21 with two Targets A and B. The corresponding test file in shown in Figure 2-22. In this test, both Targets receive the parameters (e.g. channel) of the test. Target B begins to receive and Targets A begins to send. After the frames are sent, Target A stops and Target B computes the results which are sent to the DSN-Server. When the DSNAnalyzer has performed all subtests of the tests, it requests the results.

```
# A80 - Link Quality Test
# Creation Date: 2006-09-18 16:17:52

# Target List
%targets=80640400:80440124

# Set test configuration
all bet.init
all set.chan 20
all set.power 21
all set.iter 50000
all set.ant 0
all set.txper 120
all set.preamb 3
all set.confset 1
all set.rssi_thr 25
all get.betpar

# Subtest with target 80440124 as sender
all bet.rx_on "80440124"
80440124 bet.tx_on
%wait=7200000
all get.betres

# Stop all targets
all bet.stop

# Store results into this trace file
%tracefile=test
```

Figure 2-22: Example Test File

The reason why methods such as “Start Sending” are acknowledged immediately and not after they have completely finished, is the following: If “Start Sending” would be acknowledged only after the whole number of frames are sent and the Target either does not receive the command correctly or fails during execution, the test would hang up. This is one reason for our general rule, that commands should be acknowledged immediately. This rule belongs to the overall principle that the whole test-system should be fault-tolerant and therefore be able to continue a test with all the remaining running components. This principle is realized in that commands, for which no acknowledgements are received, are executed again.

The results from the tests are stored by the DSN-Server into the database. Results have the form of Events (see 2.4.3.1) and can be retrieved in the Analyze View (see 2.4.3). The test results, which are either retrieved in the Analyze View or stored using the result file test command (see Table 4-4), can be analyzed using the Analyze Test View (see 2.4.4).

In order to allow parsing of test results we need a clearly defined format. This format is defined in Table 4-6.

In order to allow the user to do a sophisticated scheduling of tests, we introduce the Test Suite. This feature can be used to run a set of tests one after another where for each test, a

delay before, a delay after and the number of iterations can be specified. The delays can also be used to run a test at given time in the future.

2.4.3 Analyze View

The goals of the Analyze View (see Figure 2-23) are to provide the functionality of analyzing the events that occurred at the Targets in both the simulation and the real world. In the following sections we show how events are managed, why real-time monitoring was not implemented, how events can be filtered, how they are displayed in a sequence chart, a map and a log window and how they can be exported.

The Analyze View also works in a standalone mode, i.e. when no DSN-Server is available. In this mode, the user can import trace files which consist of events.

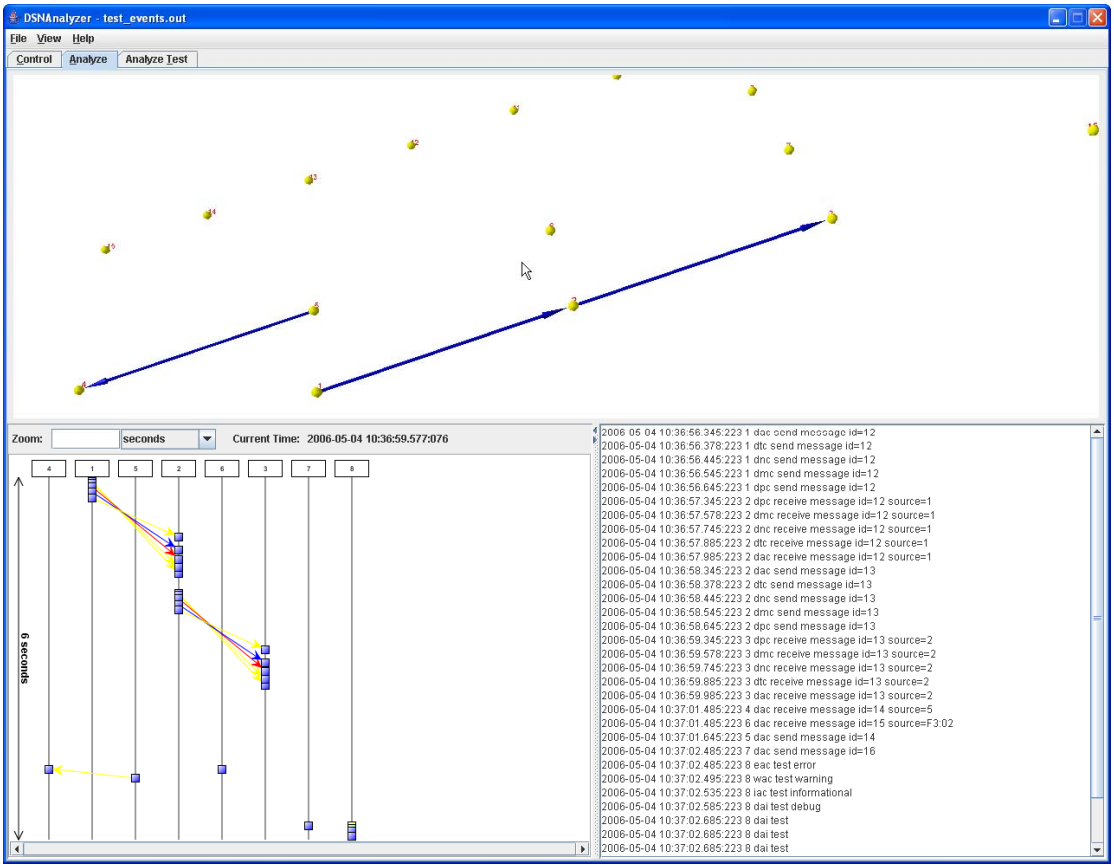

Figure 2-23: Analyze View

The structure of the Analyze View is shown in Figure 2-24.

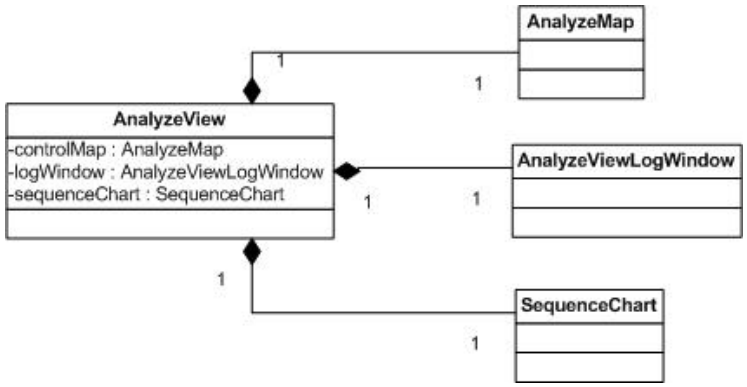

Figure 2-24: Analyze View Class Diagram

2.4.3.1 Event and EventList

In order to analyze the behavior of the Targets, we need a mechanism to see what is happening on them. The solution is to create events whenever something important happens.

An event is a textual representation of a happening at a given Target at a given time. It describes what happened and additionally gives information about the severity (log level), the place of the occurrence in the software (log layer) and what it is related to (log aspect). Because the DSNAnalyzer should handle both outputs from Targets and from the simulator, the simulator also creates events according to the described format (see Table 4-5).

A special kind of an event is the communication event. It contains information which can later be used by the analysis tool to fetch the sender and receiver of the message (see Table 4-5 in the appendix).

The data structure for the event in the DSNAnalyzer is the EventList which contains an ordered list of events between a time A and a time B (see Figure 2-25).

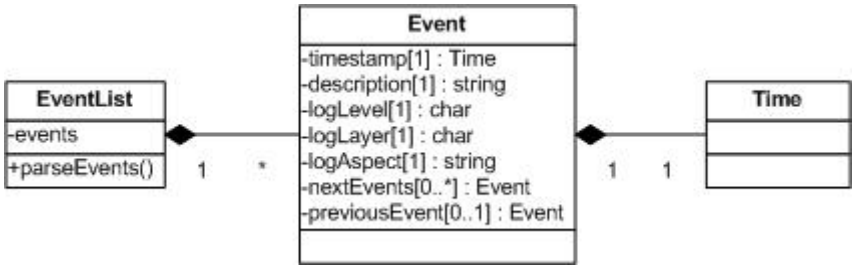

Figure 2-25: Event List UML Diagram

For archiving, an EventList can be exported into a trace file which consists of events. The detailed format of a trace file and its events is shown in Table 4-5.

2.4.3.2 Real-Time Monitoring

A first goal of a user of an analyzer-system for WSN is to be able to real-time monitor what happens in the network. The user would like to have a map where all Targets are represented by icons and events happening on them are displayed exactly when they happen.

Although this would be a nice feature for the user, we have decided not to implement that because of the following reasons:

- The logging mechanism of the DSN works with a pull mode, i.e. events[43] are requested by the DSN-Server and not directly sent to the GUI-Node when they appear[44]. This mode was required because the GUI-Node is a bottleneck of the DSN and cannot process unlimited logs, which would appear in the case that every Target sends all its logs directly. Because of this pull mode, the events will be kept at the DSN-Node until requested what makes it impossible to show events in real-time[45].
- Even if the events are sent using a push mode exactly when they are created, we would have the problem that an event created at Target A with a distance of 2 hops to the GUI-Node at time t could be at the DSN-Server before an event created at Target B with a distance of 10 hops at time t-1.
- If events would not arrive in correct time order (because log entries go over a different number of hops), it is difficult to draw them into a sequence chart (see 2.4.3.4) because all graphical events would have to be rearranged in case an event with a timestamp between them would arrive, which makes the drawing process complex and therefore slow.

Although events cannot be shown in real-time, they can be requested at the DSN-Server after a short time[46] and displayed e.g. in the Sequence Chart (see 2.4.3.4).

2.4.3.3 Filter

Because the user is sometimes interested in only a set of events, the concept of a filter is needed. The filter should allow the user to see exactly the events which are of interest and should be easily configurable.

The filter has the following properties:

- Log
    - Log level (Error, Warning, Informational, Debug)
    - Log layer (System, Physical, MAC, Network, Transport, Application)
    - Log aspect (User customized)
- Time range
- Target list

Using the log filter, the user can define exactly which events should be visible (e.g. the triple “enc” filters all events which have log level Error, happened in the network layer and belong to a communication process)

[43] In the DSN, events are referred to as logs or log entries.

[44] The opposite of the “Pull-Mode” is the “Push-Mode” in which the log entries are immediately sent back to the GUI-Node.

[45] We think that the performance of a pull or a push mode very depends on the implementation of the DSN-Node software. We also think that this question has to be treated separately.

[46] This time depends on the DSN-Server configuration and the number of active Targets but should be in range of one or several minutes.

In a first design we allowed the log aspect to be only one of a list of possible values. After having discussed this property with alpha users of the system, we have decided to allow user-define values. This makes it possible for users to use a special string for the log aspect for certain events for filtering them later.

Because some user may always be interested in the same information, we have decided to put the filter configuration into a property file (see 2.4.1.9 and the DSNAnalyzer Readme in the Appendix) where they can easily be changed.

2.4.3.4 Sequence Chart

A Use Case for the DSNAnalyzer is to show events using graphical support (see Table 2-2). A way to display events is using a sequence chart which shows events on timelines (we have one timeline per Target).

The sequence chart should be able to draw events on the lifeline of the Target on which the event happened. Additionally, it should display arrows between Targets which exchange messages. A message should be displayed by an arrow from the sender to the receiver. In order to solve that, we created the concept of a communication event, i.e. events that happen not only at one Target but are a combination of a send and a receive event. Here we have the problem to match a send event with its corresponding receive events which is difficult because of the reason that we cannot just insert the ID of the sender into a message because the message could be corrupted.

We have found that the only method to correctly draw a communication event in the sequence chart is to include the message ID at the send event and the message ID as well as the sender ID at the receive event[47]. Therefore, we have to use error control mechanisms of the MAC layer to be sure that the message was correctly received at the sender. Only if we are sure about the correctness of the message, we can create a proper receive event. It is obvious that a lost or corrupted message can be identified in the sequence chart when only a send event and no corresponding receive event is visible.

In a first approach, we only showed arrows between the highest log-layer available, i.e. if we have network layer enabled, we only show communication events from the level network. However, because the source and the target of a message from different layers could be different (e.g. between network layer and a MAC layer), we have chosen to draw all communication events independently of their layer. In order to avoid that the whole screen is full of arrows, the user can filter layers which produce a lot of messages.

Another requirement is to combine all windows of the Analyze View, i.e. the Sequence Chart, the Map (see 2.4.3.5) and the Log Window (see 2.4.3.6). The goal is that the user can see when and where the events happened and how they are related to each other.

As a result of the decisions above, we have designed a sequence chart that displays events that match the filter settings (see 2.4.3.3) according to their timestamp (y-axis) and the Target on which they happened (x-axis). The Sequence Chart has a zoom property which is set to the time which should be visible without scrolling. The Sequence Chart can be controlled by the scrollbar and by setting the time-range which should visible without zooming. An example of

[47] Unfortunately, this is against our rule of not including relevant information into messages but up to now, it is the only solution that works.

a sequence chart that shows a protocol which is currently in development can be seen in Figure 2-26.

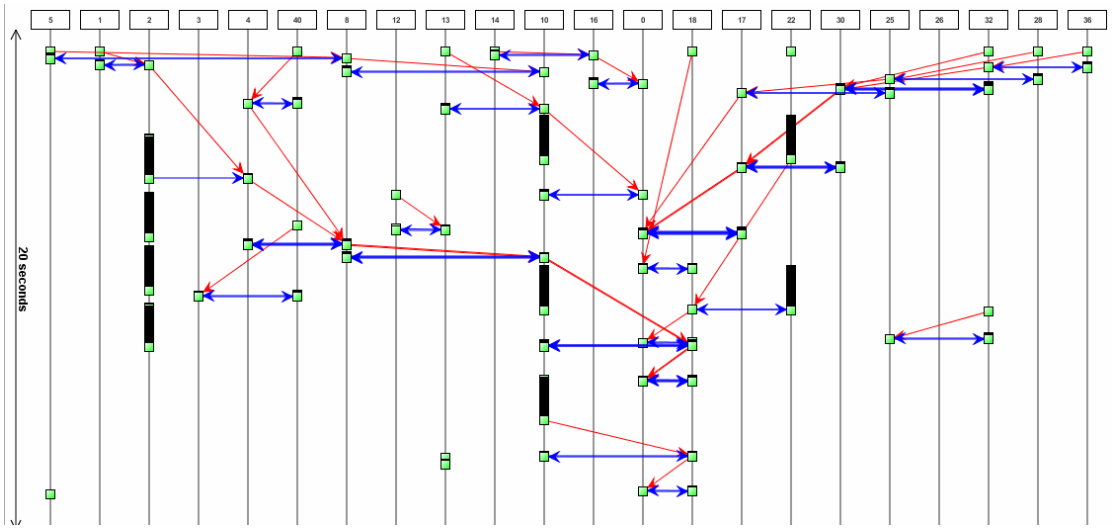

Figure 2-26: Example Sequence Chart

The control concept of the Analyze View is the following: The Sequence Chart is the controller of the map (see 2.4.3.5) and the log window (see 2.4.3.6). In the map and the log window we can see all the events which are visible in the current section of the sequence chart, i.e. what is visible in the sequence chart without scrolling is also visible in the other windows[48]. When the user slowly moves down the scrollbar, event order and place become visible in the map[49]. If an event of the head of the timeline is selected in the Sequence Chart, the corresponding Target is marked in the Map.

For the coloring of the events, we used the log level information. Error events are shown red, warning events yellow, informational events green and debug events blue. As a help for the user, the time in the middle of the sequence chart is also displayed.

As a result of evaluation and input from alpha testers, we added the following features to the Sequence Chart:

- If an event is selected and the zoom or filter settings are changed, the sequence chart centers on the selected event (in case it is not filtered).
- A feature for reloading imported trace files which allows the users of simulators to display new created output without explicitly reloading the file.
- A feature for showing all events without scrolling.
- Possibility to quickly hide Targets lifelines without opening the filter dialog.

[48] If a communication event has a counterpart that is not visible in the sequence chart anymore, it is shown even though in the two other windows.

[49] A first idea of including a play function which changes the time automatically was dismissed because the user can perform that using the scrollbar and the mouse.

2.4.3.5 Map

The user of this client is not only interested in the Target on which an event was created, there should also be information about where this Target is located. If a message is sent from one Target to another, it should be visible to the user over which path and distance this message was sent. The map should give the user a feeling of how large the WSN is and in which area traffic is. This enables the user quickly find protocol errors.

For the map, we took the same software base as used in the map of the control view (see 2.4.3.5). Additionally, communication events happening on the Targets are shown by drawing arrows. In the map we see all communication events which are currently visible in the sequence chart without scrolling.

After discussions with alpha testers, we decided to add an option to enable or disable the map in runtime as well as with a property in the application property file. Disabling the map makes the application faster and displays a larger Sequence Chart to the user.

2.4.3.6 Log Window

The user of the system should be able to see the details of an event. Because the Sequence Chart and the Map are used to get a picture of where and why the event was created, we need a third view which displays the event details.

The Log Window of the Analyze View shows all events which are currently visible in the sequence chart without scrolling. For showing the context of a selected event, it can be selected to highlight it in the other two windows.

2.4.3.7 Data Export

A requirement of the application is to make the traces persistent. We need a way to export data which the application gathered from the Targets in a format which is easy to import.

The Analyze View provides the following export mechanisms:

- Export all textual events from the trace
- Export only the textual events which are currently displayed in the Log Window
- Export the whole Sequence Chart as a graphic

An additional export feature is included in the Analyze Test View (2.4.4) and is used for exporting the results from a measurement.

2.4.4 Analyze Test View

Important Uses Cases described in (2.3) are analyzing a link and analyze packet as well as bit errors. There should be a way to make the results of different link quality measurements visible using charts which show the quality of the link and information about the link strength. Additionally, there should be a possibility to analyze the frequency distribution of bit errors and the link strengths when they happened.

In this section we show how the Analyze Test View (see Figure 2-27) was designed by defining the data structure of a test result (2.4.4.1), introducing charts and chart containers (2.4.4.2) and further features.

Figure 2-27: Analyze Test View

The structure of the Analyze Test View is shown in Figure 2-28.

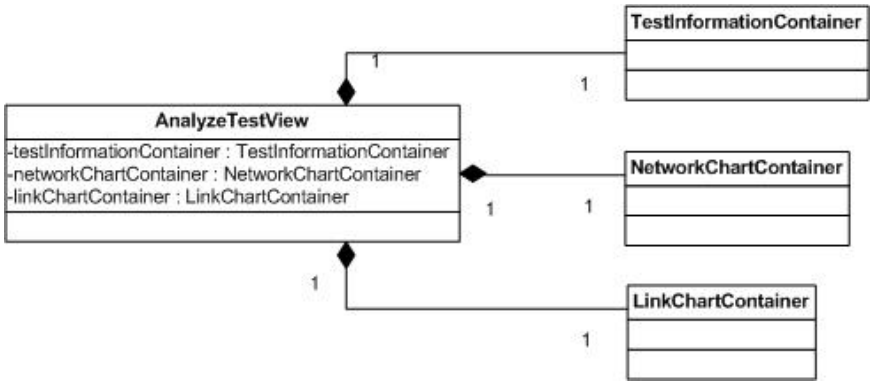

Figure 2-28: Analyze Test View Class Diagram

2.4.4.1 Test Result

A test executed in the rotation mode (see 2.4.2.4) contains several subtests, one for each participating Target. After a subtest, each receiver generates its subtest result. During the sending phase, each Target which receives a frame creates a frame result[50]. Both the subtest result and the frame result are events which are sent to the DSN-Server. As described in 2.4.2.4, test results can be stored into a text file.

[50] This depends on the log setting of the Target.

While parsing the test result text file, the application creates the data structure shown in Figure 2-29 for storing the results.

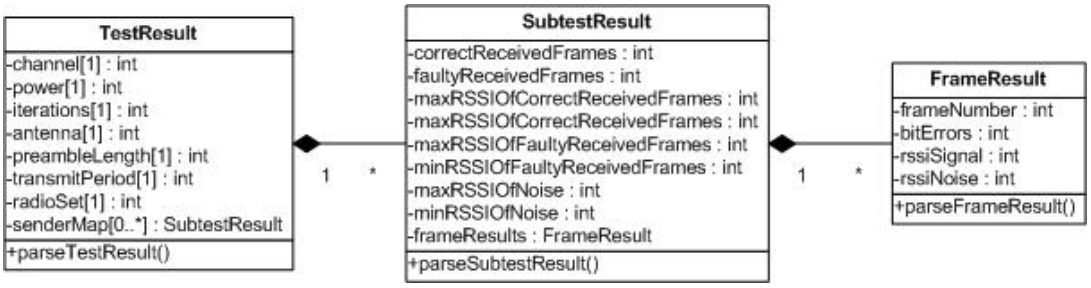

Figure 2-29: Test Result UML Diagram

2.4.4.2 Charts

For analyzing the results of the measurements, we need to draw charts which show information about the test results. After analyzing and discussing the main questions which we want to answer using this tool, we decided to use the following charts:

- Stacked Bar Chart with correct received frames, faulty received frames and missed frames for all receivers.
- Chart which shows the range of RSSI values of the correct received frames, the faulty received frames and the noise for all receivers.
- X/Y-Chart which shows the RSSI for each frame for one receiver.
- RSSI Histogram (How many correct frames and faulty frames we had at which RSSI).
- Chart which shows how much bit errors we had at which RSSI.

For the creation of charts, we evaluated the following three candidates:

- Microsoft Excel[51]
- Mathworks Matlab[52]
- JFree Chart[53]

The first two candidates would need an export function in the DSNAnalyzer which exports the test results in a common format.

Microsoft Excel is easy available at most office computers but has the drawback of providing only basic chart drawing functionality which cannot be easily extended.

Mathworks Matlab needs an expensive license but provides rich functionality in chart drawing and has a good performance.

JFree Chart bases on java and is therefore not very fast but supports a lot of drawing possibilities, is open source and easy to extend. Additionally, it is the only tool that can be directly integrated into the DSNAnalyzer.

[51] See http://office.microsoft.com
[52] See http://www.mathworks.com/products
[53] See http://www.jfree.org/jfreechart

In the DSNAnalyzer, we decided to use JFree Chart because of its advantages. We use it to draw the charts described above in the Analyze Test View. Additionally, we decided to support Matlab because of its rich functionality and easy usability by providing an export function for the test results. For non-automated chart generation we also use Microsoft Excel.

The general idea of using this view is to provide an overview about the most important results with the charts directly integrated. If special results are found, they can be exported to Matlab. Matlab provides an easy programming language which enables the user to write very short code for analyzing data in a special way which is not needed for all kind of results.

A design goal of the Analyze Test View is easy extendibility because the charts and the measurement results change for each application or field of interest. Therefore, we developed a framework which is based on containers in which the charts are represented.

When the developer of the DSNAnalyzer wants to extend the Analyze Test View with new charts, only two methods have to be added and little has to be changed.

The first container of this view is the Network container which contains all charts which are about a set of links. In the second container, the Link Container, we draw all charts which are only about one link and allow representing detailed data.

2.5 Evaluation

In this section, we evaluate the application described above as well as the whole DSN. Because integration of the existing DSN is an important part of this thesis, we primarily concentrate on the evaluation on the DSN itself (see 2.5.1). Although we did not work on this software, we have to know how it performs and how stable it is because our software acts as a client to it.

For the DSNAnalyzer we additionally performed a survey (see 2.5.2).

2.5.1 Measurements

In this section, we perform several measurements in order to determine values for import durations, power consumption, stability, latency, fault-tolerance, performance and range of the DSN.

2.5.1.1 Trace File Load Duration

In this measurement we evaluate how long it takes to load a large trace file in the Analyze View. We take a trace file with over 17000 events[54] and a file size of 1 Mbyte. The setup consists of three Siemens computers with different hardware. On every computer we perform three iterations of the measurements and compute the average times which are shown in Table 2-3.

[54] 9300 of them are communication-events.

Computer	Load Duration
Intel Pentium 4, 2.8 GHz, 504 MB RAM	16.7 s
Intel Pentium 4, 2.6 GHz, 504 MB RAM	18.6 s
AMD Mobile Athlon, 1.8 GHz, 448 MB RAM	28.6 s

Table 2-3: Trace File Load Duration

The times shown in Table 2-3 are the durations for parsing the trace file (generating events), creating a graphical element for every matching event and drawing the messages in the map.

In a second measurement we turn off the 3D Map of the Analyze View in order to measure its influence in the loading time. The results are shown in Table 2-4. We can see that drawing messages in the 3D Map takes between 21 % and 27 % of the whole loading duration. The time to draw messages in the 3D Map is very dependent on the type of events in the trace file. If there are a lot of communication events, it takes longer.

Computer	Load Duration
Intel Pentium 4, 2.8 GHz, 504 MB RAM	12.2 s
Intel Pentium 4, 2.6 GHz, 504 MB RAM	13.9 s
AMD Mobile Athlon, 1.8 GHz, 448 MB RAM	22.6 s

Table 2-4: Trace File Load Duration (Without Map)

2.5.1.2 Power Consumption / Battery Lifetime

In this section we determine power consumption properties of the “Blue Box”. We therefore measure the average power consumption of a “Blue Box” with all meaningful configuration modes.

Setup	Adapter Board	A80	BTnode	Average Current Consumption
I2C-Mode	X	-	-	1.3 mA
Serial-Mode	X	-	-	1.3 mA
Spy-Mode	X	-	-	7.3 mA
Pass-Through-Mode	X	-	-	5.9 mA
Spy-Mode	X	X		7.5 mA
Spy-Mode	X	X	X	47.8 mA
Spy-Mode (Sending A)	X	X	X	50.0 mA
Spy-Mode (Sending A, Logging)	X	X	X	50.2 mA
Spy-Mode (Sending B)	X	X	X	57.0 mA
Spy-Mode (Sending B, Logging)	X	X	X	57.6 mA
Spy-Mode (Sending C)	X	X	X	76.6 mA

Table 2-5: Average Current Consumption (X: present, -: not present)

In the test mode “Adapter Board Spy-Mode, A80 (Sending A, Logging), BTnode” (see Table 2-5), the A80 is sending test-frames with a period of 1000 ms what results in the creation of an event every 1000 ms. This event is directly transmitted by the DSN-Node. In “Sending B” the period is 300 ms and in “Sending C” it is 100 ms. We also measured the maximum peak which is 83.0 mA. What seems strange is that we have a very small difference in current consumption between the logging-mode and the non-logging-mode of the A80.

In a second measurement we determine the lifetime of the “Blue Box” when powering it with batteries. For this measurement we used 4 NiMH AA 1.2V “GP 2300” rechargeable batteries with 2250 mAh. In order to be sure that we have the same power at all boxes, we recharged all batteries again at the same day.

Our setup consists of 10 “Blue Boxes”. In the first measurement we place an A80 Target and a BTnode on the Adapter Board. The A80 does not communicate and does not create any events. The BTnodes run the JAWS software, i.e. the Bluetooth stack is running, therefore there is a continuously Bluetooth communication between the 10 BTnodes. The Adapter Board is running in the spy mode. When taking 47.8 mA (the average power consumption from Table 2-5 for this mode) we would expect a lifetime of 2250mAh / 47.8 mA = 47.1 h.

The result of this measurement is shown in Figure 2-30. We can see that we have a mean value of 44.45 h which is near to the expected value. It is obvious that we measured less than expected because of the fact that current is dropping over the time and because the devices turn of before all the energy is used. We also see the standard deviation of 4 h. However, we also see one “Blue Box” which performed very bad and stopped running after only 39 h.

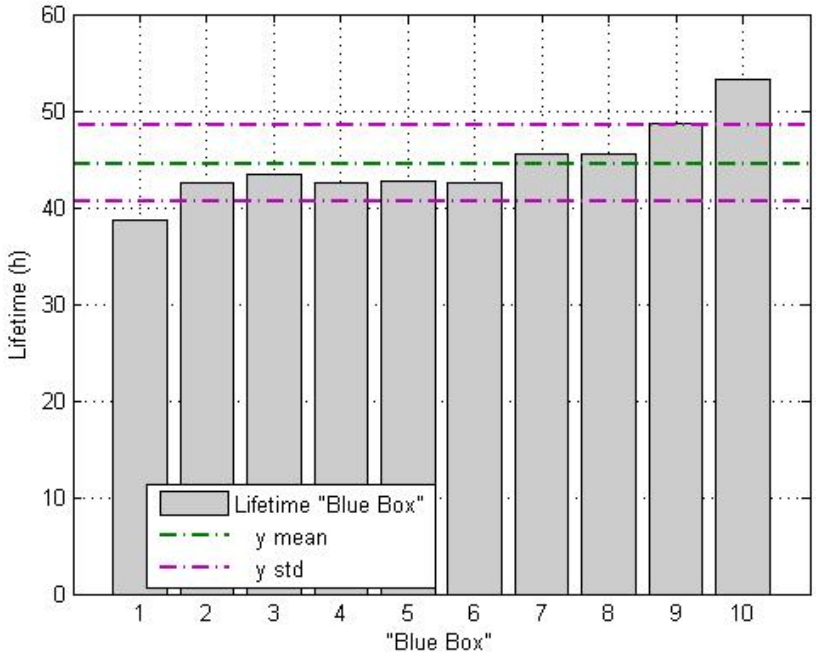

Figure 2-30: "Blue Box" Battery Lifetime

The result of this measurement can be compared to a measurement performed in [Hobi/Winterhalter 2005, 59] where BTnodes were powered using 2 NiMH AA researchable batteries. The result of the measurement is a lifetime of 27 hours. This result is in the same range as ours if we regard the fact that we use 4 batteries but also power the Adapter Board and the A80.

2.5.1.3 Binary Distribution

The DSN supports a mechanism for distributing a binary file to all DSN-Nodes which can later be used to flash the Target or the DSN-Node itself. In this section we measure the time which is needed until all DSN-Nodes have the newly distributed binary file. We start measuring after the user enters the command to distribute the binary. This process includes converting the binary into a string format, loading it using XML-RPC to the DSN-Server, converting the string-binary to a hex-binary, loading the binary to the GUI-Node and finally distributing it in the DSN. We perform this measurement on a small DSN with 8 DSN-Nodes and on a large DSN with 26 DSN-Nodes. We use a binary file with a file size of 70 Kbyte.

On the small setup it takes 25 s until the file is loaded to the GUI-Node. After that it takes another 30 s until all 8 DSN-Node have the new binary.

On the large setup it takes 46 s until the file is loaded to the GUI-Node[55]. The additional duration for the distribution of the binary is shown in Table 2-6.

[55] The time-difference is a result of several retries of the DSN-Server to load the binary to the GUI-Node.

Number of DSN-Node with new binary	Duration
5	30 s
11	60 s
24	90 s
26	120 s

Table 2-6: Binary Distribution Duration 26 DSN-Nodes

2.5.1.4 Target Flashing

In this section we show the results of a flashing measurement. In a first part we show how dependable the flashing mechanism of the DSN-Node is, i.e. how often it fails. Please note that flashing does not include distribution of the binary in the DSN but is only the process of loading the binary from the DSN-Node to the Target and resetting it.

We used 7 Targets that were 10 times flashed using a baud rate of 38400. 5 Targets flashed 10 times successfully, 2 Targets flashed only 2 times successfully. This test was repeated several times with more or less the same outcome.

In a second measurement, we test how long the flashing process takes at which baud rate. The time we measure starts when the user of the DSNAnalyzer enters the flashing command and ends when the result of the flashing process is displayed. We perform the measurement by flashing 7 Targets with a 70 Kbyte file one after another and compute the average duration.

The flashing process consists of sending the flash command to the DSN-Node, starting the boot loader, erasing the existing binary, flashing the new binary and verifying it and returning a message to the DSNAnalyzer. Because the binary is a hex-file and the DSN-Node is sending the binary data, we actually have to transmit less than half of this size (about 30 Kbyte). Flashing and verifying with a baud rate of 38400 should take both (30 * 1024 * 8) / 38400 = 6.4 s. Together they use 12.8 s. Additionally, we need time for starting the boot loader, erasing and for communication which take several seconds. This makes about 20 seconds.

The results of the measurement are shown in Table 2-7. We can see that we are in the range we expected. If we halve the baud rate, we get less than the double of the duration because the time to erase and start the boot loader remains constant.

Baud Rate	Average Duration
38400	22.2 s
19200	38.5 s
9600	70.3 s

Table 2-7: Target Flashing Duration

2.5.1.5 DSN-Node Watchdog Resets

Unfortunately, software running on embedded devices can also hang. They therefore use a technique called watchdog which automatically resets a device in case it hangs.

Here we measure the number of watchdog resets a DSN-Node performs within a specified time range. This property is determined because in case of a watchdog reset, the log of the

DSN-Node is reset what could lead to data loss. A watchdog reset is executed if a DSN-Node does not receive any ping from the DSN-Server within five minutes.

For this measurement we let 27 DSN-Nodes run over 63 hours. The watchdog reset statistics are shown in Figure 2-31. We can see that 11 Targets had 0 watchdog resets and that we have a mean value of about 1.5 watchdog resets in 63 hours and a standard deviation of 1.7. That is about one watchdog reset every two days.

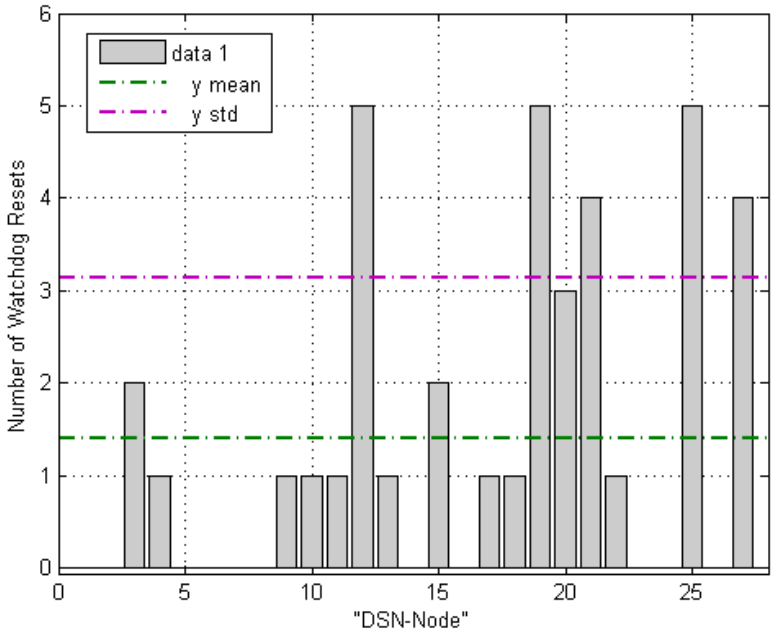

Figure 2-31: DSN-Node Watchdog Resets

2.5.1.6 Command Latency

The command execution mechanism is described in 2.4.2.1. Here we measure the roundtrip time of a command, i.e. the time between the user presses enter and the result is displayed.

This time depends on where the DSN-Node is located in the connection tree of the DSN. The closer the DSN-Node is to the GUI-Node (the root) in the connection tree, the less hops are needed to transmit a message.

The result of a measurement with a hop distance of 1 is shown in Figure 2-32. We have a mean value of 974 ms. The time spent between the DSNAnalyzer and the DSN-Server is 400 ms. This results in a mean value of 574 ms on the DSN-Nodes and the Target.

What looks a bit strange is that we have a very large latency at the first five commands. This could be a property of the routing mechanism of the DSN.

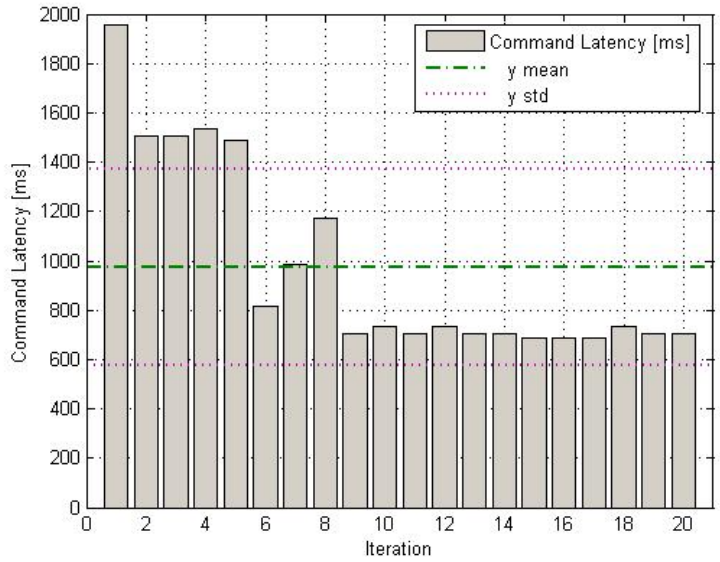

Figure 2-32: Command Latency 1 Hop Distance

In Figure 2-33 we see the result of a measurement with a Target that is 6 hops away from the GUI-Node. Here we have a mean execution time of about 1.5 seconds where the standard deviation is smaller. This results in a mean value of 1.1 seconds on the DSN-Node and the Target.

If we interpolate the two results we would expect an increase of 100 ms in latency for every additional hop.

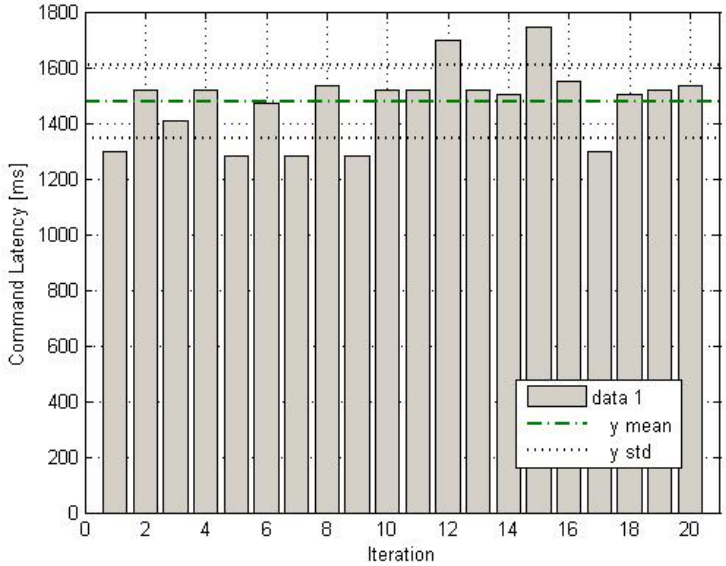

Figure 2-33: Command Latency 6 Hops Distance

2.5.1.7 Log Verbosity

In this section we determine how many events per second can be created by a Target. A DSN-Node receives events from the Target and creates a log message out of it. This log message is

then directly sent to the GUI-Node in case we are in the push mode.[56] In this measurement, we determine the number of log messages that n DSN-Nodes can send to the GUI-Node at different rates which are received correctly[57]. In Figure 2-34 we can see the result of this measurement. These results can be compared to the results from a measurement with almost the same test parameters at the ETH (see Figure 2-35). The only difference is the number of active DSN-Node in the system and the topology of them.

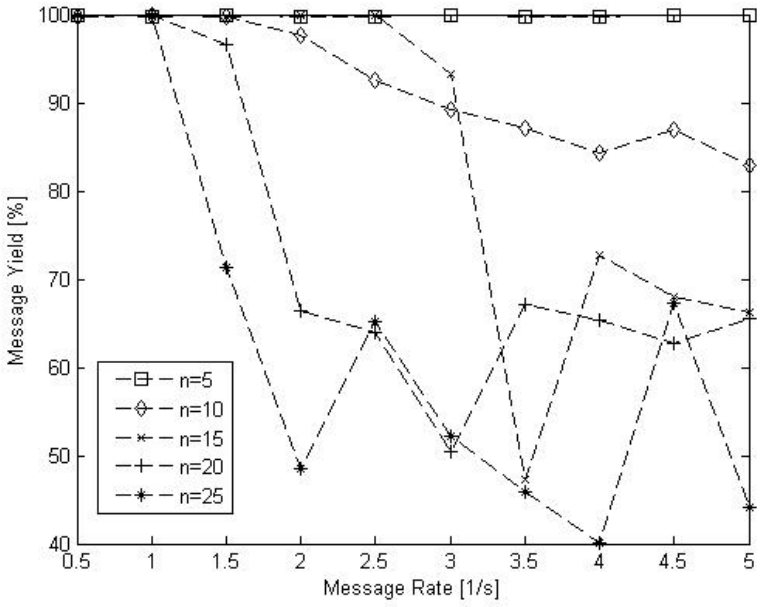

Figure 2-34: Correctly Received Log Messages

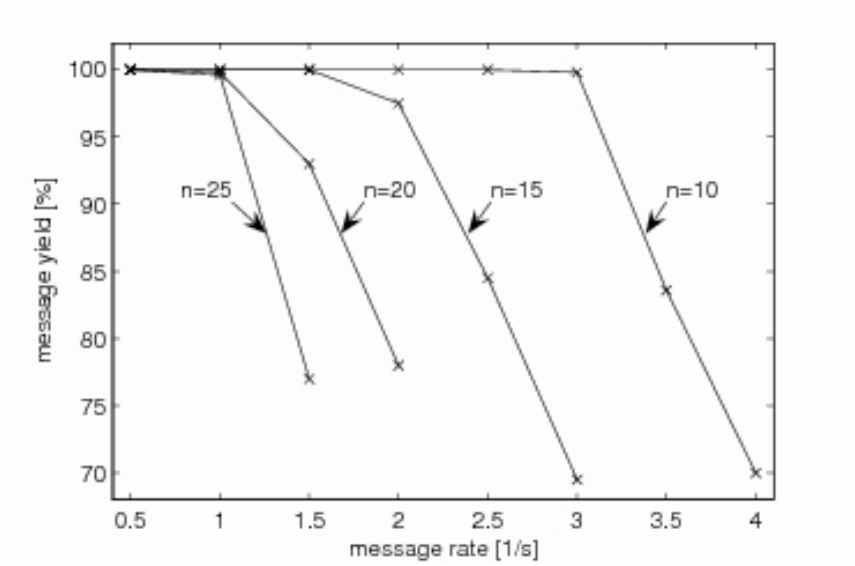

Figure 2-35: Correctly Received Log Messages (Results from ETH)

[56] "Push mode" means that log messages are directly sent to the GUI-Node at the time they are created. Although this is not the default mode of the DSN-Server, we used it in order to get a higher number of log messages back.

[57] This number is also refereed to as *yield.*

The reason why log messages can get lost is because they cannot be relayed. A log message is relayed by a set of DSN-Nodes until it reaches the GUI-Node. DSN-Nodes have a buffer for temporarily storing a fixed number of messages[58]. If one of these DSN-Nodes has a buffer which is already full, the message is thrown away.

What remains unanswered is the question if the log messages are lost only at the root of the DSN (the GUI-Node) or also at intermediate DSN-Nodes.

2.5.1.8 Influence of the “Blue Box”

Here we measure the influence of a “Blue Box” to the RSSI and PRR. The factors that could influence these metrics are the communicating BTnode, the Adapter Board with its various electronic components and the box itself. In order to do that, we take the result of another test with two Targets with a distance of 2 meters as the base. In this test, 1500000 frames were sent and all were received correctly. The range of RSSI was between -43.5 and -61.5 dBm, the range of noise between -108 and -126 dBm.

The test setup is the same as in the base test except that we use “Blue Boxes” instead of only A80 boards. We received all of the 1500000 sent packets without any errors and had an RSSI signal range between -42 and -43.5 dBm and an RSSI noise range between -105 and -121.5 dBm.

2.5.1.9 Range of DSN-Node

In this section, we determine the maximum distance between two DSN-Nodes such that they can communicate. This value is determined indoor in an office building and outdoor on a parking lot. The measurement was done by executing a command on the GUI-Node which triggers the DSN-Node to blink. The maximum distance was determined by the place where the DSN-Node did not receive the command anymore.

In the first measurement on a parking lot, we measured a maximum distance of 25 m if both nodes were on the ground. After we put the nodes onto a height of 70 cm, we measured a maximum distance of 105 m.

In the second measurement which we performed indoor, we also measured 25 m if the nodes are both on the ground. After setting the nodes to a height of 70 cm we reached a distance from the one ending of the building to the other which is 55 m (see point 1 and 2 in Figure 3-1).

2.5.2 Survey

For the DSNAnalyzer itself we performed a small survey because, when evaluating a GUI-Application, the acceptance of the users is more important than concrete metrics. This survey does only reflect the user acceptance of the Analyze View.

The participants of the survey stated that the DSNAnalyzer has the advantage of enabling the user to see what happened in the network, has zoom functionality and shows the information which is visible in the sequence chart also in a log and a 3D map. The negative points and feature requests are that there should be no need for a network file, that a selected events is

[58] The default value which is also used in the test is 10.

not optimal positioned after changing the zoom and that there should be a possibility for setting the colors of the events by the user.

As a result of that survey, we added the last two negative points and changed the system accordingly.

2.5.3 Conclusion

In this section, we conclude the evaluation of the DSNAnalyzer (see 2.5.3.1) and the DSN-Server/DSN-Node (see 2.5.3.2) and describe further work on these parts of the system (see 2.5.3.3).

2.5.3.1 DSNAnalyzer

We think that the only meaningful quantitative value for the DSNAnalyzer itself is the time to load trace files. We have found that on a fast computer with the 3D map turned off it takes about 12 s to load a very large trace file what seems feasible. However, laptops with slow mobile processors are often not fast enough to load very large trace files quickly.

As a result of the evaluation we can state that we need special tools in order to efficiently analyze complex protocols for WSN. This functionality is provided by the DSNAnalyzer.

2.5.3.2 DSN-Server / DSN-Node

What results from the power consumption and lifetime measurements described in 2.5.1.1 is that a “Blue Box” is running on batteries for about two days. For short measurements to evaluate the link quality at special places in the building or outdoors, this time is sufficient. However, if we keep in mind that the most important places for the “Blue Boxes” are the locations of existing fire detectors which are mounted on the ceilings, this time is too short. Additionally, for proving the failure-resistance of an algorithm, we need up-times of several days or even weeks. Therefore, some work has to be done on a low-energy mode for the DSN-Node because that device is responsible for most of energy consumption.

We measured the time to distribute a binary in the DSN and found that it takes about three minutes for a network with up to 30 DSN-Nodes. Another result is that the distribution duration does scale well with an increasing number of DSN-Nodes. The time to flash is about half a minute when using a high baud rate which is fairly negligible when flashing in parallel. The overall reprogramming duration of less than 4 minutes is therefore fast if we compare it with the process of manually reprogramming devices.

The Target flashing does not work well enough because there can be some DSN-Nodes which are unable to flash their Targets even after several tries. However, the speed of flashing is fast.

We have seen that we still have hang-ups and watchdog resets in the DSN. Watchdog resets do only matter if we do not send our events immediately back to the GUI-Node which can lead to data losses. However, hang-ups can stop measurements to be executed correctly.

The results on the command latency shown in 2.5.1.6 are important for the discussion of executing several commands at different Targets at the same time. We have to keep in mind that if we send a command to the DSN-Server which should be executed at several Targets, there could be a time deviation of up to several seconds in the execution time of the

command. This has impact on the design of application level tests because we cannot execute a command at several targets at the very same time[59]. This is one reason why we need Timed-Commands (see 2.5.3.3).

From the measurements of the log verbosity we know that we can produce about one event per second at up to 25 Targets. If we want to produce more events, we have to use a smaller set of Targets. However, if we are doing only measurements with a set of five Targets, we can produce up to five events per second or even more.

We measured the influence of the DSN-Node, the “Blue-Box” and the Adapter Boards which turned out to be negligible. We can also say that we have a maximum signal strength of -42 dBm.

In measurements about the range of the DSN-Node we found out that the maximum distance between two DSN-Nodes very depends on the height above ground. We can say that 2 DSN-Nodes can communicate with each other over more than 50 m in our office building when they are placed 70 cm above ground.

All in all we can say that we now have a very powerful system that enables the developer to control the Target-Network and analyze its performance in a manner which is not possible without using the system described in this thesis. Another important point is that we have already discovered a lot of problems and communicated them to the DSN development team, some of which have already been solved. However, because we work in an environment with restricted resources (such as battery lifetime) we have to do a trade-off between a very powerful and a very long-living system.

2.5.3.3 Further Work

Here we list features and improvements for the DSNAnalyzer which are meaningful:

- Extend the 3D Map to draw the ground plans and ceilings from information provided by the network file. Also drawing walls or obstacles could be possible.
- Show results of measurement also using arrows in the 3D map. This introduces a discussion of how much the views should depend on each other.
- A feature to create a link quality matrix which includes a histogram of RSSI values and the average PER of all possible combinations of a set of links.
- Use different shapes for representing events in the Sequence Chart. This helps the user to distinguish between different types of events.

We also list features and improvements for the rest of the DSN:

- The DSN-Sever should provide a reliable command execution function which blocks until the result is received.

[59] Something like that is needed if we want to simulate a link failure on the Target using a command.

- A Timed-Command feature on the DSN-Node. A Timed-Command is a common command with a time. The DSN-Node executes this command exactly at this time.
- A binary distribution feature which continuously checks the current software versions of the Targets which should be reprogrammed and starts flashing when all of them have the new binary.
- A reliable Target flashing function at the DSN-Server which automatically retries flash operations and returns the list of successfully flashed Targets.

3 WSN Link Quality Measurements

In this third part of the thesis we describe the reasons, the design and the results of the measurements we performed using the application described in chapter 2. We first introduce the problem (see 3.1), describe the methodical preceding (see 3.2), the measurements itself (see 3.3), provide a discussion (see 3.4) and give a conclusion (see 3.5).

3.1 Introduction

In this section we introduce the WSN link quality measurements by describing the problem (see 3.1.1), define the influencing factors (see 3.1.2), describe the metrics (see 3.1.3), describe why we have to perform tests (see 3.1.4) and draw a conclusion (see 3.1.5).

3.1.1 Problem Specification

As mentioned in 1.1.3, we need real-world measurements of the link quality on the one hand to compare them with the results from the simulation and on the other hand to gain a deep understanding of the radio wave propagation in office buildings which serves as input to the radio models used for simulations again. Additionally, most protocols rely one some assumptions about the link quality which have to be determined in different conditions.

In this section, we show how radio waves propagate (see 3.1.1.1), describe why we need reliable protocols (see 3.1.1.2) and mention the work which is related to this thesis (see 3.1.1.3).

3.1.1.1 Radio Wave Propagation

In the free space, radio waves propagate according to the inverse-square law what means that when the distance is doubled, the signal strength is divided by four. But because we are on earth and not in the free space, we have a set of additional factors which disturb the propagation but also allow the signal to reach areas which are not in line-of-sight [Wikipedia 2006A, Radio Electronics 2006]:

- Absorption: Radio waves with high frequency are partially absorbed by obstacles like walls.
- Reflection: Radio waves are reflected by objects. When reflection occurs, parts of the signal can be lost due to absorption or because of passing the obstacle.
- Slow Fading (Shadowing, Attenuation): Radio waves are shadowed by obstacles (this is a combination of absorption and reflection) [NGMN 2006].
- Diffraction: Radio waves can move around objects.
- Refraction: Radio waves can undergo refraction.
- Fast Fading (Rayleigh-Fading): Because of the multi-path propagation, which results from the effects described above, signals arrive at the

receiver at different times which could lead to amplification, attenuation or effacement [NGMN 2006].

3.1.1.2 Reliability

What follows from radio wave propagation properties described above is the complexity of deriving the strength of a signal at a given place. Additionally, if we are in an office building, which is the main application area of fire detectors, the propagations become even more complex [Reijers 2004, 232].

However, if we would like to design e.g. an efficient MAC protocol or a physical layer, we have to consider the properties of the radio wave propagation and the resulting link quality.

For designing WSN applications, we have to use reliable protocols which allow the correct transmission of messages. The design of these protocols depends very much on the channel characteristics which have to be determined by measurements.

3.1.1.3 Related Work

In this section, we present work which is related to ours and which we use to verify the results of our measurements.

In [Nguyen/Thiran 2006], the authors propose to use end-to-end data transmissions to say something about the quality of links. They also show that most data in a sensor network is transported over a few number of links and that basically there are good or bad links but hardly any in between. Another finding is that links are very stable if there are no external influences.

The authors of [Willig/Mitschke 2006] analyzed the detailed link behavior and checked different error control mechanisms such as *Forward Error Correction* (FEC) and *Automatic Repeat Request* (ARQ). They propose that FEC is not applicable in WSN because bit errors most often come in bursts.

In [Cerpa et al. 2005], the authors analyzed the temporal properties of links in WSN. They state that trying to use multiple paths concurrently when sending a packet does not help to improve the end-to-end link because if one good link performs badly at a given time, all other good links are very likely to perform also badly at the same time.

The authors of [Son et al. 2005] determine the influence of concurrent packet transmissions in WSN. They introduce an extended SNR which includes interference and show that this value should be higher than a specified threshold in order to guarantee successful packet transmission. They also state that it is hard to determine the level of interference in case there are multiple interferers.

[Thelen et al. 2005] describe propagation of radio waves in potato fields. They show that the foliage influences the propagation of the radio waves. They also state that at times with high humidity (in the night and when it is raining) radio wave propagation is better.

[Zuniga/Krishnamachari 2005] shows how radio waves propagate and describes a way to create link layer models for different environments.

The authors of [Reijers et al. 2004] did a detailed research of the radio wave propagation in different indoor and outdoor environments. They investigated different factors like multi-path

effects, directionality and human activity which can influence the link quality. They also looked at interference, symmetry and the correlation of RSSI and link quality. Additionally, they proved the existence of a gray area which includes good as well as bad links.

In [Zhou et al. 2004] the authors describe the impact of radio irregularity on WSN. They show that radio irregularity comes from multiple factors like directionality and small changes in the sending power of radio transceivers. As a result of their research, they created the Radio Irregularity Model which can be used for simulations.

The authors of [Zuniga/Krishnamachari 2004] describe the causes for the gray area. They show that the gray area is not a result of hardware non-ideality but exists because of multi-path effects. As a result they propose an expression for the PRR (see Table 3-3) using the distance to the sender.

The authors of [Zhao/Govindan 2003] did measurements of the link quality in an office building, a habitat and in a parking lot. They show the existence of a gray area where communication is difficult and describe the reasons for such a region.

Most of the papers presented above used the popular MICA2 sensor node with the CC1000 radio chip for their measurements. Because of that, the results of our tests cannot easily be compared with those presented in the papers.

3.1.2 Influencing Factors

In this section we describe factors that influence the metrics shown in 3.1.2.3. Influencing factors can be divided into internal factors (see 3.1.2.1) which are related to the hardware and external factors (see 3.1.2.2) which are results of changes in the environment. In Table 3-1 and Table 3-2 the number 6 means easy manageable and respectively, 1 means not manageable.

3.1.2.1 Internal Factors

In Table 3-1, we describe the internal factors which are related to the hardware and most often can be managed easily.

Factor	Manageability
Channel (Frequency)	6
Preamble length	6
Adjustments Values (Target characteristics)	6
Modulation Schemes	6
Sending Power	6
Remaining Battery Power	4
Coding Schemes (4B5B, Manchester, SECDED) (see [Zhao/Govindan 2003, 10])	6
Frame Size	6
Receiver Noise (see [Zuniga/Krishnamachari 2004, 2])	3
Antenna-Diversity	6
Receiver Sensitivity [Zhou et al. 2004, 126]	4
Amplifier Properties	4
Battery powered / USB powered	6

Table 3-1: Internal Factors

3.1.2.2 External Factors

External factors (see Table 3-2) consist of all the things which are not dependant of the hardware.

Factor	Manageability
Receiver Location	3[60]
Fading[61]	1
Directionality of the sender (vertical/horizontal)	4
Directionality of the receiver (vertical/horizontal)	5
Humidity	2
Room Temperature	2
Sender/Receiver proximity to the desk/ceiling	3
Interference from other Devices	2

Table 3-2: External Factors

3.1.2.3 Conclusion

We have shown a variety of internal and external factors that influence the radio wave propagation. It can easy be seen that creating a fast simulator model using all these factors is nearly impossible. However, if we can quantize the impact of these factors we can provide information that helps designing models which are accurate enough for simulation. Additionally, a quantization helps also for the design of algorithms for WSN.

[60] The number 3 comes from the fact that sometimes the location of the receiver is determined by non-radio requirements (e.g. optimal placement of a fire sensor).

[61] Fading is more a result of the other external factors than a factor itself. If we talk about fading we often mean the variation of the signal strength. See http://en.wikipedia.org/wiki/Fading.

3.1.3 Metric Evaluation

In order to measure the link quality in WSN we need a way to quantify it. Therefore, we list possible metrics in Table 3-3.

Metric	Description	Evaluated
RSSI (Received Signal Strength Indication)	The strength of the received signal for a receiver in dBm.	Yes
Packet Error Rate/Ratio (PER)	The number of incorrectly received packets out of the total number of packets. Incorrectly received packets are the sum of "missed" packets (= packets where the Start Frame Delimiter (SFD[62]) was not seen) and the packets with at least one bit error.	Yes
Packet Reception Rate (PRR)	1 – PER [Zuniga/Krishnamachari 2004, 1]	Yes
Bit Error Rate/Ratio (BER)	The number of incorrectly received bits out of the total number of bits (e.g. BER = 10^{-3} means that out of 1000 sent bits 1 bit is sent with error) [Wikipedia 2006B].	Yes
Average Bit-Burst Length	A metric that tells us something about the distribution of bit errors. Bit errors can be very seldom but if they arise then in a burst, or bit errors are linearly distributed over the time [Willig/Mitschke 2006].	No
Average Packet-Burst Length	Average of how much packets received one after another are missed/have bit errors.	No
Average Number of Bit-Errors per Packet	-	No
Signal-to-Noise Ratio (SNR)	The ratio of meaningful information (signal) to the noise.	No
Signal-to-Interference-Plus-Noise Ratio (SINR)	SNR that explicitly includes the signal strength from interfering senders into noise [Son et al. 2005, 1].	No
Required Number of Packets (RNP)	The required number of packets to be sent before a packet is successfully received (When using ARQ) [Cerpa et al. 2005, 3].	No

Table 3-3: Link Quality Metrics

For getting a first impression of the complex behavior of radio propagation in office buildings, we do not want to use every metric listed in Table 3-3. We therefore choose a set of important metrics.

In order to compare our results to previous work, we have to use the RSSI and PER. For detailed link analysis, we are also interested in the BER. The SNR is a result of RSSI measurements from the signal and the noise and therefore needs no additional measurements. The rest of the metrics are considered to be used in cases we explore special behavior.

Because the PER can be computed from the number of packets which have BER greater zero, we just have to explicitly measure BER and RSSI in the Target software.

[62] An SFD marks the start of a frame.

3.1.4 Tests

In 2.4.2.4 we described what a test is and how it is performed using the DSN. Here we want to describe why we do perform tests in the way described.

In order to get values for the metrics chosen in (3.1.2.3), we have to perform tests which consist of Targets which send test-frames and receivers which receive them. A test-frame is just a set of bytes. The Target which is continuously receiving has a copy of the reference test-frame which is sent and compares it to the test-frame received. Using simple bit operations, we can get the number of bits which do not match the reference test-frame. Additionally, we measure the RSSI while receiving and also shortly after reception to get the RSSI value of the noise.

Because the DSN is limited in its resources, we need a way to control the granularity of the events that are created at the Target. We therefore included a logging mechanism at the Target which allows controlling the number of events which are generated during the receiving phase.

3.1.4.1 Test Setup

In our measurements we use the Siemens A80 (see 2.1.2.1) with FSK-Modulation and a baud rate of 4.922 Kbit. Most of the measurements are done in our office (see Figure 3-1).

The placement of our Targets is chosen differently than in other work (see [Reijers et al. 2004, 225; Zhao/Govindan 2003, 3]) where the Targets are paced in a chain topology. In our measurements, the Targets are most often placed on locations in the office building which seem difficult according to fading effects. We do this because in the fire detection application, for which we perform these measurements, the Targets are also placed in complex topologies.

For all our measurements, we used channel 20 in the 868 ISM band. In order to avoid collisions with other communicating devices we use an Excel sheet which is used by all developers to reserve frequencies. If not specially defined, we send using 9.7 dBm. The test frame is a string with the size of 26 bytes (208 bits).

For sending the A80 uses the antenna A.

3.1.5 Conclusion

We have defined radio wave propagation, influencing factors as well as metrics. Now we use this information to plan the measurements which should be performed in this thesis (see 3.2.2).

We want to remark that the link quality measurements performed in the last part of the thesis are not only for answering questions related to radio wave propagation but also to evaluate the software described in chapter 2 under real conditions and over a long time.

3.2 Methodical Proceeding

As already done in chapter 2, we present the methods we used in order to provide accurate results and allow other developers to see the results of our measurements.

3.2.1 Wiki Website

For temporary storing the test results and related discussions we create Wiki-Sites (see 2.2.6) for each successful measurement. On this website we list the test parameters, pictures of the result charts and links to the test file and the test result file which allows other developers to do analysis on the test material.

Because a Wiki-Site should be easily creatable, we create a template for link quality measurements which already includes the framework for describing the result of a measurement.

3.2.2 Roadmap / Planned Measurements

We have briefly introduced how radio waves propagate in the free space, how this situation changes if we are on earth in office buildings (see 3.1.1.1) and listed the internal and external factors which influence this propagation (see 3.1.2) and as result, change the values for the metrics mentioned in 3.1.2.3.

In order to plan the measurements shown in 3.3, we analyze the influencing factors to check which are important and which do influence the metrics in office buildings.

We start our measurements by leaving the internal factors static because they are not as interesting as the external factors. When deciding which external factors we want to examine more detailed, we have to take those factors which can be managed.

One of the most important factors, we think, is the placement of the nodes because about this factor we can show that it influences the signal strength. However, there are also far places from the sender that have better signal strength then close places [Reijers et al. 2004, 226]. Therefore, it is obvious that we have to analyze this influencing factor (see 3.3.1.1). In all the measurements we do not try to move Targets until we find a good place, but we try to describe why we have a poor link quality at given place.

Fading primarily comes from obstacles or humans that change their positions due to several reasons. Therefore, we would like to analyze the fading e.g. at day and night (see 3.3.1). We found that fading is largely influenced by human activity and saw that there is less fading at night than at daytime.

Interference is an influencing factor that depends very much on the environment. Because we are performing measurements for prototyping fire sensor networks, we are in difficult environments such as office buildings with a lot of interference (see 3.3.2). We found that interference from different channels really influences the link quality, especially if the channels are close to each other.

An influencing factor that is very easy to manage is directionality. Here we can analyze if the sender or receiver directionality does influence the link quality. Another thing to determine is if directionality only has influence if the Targets are close to each other (see 3.3.3). We found that the influence of directionality is not that much.

Something we also want to analyze is the temporal behavior of the factors described in 3.3. We want to see if RSSI values change between milliseconds or between hours or even both (see 3.3.1.4). We found that there could be large fading about short and long periods of time.

3.3 Measurements

In this section we describe the measurements and show how they are performed. The measurements include a determination of fading and link quality in our office (see 3.3.1), a valuation of interference (see 3.3.2), a description of the influence of sender or receiver directionality (see 3.3.3) and a description of link quality at different sending power (see 3.3.4).

3.3.1 Signal Strength / Link Quality / Fading

In this section we determine the signal strength and the resulting link quality in our office building. We analyze changes in signal strength, which also refers to as *fading*.

3.3.1.1 Base Measurements

In these base measurements, we often use a fixed set of Targets which are distributed all over the floor (see Figure 3-1). We do these measurements in order to get a feeling of where we can receive all packets without errors and where not. We are also interested in where we have good signal strength and where not. In these first measurements we do not analyze any external circumstances such as interference and people moving around.

In Figure 3-2 we see the results of such a measurement with one sender (80640419) and 6 receivers (the y-axis of the chart). The bars show the range of measured RSSI values of the receivers where the red bars are for RSSI values of correct received frames, the blue bars for the RSSI values of faulty received frames and the green bars for the RSSI values of the noise. If a bar is large, we know that a this time and place we have a large change of signal strength what also refers to fading. We can see that we have RSSI values around -40 dBm if the receiver is near to the sender (such as 80640400) and values around -105 dBm if the receiver is on the other side of the building at a distance of about 50 m with 4 small walls in between. We can also see a receiver (806403ec) which is near to the sender but has large fading. The noise is measured with an RSSI value between -120 dBm and -110 dBm.

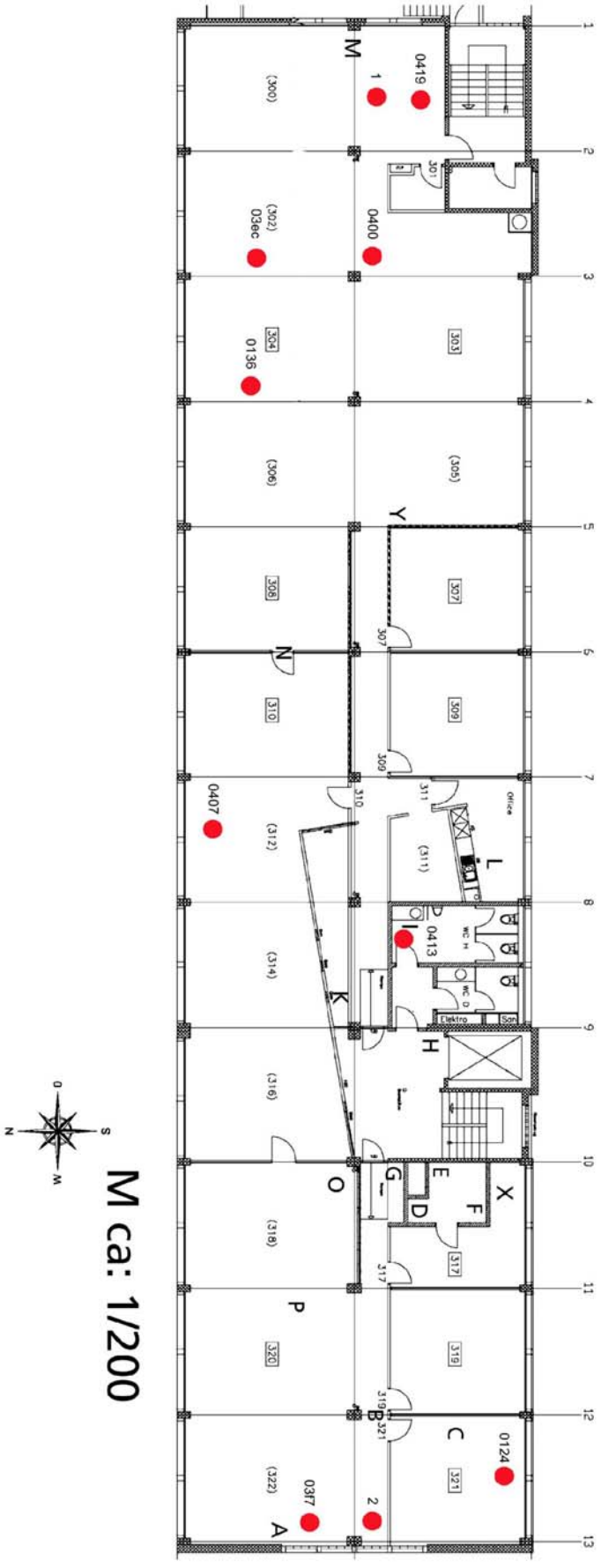

Figure 3-1: Floorplan SBT Gartenstadt 2a Floor 3

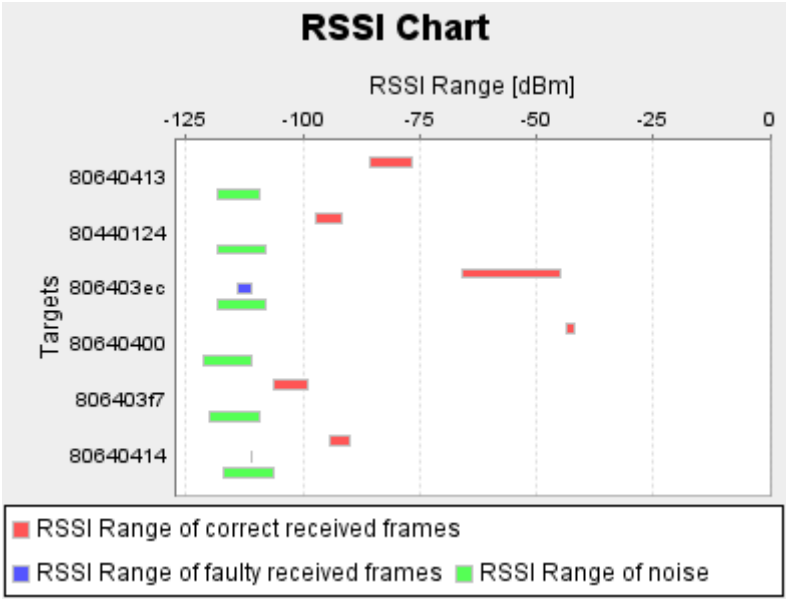

Figure 3-2: RSSI Chart of simple base measurement

3.3.1.2 EMC Measurements

In order to be sure that our measurements reflect physical phenomena and not internal software or hardware bugs, we perform a “burn-in” test in a special isolated cell used to determine Electromagnetic Compatibility (EMC). This cell provides very large signal attenuation what allows us to perform tests without external influences of other devices and hardly any multi-path propagation because of special attenuating walls.

The measurement is done with an iteration of 1500000 frames and a period of 120 ms sent by a sender that is 2 meters away from the receiver. The duration of the test is 50 h. We perform the measurement on channel 20, a period of 120 ms, sending power of 9.7 dBm and a preamble-length of 16 bytes.

The result is that we received all 1500000 test-frames correctly. We had a maximum RSSI of the signal of -43.4 dBm and a minimum of -61.5 dBm. The noise level was between -126 dBm and -109.5 dBm.

3.3.1.3 SFD Noise Reception

In most of the results of measurements with an iteration of more than just several hundred frames, we can see that we have always faulty frames with a very low RSSI (see Figure 3-13). The fact that the RSSI range of these faulty frames is included in the range of the noise is no accident: The Target is continuously listening while performing a test. If it detects an SFD it receives the following bytes and compares it against the reference frame (see 3.1.4). But because noise consists of random bit patterns there is also the probability of receiving an SFD out of the noise. This is exactly what happens when a frame with many bit-errors at a very low RSSI is received.

The problem of this effect is not only that we have a number of frame-errors in the results of a test which result from noise but also that during reception of a frame out of the noise the receiver does not listen to the sender. This has the effect that a small number of frames are not seen by the receiver (see Figure 3-3).

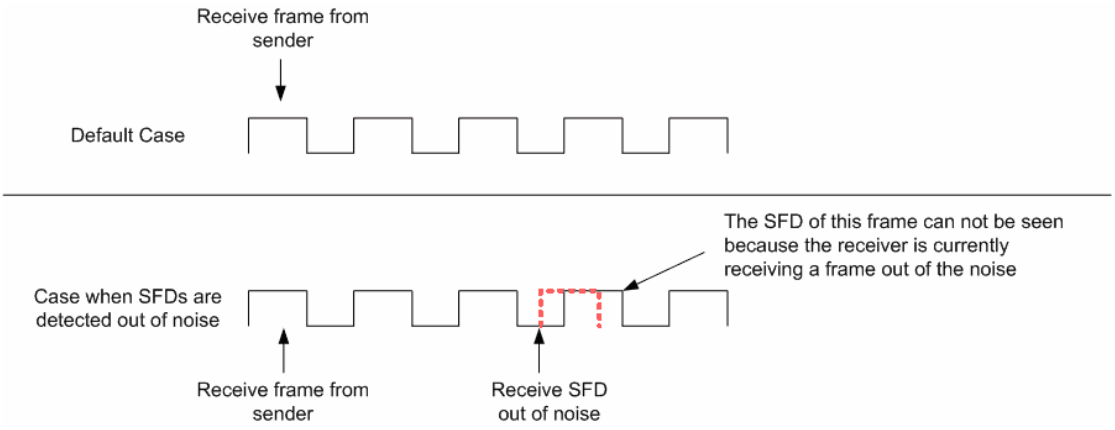

Figure 3-3: SFD Noise Detection

Because of the effect described above, we perform a special measurement in order to determine the number of wrongly detected SFDs (see Figure 3-13). We measure how much frames are detected out of the noise if no sender is active. For that we use 15 Targets which are listening for 12 hours. The result of this measurement is shown in Figure 3-4. We have an average of 297 received SFDs per Target what is 25 SFDs per hour. We also see that the values do not very depend upon the location of the Targets but more reflect the properties of the hardware.

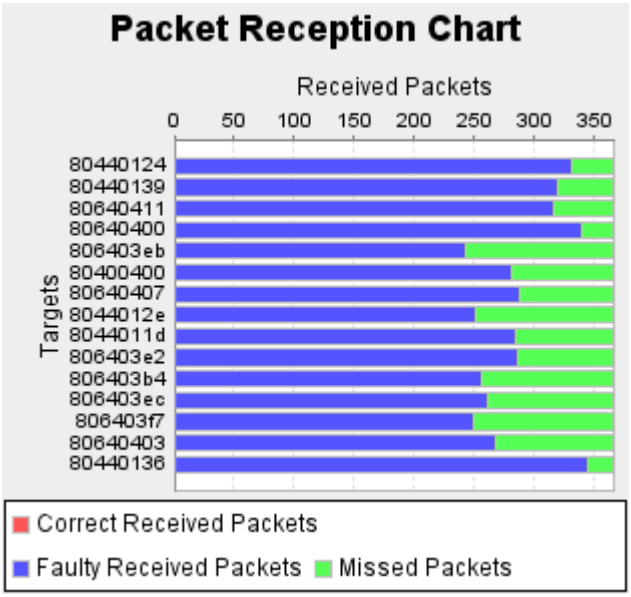

Figure 3-4: SFD Noise Detection Measurement

As a result of the observations above, we introduce a threshold value at the Target software. This threshold can be used the set the minimum RSSI value an SFD must have in order to be accepted. After using this software with a threshold of -100 dBm, we could produce results as shown in Figure 3-5 where no wrong SFDs are detected.

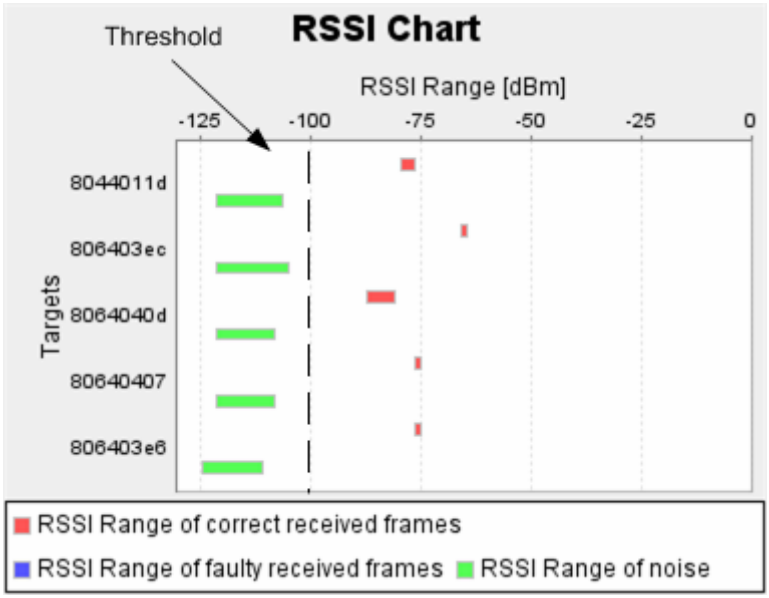

Figure 3-5: RSSI Chart (No wrongly detected SFDs)

3.3.1.4 Temporal Fading Properties

In this section, we determine the period of fading effects. We want to know if the signal reaches its minimum and maximum several times per second, per minute or even per hour. This property is important for protocol design because some protocols retry transmissions in case of a failure after a certain time. In order to determine this time, we need to know the properties of large changes of the link quality.

In these measurements we concentrate on links with high fading, i.e. links that have changes in their signal strength of more than 25 dBm.

In a first measurement, which is performed at daytime, we evaluate a link which we saw that is has fading of about 27 dBm. The receiver is located 7 meters away from the sender which sends one frame every second. The test duration is two hours and 45 minutes. The whole link quality chart is shown in Figure 3-6. We can see that we have fading all over the time except during the period between frame 6000 and 7000 with almost no fading. All the faulty frames in the range of the noise are a result of the effect described in 3.3.1.3. What is not clear is the reason why we have a bunch of bit error with high RSSI between the frames 7500 and 10000.

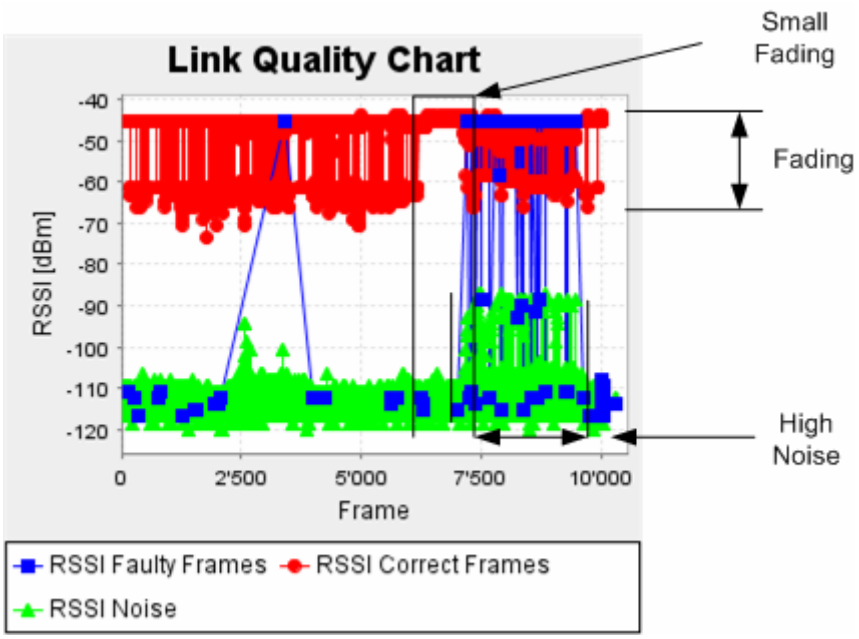

Figure 3-6: Link Quality Chart Link 1, Daytime

In Figure 3-7 we see a part with 100 frames from the measurement shown in Figure 3-6. We can see two types of fading: Some frames are received with a RSSI that is up to 20 dBm higher or smaller that at the previous frame. The second type is a change of the RSSI that holds for about a minute.

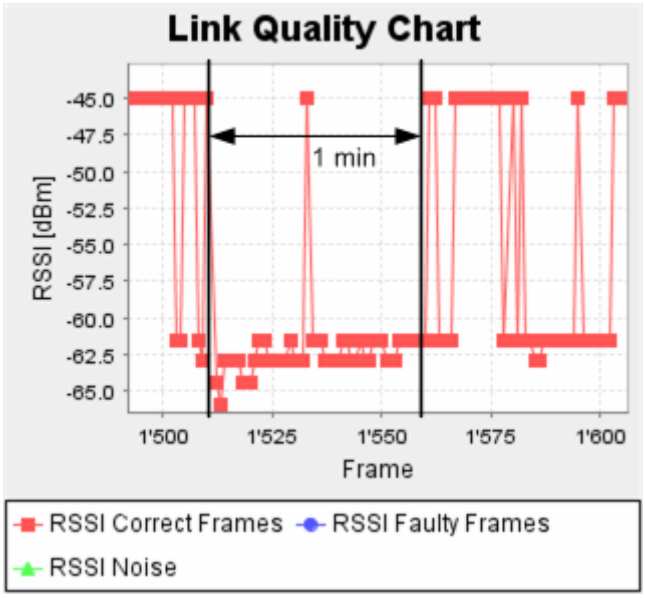

Figure 3-7: Link Quality Chart Detailed Link 1

In a second measurement, also performed at daytime, we also include Targets which are more far away. In this measurement we sent 5000 frames with a period of 500 ms. The test duration was 42 minutes. In Figure 3-8 we can see the Link Quality Chart of a link to Target 80640413 (see Figure 3-1) with a distance of 30 m through several small walls. We can see that there is a change of the signal strength over the whole time with three bad phases around frame 1000,

frame 3000 and frame 4000. Although this link has large fading, we have received almost all packets without errors.

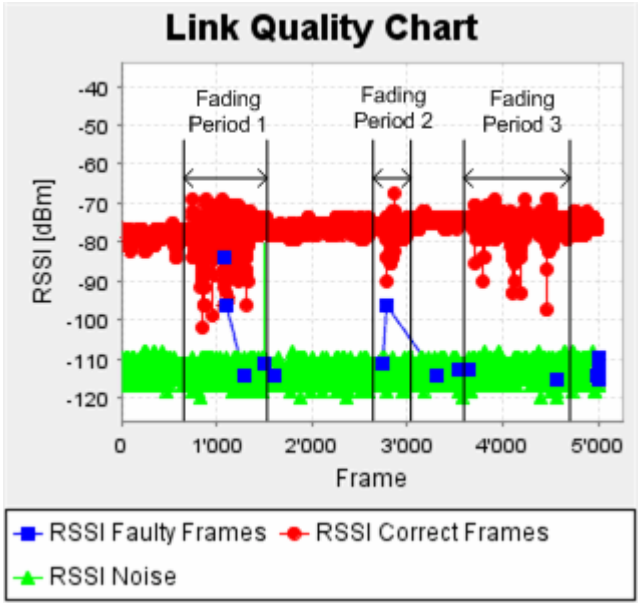

Figure 3-8: Link Quality Chart Link, Daytime

In Figure 3-9 and Figure 3-10 we can see the difference between a 3 m link between Target 80640419 and Target 80640400, and a 35 m link between Target 80640419 and a Target, which is located 35 m away on the next floor at position H (see Figure 3-1). In Figure 3-9 we hardly have any fading because the signal changes only between 1.5 dBm. In Figure 3-10 we can see that we have more fading, but do not yet have bit errors.

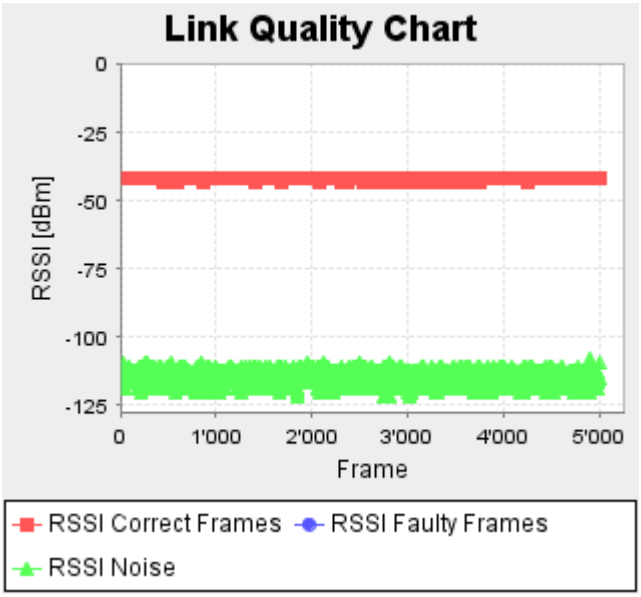

Figure 3-9: Link Quality Chart Link 3, 3m-distance

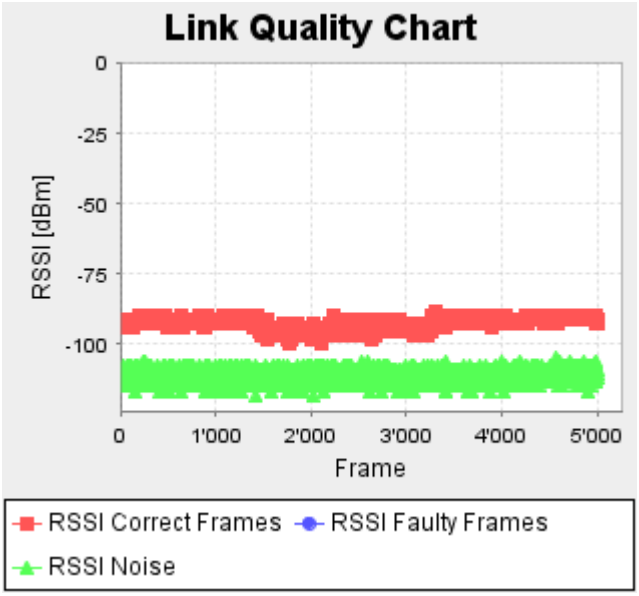

Figure 3-10: Link Quality Chart Link 4, 35m distance / 2 floors

In Figure 3-11 we can see a hypothesis of the decrease of RSSI with increasing distance. Additionally to distance, obstacles also reduce the signal strength and increase the fading.

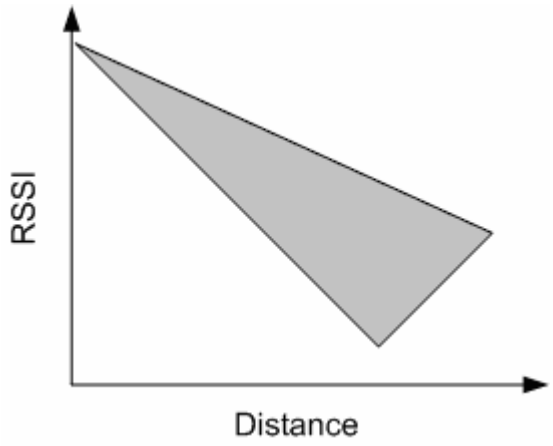

Figure 3-11: Signal Strength / Distance

3.3.1.5 Day and Night

In this measurement we compare the fading during the night against the fading during the day. The measurement consists of 50000 frames sent with a period of 125 ms with 9.7 dBm sending power and takes about 1 h 45 min. The sender is placed at the ceiling in our office.

The result of the measurement during the day is shown in Figure 3-12. We can see that we have fading only at two of the three receivers. This is because Target 80640400 is very near to the sender and also located at the ceiling. However, if we look at 80640413, which is also at the ceiling but has a distance of about 40 m to the sender, we can see fading at day. We also have fading at Target 80640407 which has a distance of 20 m and is located on the desktop of an employee (see Figure 3-1).

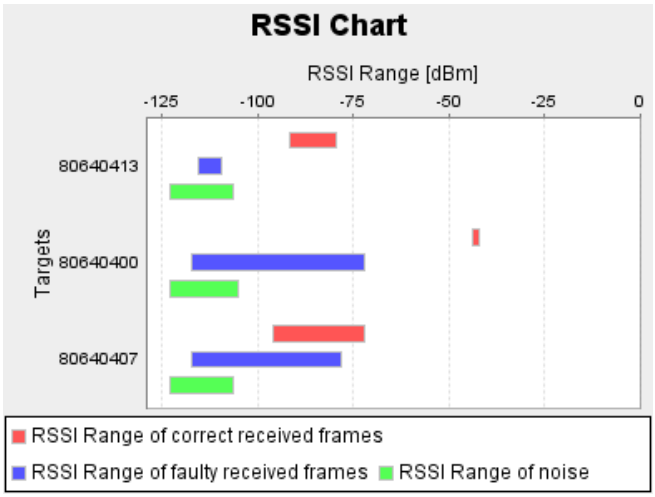

Figure 3-12: Fading Day

In Figure 3-13 we see the result of the same measurement at night. We see that we have almost no fading.

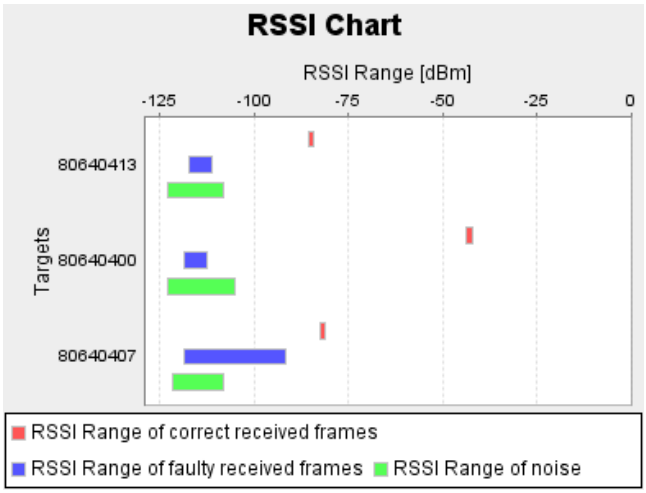

Figure 3-13: Fading Night

In order to determine the difference of fading between day and night, we perform a second measurement set. We performed 6 iterations of a test with 50000 frames, a period of 125 ms and sending power of 9.7 using 4 Targets. The duration of one iteration is 1 hour 45 minutes. The first 3 iterations are performed at daytime when people are in the office, the second 3 iterations are performed at night (In the first test of the night there could be movements of person because of the cleaners).

In Figure 3-14 we can see the fading differences between day and night. We have drawn the difference between the maximum and the minimum value of the RSSI of correctly received frames what is also called *fading*. We see that the overall fading at daytime is larger than at night what seems obvious. What we clearly can see is that at the Target 80640400, which is like the sender on the ceiling and only 3 meter away from it, has hardly no fading. The Target 806403ec has the largest fading because of its special location which is between two metallic objects (see Figure 4-8). That was also seen in other tests (see Figure 3-2).

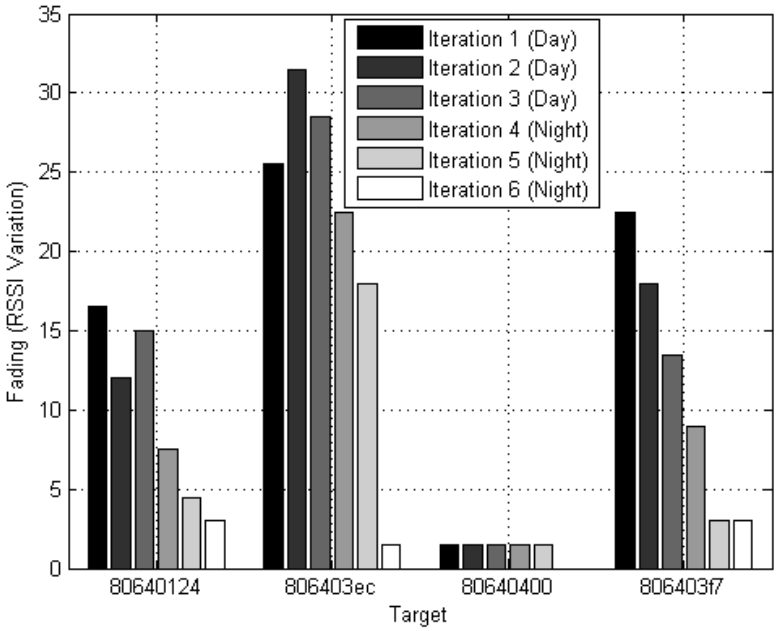

Figure 3-14: Day/Night Fading

In Figure 3-15 we can see the number of successful received packets out of 50000.[63] We can see that the first 3 tests are worse than the second 3 tests. This result is due to people moving around and causing (fast-) fading. We can see that only two Targets perform worse in the first 3 tests (80640124 and 806403f7). These are exactly the two Targets which are not near to the sender and are therefore influenced by people and movements of obstacles.

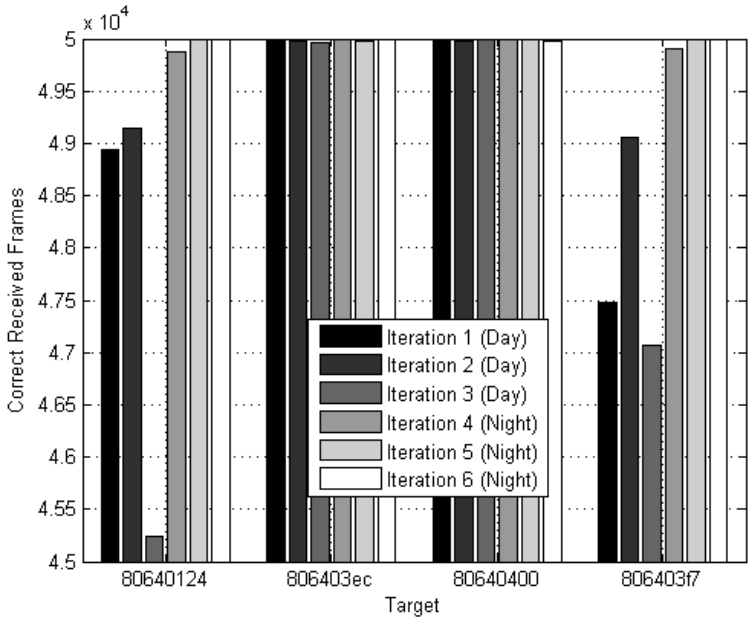

Figure 3-15: Day/Night Received Packets

[63] Note that we took not the same set of Targets as in the previous measurements due to software-hanging problems.

3.3.1.6 Special Places

In order to determine the behavior of the Targets in regions with high fading, we have to find such places. On the one hand we already know such places from initial measurements but on the other hand we use an analog receiver to find them.

One place about which we know it has high fading is the location of Target 806403ec shown in Figure 3-1 and Figure 4-8. This Target is between two metallic boxes under the desk. In order to proof if the fading does not come from hardware failures, we also replaced the hardware.

Another special place was found using an analog receiver. It is in the restrooms where we have reflecting elements such as a mirror and a heater (see Figure 4-4). However, in Figure 3-13 we can see that we do not have large fading when performing our measurements.

A third special place was the basement of our office building (see Figure 4-9). There we found that we can have over 30 dBm fading over the night without any human activity and no lights going on and off. This shows that fading is not a phenomenon that only arises in environments with visibly changing conditions, but also in static environments like basements.

3.3.1.7 Measurement with Threshold

In this measurement we determine the PER of a receiver which is on the same floor as the sender. The distance between sender and receiver is 50 m, we are sending with 9.7 dBm on channel 20 with a period 120 ms over 50000 frames. The duration of this measurement is 1 hour 40 minutes. We first do the measurement at night were no people are in the office what reduces the fading. We also used a threshold of -99 dBm. The result of the measurement is that all receivers with an RSSI range higher than the threshold have a PER between 0 and 0.0001.

3.3.2 Interference

Since the beginning of the measurement part we noticed different results in our measurements if devices in our environment were continuously communicating. Although these devices do not use the same channel as we do, we see some impact. We know that there is a *SigmaSpace* system (see Figure 3-16) which usually communicates on one channel, but also uses different channels in case of collisions.

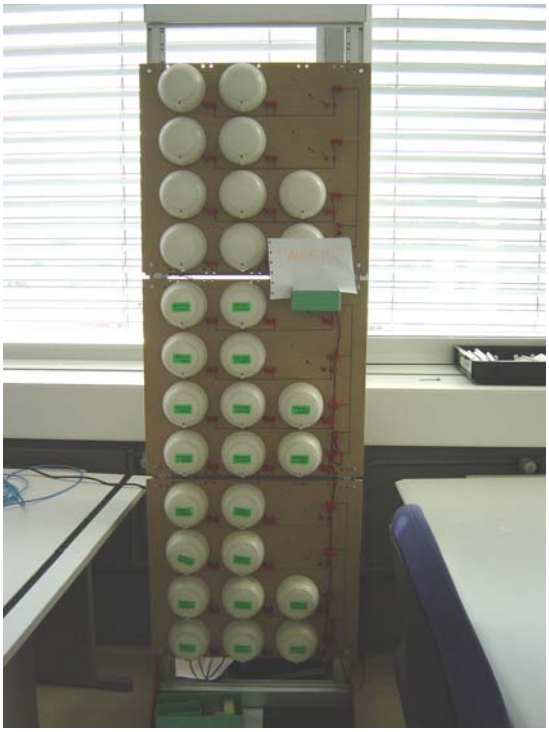

Figure 3-16: SigmaSpace Test System

In order to measure the influence of interfering devices to our communication we have to perform some tests.

3.3.2.1 Link Symmetry

Because we know that in Room A we have interference from a different fire detection system, which is in development, we perform the following test: First the Target in Room A is sending and the Target in Room B is receiving, afterwards they change roles. We perform this measurement using 8 iterations while at each of them 5000 frames are sent in a period of 125 ms. The duration of one iteration is 10 minutes.

The interferer is a testbed of wireless fire detectors which has a base channel but is also allowed to send in a set of different channels. Because of this property and the fact that the interferer is not always sending, it is very difficult to characterize the interference in a detailed manner. We therefore just see the interfering devices as a source of interference which influences receivers in the same room.

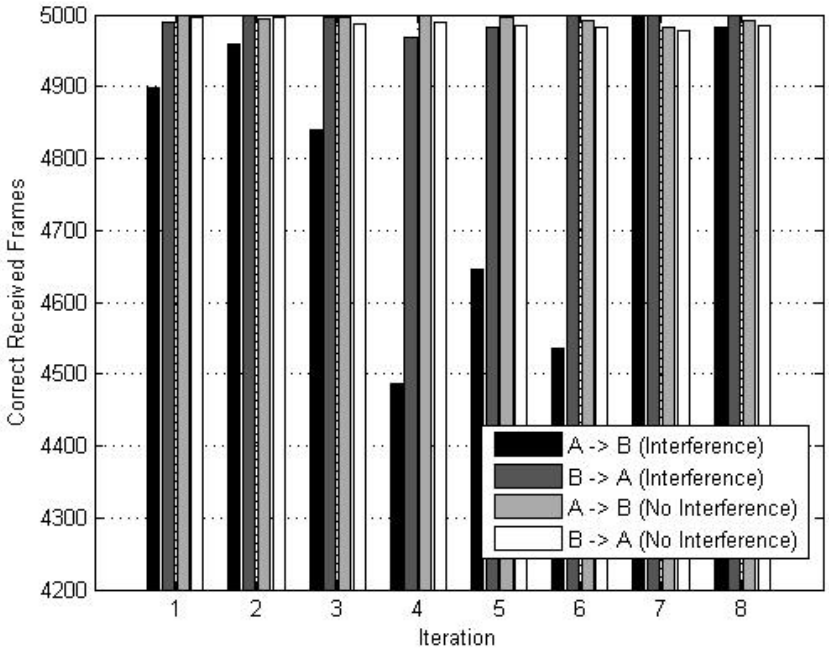

Figure 3-17: Link Symmetry Measurement Results

The results of the measurement are shown in Figure 3-17. We see that in the first measurement, when devices in Room B were communicating, we have a good link quality if we send from B to A but a bad link quality the other way around (when the receiver was in the same room as the interferer). In the second measurement we disabled the interferer what lead to good reception in both directions.

The reason why in iteration 7 and 8, in the first measurement, both directions are good is that the interferer was down at that time.

After this measurement we know that other communicating devices influence our communication. Because we cannot clearly determine when this interferer is communicating on which channels we also perform a measurement with a controlled jammer (see 3.3.2.2).

3.3.2.2 Jammer

In this test we measured the influence of a jammer near the Target. They are both sending with 9.7 dBm, an iteration of 10000 and a period of 125 ms. The duration of one iteration is 20 minutes. As a jammer we used another A80 which is positioned first 10 cm (Pos 1) and afterwards 100 cm (Pos 2) away from the receiver (see Figure 3-18). The sender is 15 meters away from the receiver. In both positions we set the jammer channel to 21, 23, 50 and finally turned it off.

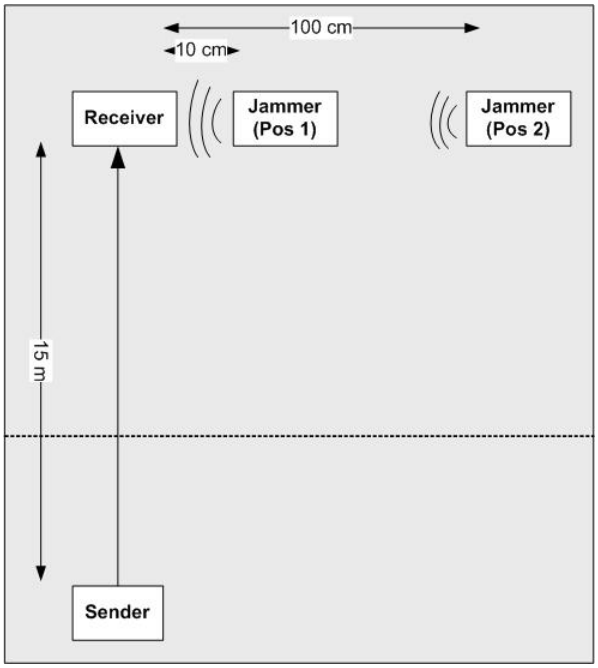

Figure 3-18: Jammer Position

In Figure 3-19 we see the results with a jammer on Pos 1. We can see that almost all packets are received when the jammer is off and also with a jammer with frequency 50. We can also see that if the channels are closer to the channel 20, which is used by the sender and the receiver, we almost get no correct packets.

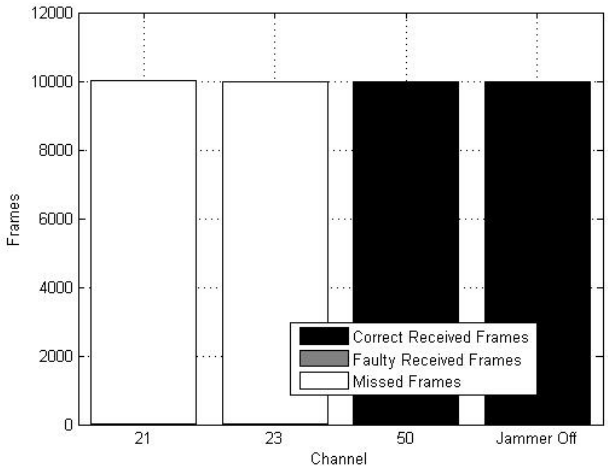

Figure 3-19: Jammer Result 10 cm Distance

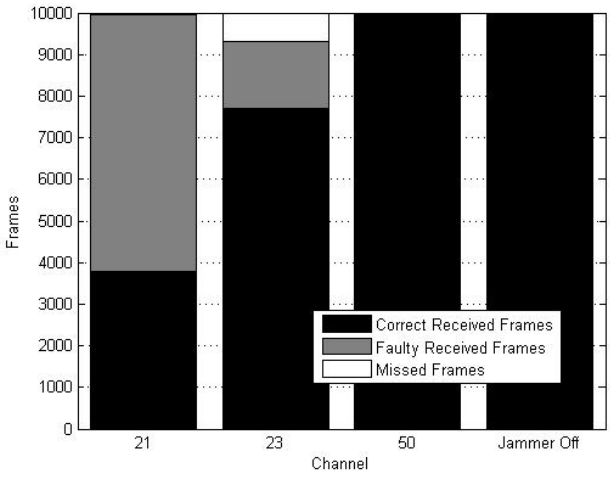

Figure 3-20: Jammer Result 100 cm Distance

If the distance is increased to 100 cm (see Figure 3-20), we can see the same effects but not as strong as with 10 cm distance. However, if the jammer is at a frequency close to the sending frequency we observe strong influence.

3.3.3 Directionality of sender/receiver

In this section, we determine the influence in signal strength and link quality when changing the antenna. This is possible because the A80 supports two different antennas (see 2.1.2.1). In this measurement we send 5000 frames with a period of 120 ms at Target 80640419 which are received by five Targets. The duration of the measurement is 10 minutes.

The result when receiving with antenna 0 is shown in Figure 3-21.

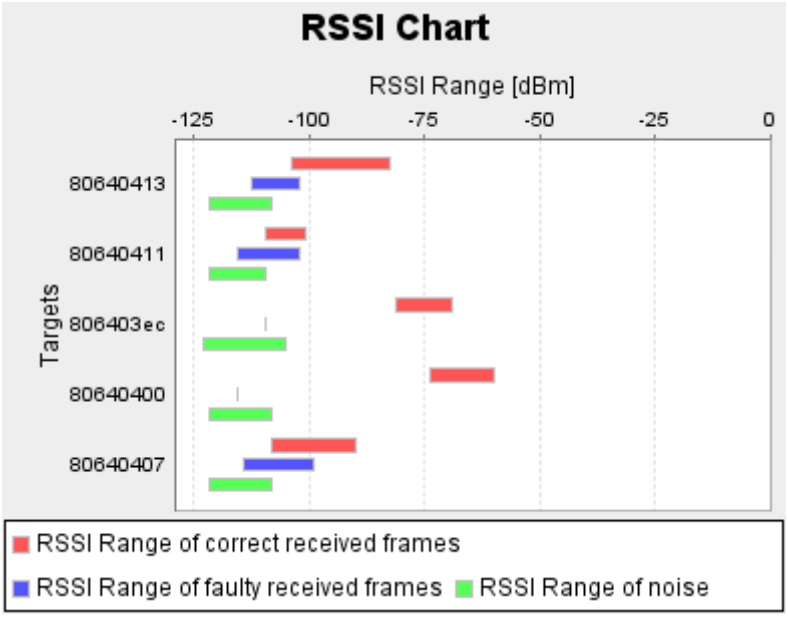

Figure 3-21: RSSI Chart Antenna 0

The result when receiving with antenna 1 is shown in Figure 3-22.

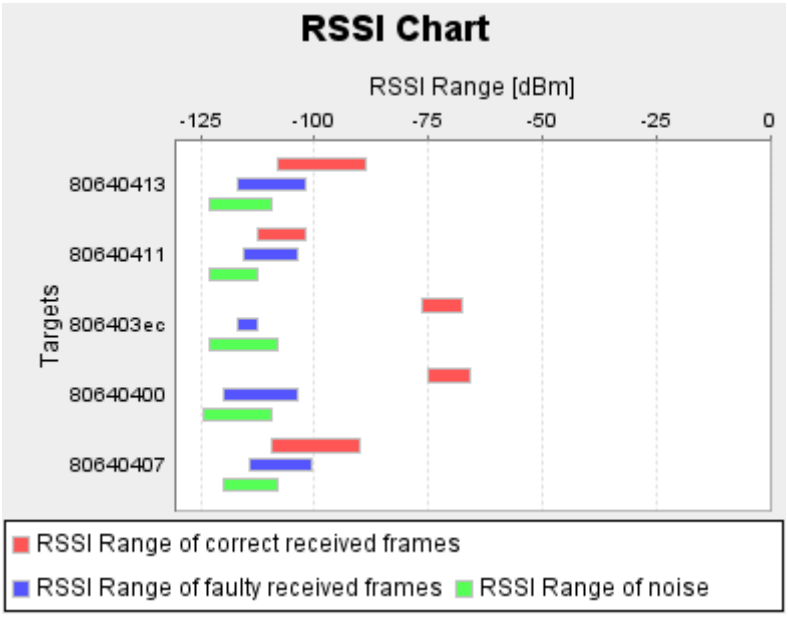

Figure 3-22: RSSI Chart Antenna 1

When we compare the results, we can see that we almost have the same signal strengths with antenna 0 and antenna 1. We just see that some links with antenna 0 have a maximum signal strength which is a bit larger than the maximum signal strength of the same link with antenna 1.

3.3.4 Link Quality at different Sending Power

In this section we determine the changes of link quality and signal strength when the sending power of the sending Target is altered.

In a first measurement we just run a set of measurements with 10000 iterations and a sending power of -2.06 dBm with the sender 80640419 with no threshold. The period is 120 ms and the duration is 20 minutes. The result is shown in Figure 3-23. If we compare it to results which are taken earlier with a sending power of 9.68 dBm (see Figure 3-24), we can see that the graph has shifted about 12 dBm to the left which is the change of the sending power. We can also see that fading has increased at Targets which are near to the sender (80640400). Some links, that already had a low RSSI range with 9.68 dBm, do receive nothing when sending with -2.06 dBm (see 80640411 which is placed at another floor).

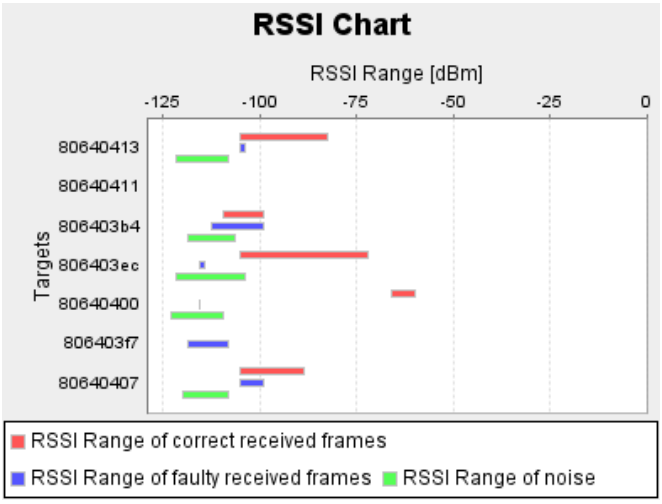

Figure 3-23: RSSI Chart Power -2.06 dBm

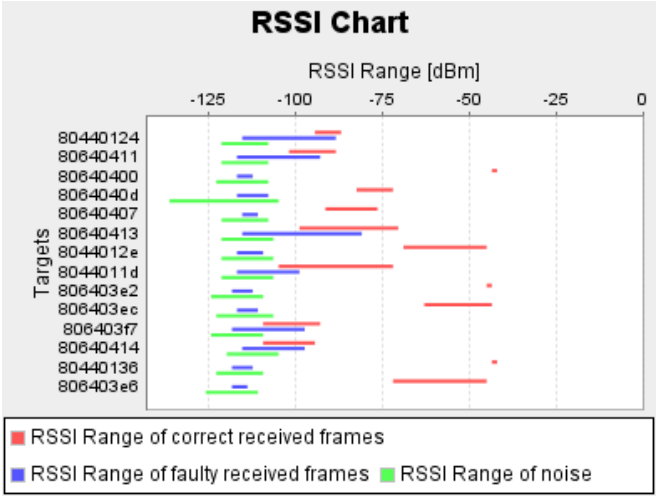

Figure 3-24: RSSI Chart Power 9.68 dBm

We have already seen in other measurements, that if the RSSI goes under a certain value, we will have more bit errors and missed frames (see Figure 3-8). Using the lower sending power setting we can determine this value more detailed because we also have Targets on the same floor which have an RSSI of less than -100 dBm.

In a second measurement with 1000 frames, a period of 1 s and a duration of 17 minutes, we undermine a link which is bad and has over 20 % erroneous frames. This link has a signal RSSI range between -111 dBm and -99 dBm and a noise RSSI range between -121.5 and -111 dBm. I.e. we have a SNR which is about 1. In the RSSI histogram (see Figure 3-25), we can see that if we go under a certain RSSI value we get more and more bit errors (see Table 3-4).

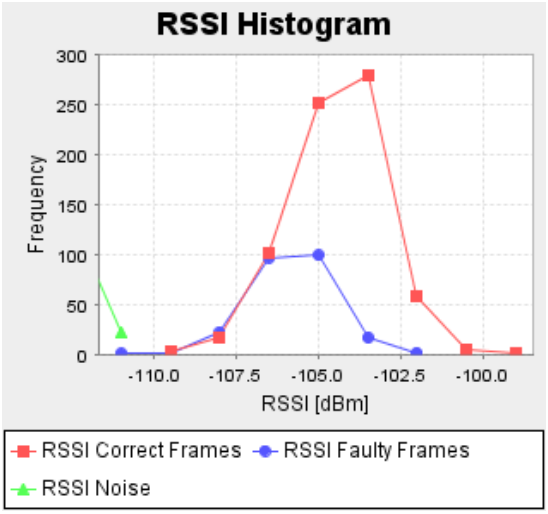

Figure 3-25: RSSI Histogram Bad Link

RSSI [dBm]	Distance to Noise Max [dBm]	Bit Error Probability
- 102	7	3 %
- 103.5	5.5	6 %
- 105	6	28 %
- 106.5	4.5	49 %
- 108	3	56 %

Table 3-4: Bit Error Probability

If we look at the histogram of bit errors per frame (see Figure 3-26), we can see that most of the faulty frames have one or two bit errors while only a few have a lot of bit errors.

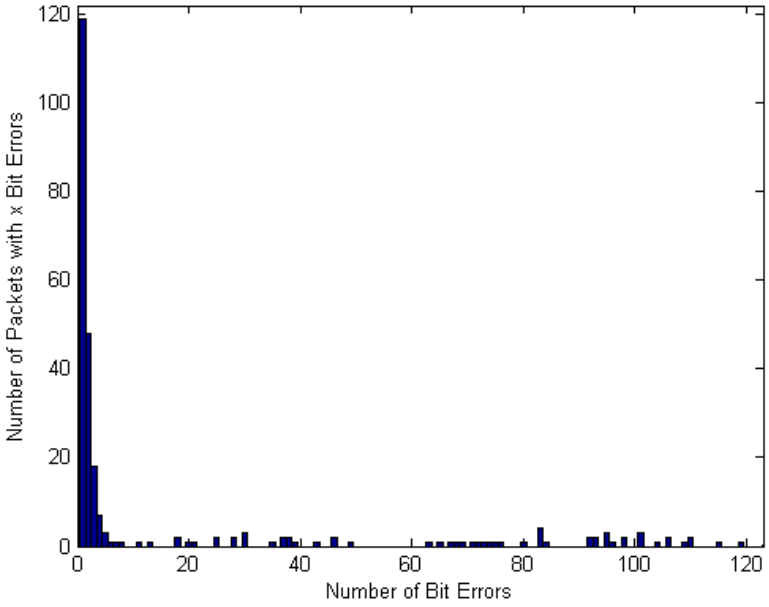

Figure 3-26: Bit Error Histogram

3.4 Discussion

In this section, we discuss the results from the measurements performed in 3.3.

In the measurement shown in 3.3.1.1 we can look up the distances and the RSSI values from the 80640400 and 806403f7 in order to compute an estimation for the path loss in an office building. Target 80640400 is 3.7 m away from the sender and has an RSSI of about -42 dBm. Target 806403ec has a distance to the sender of 51.4 m and an RSSI of about -102 dBm (Note that there are obstacles between the sender and Target 806403f7). This would result in a path loss of about 1.25 dBm per meter.

In the measurements performed in 3.3.1.2 we have proofed that the A80 Target works as intended. We can transmit 1500000 frames without errors. In other words, the bit errors and lost frames measured in our tests result from physical phenomena and not from internal hard- or software-problems.

The fact that we receive about 25 SFDs per hour out of the noise (see 3.3.1.3) does not imply that we also would receive 25 SFDs per hour if a sender is active because the receiver is not always listening to noise what reduces the probability of receiving the correct pattern out of the noise. However, we would receive several faulty frames per hour if a sender is active what vaguely corresponds to the number of missed frames in a link quality test between two Targets which are close together (see Figure 3-2).

In 3.3.1.5 we determined the difference of fading between day and night. We can clearly see that we have more fading if people are walking around in the office. But there are also places which are near the sender but have fading because they are located on special places with high fading. When we compare the PER and RSSI results at night, we can conclude that fading does not influence the PER as long it is above a certain RSSI value. We can say that fading influences the link quality (bit errors and missed frames) if it goes under a certain level which is near the noise level. Fading is therefore bad if we are already on a bad RSSI value.

In 3.3.3 we have measured the difference in signal strength when using antenna 0 and antenna 1. We found that the difference is very small, whereas antenna 0 is a bit better. This is probably a result of the Target orientation which is the same for all Targets in the building.

In 3.3.4 we have seen that if the signal comes close to the noise level, we get more and more bit errors. If we are more than 10 dBm over the maximum noise level, we get hardly any bit errors. The point where we begin to get bit errors and missed frames in our environment is under -102 dBm. We can also see that most of the erroneous frames have only one or two bit errors. This is a good property if we want to do FEC because most often only a few bits are corrupted.

Another reason for a bad link quality is the receiver sensitivity at -112 dBm. However, because we often have a maximum noise level which is in this region, the link quality frequently becomes bad even before it reaches the receiver sensitivity.

3.5 Conclusion

In this section we draw a conclusion of the measurements we performed.

Using the test in the EMC-Room, we proved the correct behavior of the Target hard- and software.

From the results of the measurements in this chapter, we can conclude that the link quality depends on signal strength and its distance to noise (including noise from interferers). We found that if we go under a certain RSSI, which is about 10 dBm over the maximum of the noise, we start to get bit errors and missed frames. Confirming [Reijers et al. 2004, 232], we have found that RSSI is a good estimator for link quality if it is over a certain level. If the RSSI is under that level, the number of bit errors and missed frames changes with every dBm which makes it very difficult to derive the link quality from the signal strength. As a general rule of thumb we can say that if the RSSI minimum of the signal is higher than 10 dBm over the noise we get hardly any bit errors or missed frames. If we are completely in the noise range, we do not receive any frames.

In 3.3.2 we analyzed the influence of interference to our communication. We can conclude that also devices which communicate over a different channel influence the link quality, especially if the channels are close together. We therefore have to try to keep as many channels as possible free around the sending channel. However, if a fire detector system is going life, it also has to sustain devices which communicate on a channel near to its own channel. Using these results, we can say that an algorithm, which changes the channel because of interference, should not take one of the six neighboring channels.

We have seen that the link quality does change both slow and fast over the time. When a link has fading, the signal can change between up to 20 dBm over seconds. Obviously it can also change more than 35 dBm over minutes or hours if the environment between sender and receiver is changing.

If we remember the influencing factors described in 3.1.2, we can say that the most important external factors are receiver location, fading and interference from other devices. The most important internal factor is obviously the sending power.

We have seen that most of the links we measured are either good or bad. Good means a PER of more than 99%, bad means a PER of less than 50 %. Bad links should not be included in the list of neighbors of a Target.

We have found that faulty frames often have only a few bit errors what contradicts the results of [Willig/Mitschke 2006] and makes it attractive for using FEC. However, because most of the links are either good or bad, we only have few links on which we would have to do error correction. For that reason, the overall cost of using FEC would be higher than the benefit because when using FEC, also good links would have to send more data. We therefore propose to use ARQ for reliable communication.

3.5.1 Further Work

We think that the following questions are meaningful to be answered if we want to get a deeper understanding of radio wave propagation in office buildings:

- How does the hardware perform in different buildings? In basements?
- Are there any other strong influencing factors than the one we found (micro-ovens, elevators)?

- Are links always symmetric in environments with no interferers? Under which conditions not?

4 Conclusion

In this thesis we first gave an introduction to WSN, the difficulties in prototyping them and the specific goals of the thesis. In the second chapter we have described the development of the DSNAnalyzer and the evaluation of the whole DSN. In the third chapter we described the measurements performed using the tool developed in the second chapter.

If we consider the three main goals described in 1.2.1, we can draw the following conclusion:

1. The task of integrating the existing DSN with the A80 by SBT was done successfully. We tested the hard- and software of the A80 and the Adapter Board, designed a protocol for communicating with the A80, tested the interaction of the DSN-Node and the A80 and proofed its correct behavior in a shielded environment.
2. We developed a tool which helps to overcome problems in prototyping and testing WSN. The development included requirements engineering, design, implementation and intensive testing. The tool is able to handle output from the Targets as well as from the simulator and proved its functionality by delivering all the data used for the WSN link quality measurements. The DSNAnalyzer also supported the alpha testers by developing and evaluating their protocols. Additionally we evaluated the DSN itself and showed under which conditions it can be used efficiently.
3. We designed and implemented a test environment with a counterpart on the Target. The test engine which is integrated in the DSNAnalyzer allows executing test scripts that can control all the functionality of the target. The test suite can be used to schedule and automate the execution of tests over several days with different parameters. Using this test environment we were able to analyze the behavior of radio wave propagation in our office building. Additionally, we tested and improved the physical layer of the Target itself.

The industry is mainly interested in information about radio wave behavior which helps to improve protocol design. However, in order to get this information we need a sophisticated toolset. The DSN and its new client *DSNAnalyzer* is such a toolset which proofed its functionality in several manners and also allows analyzing protocols which are already developed.

4.1 Experiences / Lessons Learned

Here I provide a list of experiences which describe what I would do differently in further projects:

- Because I wanted the whole DSN system at SBT to run stable as fast as possible, we often asked for quick software updates. Unfortunately, this software was often not sufficiently tested and introduced new bugs instead of solving the existing ones. In a next project, I would give other system developers enough time to check if the updated software works as intended.

- In order to save time, I often merged the deployment of several new software features into one download. Because of that, I often didnt know which part of the new software is responsible for a newly discovered bug.
- Because this thesis is part of a large project which includes people from multiple locations, the collaboration was hard work. I have learned that all participants need detailed and well-structured interfaces and requirement-documents in order to allow a clean integration.

Bibliography

[Andel/Yasinsac 2006, 48]
Andel, T.R.; Yasinac, A.: *On the Credibility of Manet Simulations*. IEEE Computer, 10.1109/MC.2006.242, 2006

[Ariadne 2001]
Ariadne Training Limited: *UML Applied – Object Oriented Analysis and Design using the UML*. http://ekalavya.iitb.ac.in/rhs/tutorials/UMLApplied.pdf (2001)

[Beutel et al. 2003]
Beutel, J.; Kasten, O.; Ringwald, M.: *BTnodes – A Distributed Platform for sensor nodes*. 1st ACM Conf. Embedded Networked Sensor Systems (SenSys), ACM Press, New York, pp. 292-293, 2003

[Beutel et al. 2004]
Beutel, J.; Dyer, M.; Meier, L.; Ringwald, M.; Thiele, L.: *Next-Generation Deployment Support for Sensor Networks*. TIK-Report No: 207, Computer Engineering and Networks Lab, Swiss Federal Institute of Technology (ETH) Zurich, http://www.vs.inf.ethz.ch/publ/papers/tik207.pdf (2004)

[BTnode 2006]
BTnode: *BTnode – A Distributed Environment for Prototyping Ad Hoc Networks*. http://www.btnode.ethz.ch (2006)

[Cerpa et al. 2005]
Cerpa, A.; Wong, J.L.; Potkonjak, M.; Estrin, D.: *Temporal Properties of Low Power Wireless Links: Modeling and Implications on Multi-Hop Routing*. Center for Embedded Network Sensing, University of California, Los Angeles, http://lecs.cs.ucla.edu/~cerpa/papers/time-estimation-tech-report.pdf (2005)

[Chipcon 2006]
Chipcon AS: *SmartRF CC1020 Datasheet*. http://www.chipcon.com/files/CC1020_Data_Sheet_1_8.pdf (2006)

[Chlipala et al. 2003]
Chlipala, A.; Hui, J.; Tolle, G.: *Deluge: Data dissemination for network programming at scale*. Computer Science Division, University of Berkeley, http://www.cs.berkeley.edu/~jwhui/research/deluge/cs262/cs262a-report.pdf (2003)

[Culler/Estrin 2004]
Culler, D.; Estrin, D.: *Overview of Sensor Networks*. IEEE Computer Society, http://csdl2.computer.org/comp/mags/co/2004/08/r8041.pdf (2004)

[Deacon 2005]
Deacon, J.: *Model-View-Controller (MVC) Architecture*. http://www.jdl.co.uk/briefings/MVC.pdf (2005)

[Fowler 2003]
Fowler, M.: *UML Distilled – Third Edition – A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley, 2003

[GloMoSim 2006]
GloMoSim: *GloMoSim, Global Mobile Information Systems Simulation Library*. http://pcl.cs.ucla.edu/projects/glomosim (2006)

[Hammel 2006]
Hammel, T.: *Extreme Java GUI Testing*. http://www.developer.com/java/other/article.php/1016841 (2006)

[Hobi/Winterhalter 2005]
Hobi, D.; Winterhalter, L.: *Large-Scale Bluetooth Sensor-Network Demonstrator*. Master Thesis, ETH Zurich, http://www.btnode.ethz.ch/pub/uploads/Projects/jaws_MA_2005ss_hobiwinterhalter.pdf (2005)

[JSON-RPC 2006]
JSON-RPC: *JSON-RPC Specifications*. http://json-rpc.org/wiki/specification (2006)

[Leffingwell/Widrig 1999, 16]
Leffingwell, D.; Widrig, D.: *Managing Software Requirements – A Unified Approach*. Addison Wesley, 1999

[NGMN 2006]
Next Generation Mobile Network: *Fast Fading*. http://www.ngmn.de/index.php?s=lex (2006)

[Nguyen/Thiran 2006]
Nguyen, H.X.; Thiran, P.: *Using End-to-End Data to Infer Lossy Links in Sensor Networks*. School of Computer and Communications Sciences, EPFL, http://infoscience.epfl.ch/getfile.py?recid=85857&mode=best (2006)

[Martin 2000]
Martin, R.C.: *Design Principles and Design Patterns*. http://www.objectmentor.com/resources/articles/Principles_and_Patterns.PDF (2000)

[McKinney 2006]
McKinney: *Regular Expression Quick Reference v1.00*. http://gmckinney.info/resources/regex.pdf (2006)

[MoteLab 2006]
MoteLab: *MoteLab: Harvard Network Sensor Testbed*. http://motelab.eecs.harvard.edu (2006)

[Nuevo 2004]
Nuevo, J.: *A Comprehensible GloMoSim Tutorial*. Université du Québec, http://externe.emt.inrs.ca/users/nuevo/glomoman.pdf (2004)

[Levis et al. 2004]
Levis, P.; Patel, N.; Culler, D.; Shenker, S.: *Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks*. Networked System Design and Implementation (NSDI), New York, http://csl.stanford.edu/~pal/pubs/trickle-nsdi04.pdf (2004)

[Lewis 2004]
Lewis, F. L.: *Wireless Sensor Networks*. Advanced Controls, Sensors, and MEMS Group,

Automation and Robotics Research Institute, The University of Texas at Arlington, http://arri.uta.edu/acs/networks/WirelessSensorNetChap04.pdf (2004)

[Radio Electronics 2006]
Radio Electronics: *Electromagnetic Waves – Reflection, Refraction and Diffraction.* http://www.radio-electronics.com/info/propagation/em_wave_reflection/reflection_refraction_diffraction.php (2006)

[Reijers et al. 2004]
Reijers, N.; Halkes, G.; Langendoen, K.: *Link Layer Measurements in Sensor Networks*. Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, http://www.st.ewi.tudelft.nl/~koen/papers/mass04.pdf (2004)

[Siemens 2006]
Siemens: *Application for funding (Safety Critical Sensor Networks for Building Applications)*. KTI/CTI Application Document (Siemens Internal), 2006

[Son et al. 2005]
Son, D.; Krishnamachari, B.; Heidemann, J.*: Experimental Analysis of Concurrent Packet Transmissions in Low-Power Wireless Networks*. Viterbi School of Engineering, University of Southern California, http://www-scf.usc.edu/~dongjins/paper/ISI-TR609-SonKrishnamachariHeidemann.pdf (2005)

[Sun 2006]
Sun: *Java Look and Feed Design Guidelines*. http://java.sun.com/products/jlf/ed2/book/ (2006)

[Sun 1997]
Sun Microsystems: *Java Code Conventions*. Mountain View, California http://java.sun.com/docs/codeconv/CodeConventions.pdf (1997)

[Thelen et al. 2005]
Thelen, J.; Goense, D.: Langendoen, K.: *Radio Wave Propagation in Potato Fields.* First Workshop on Wireless Network Measurements, http://www.winmee.org/2005/papers/WiNMee_Thelen.pdf (2005)

[Wikipedia 2006A]
Wikipedia: *Radio Propagation*. http://en.wikipedia.org/wiki/Radio_propagation (2006)

[Wikipedia 2006B]
Wikipedia: *Bit Error Ratio*. http://en.wikipedia.org/wiki/Bit_error_ratio (2006)

[Willig/Mitschke 2006]
Willig, A.; Mitschke, R.: *Results of Bit Error Measurements with Sensor Nodes and Casuistic Consequences for Design of Energy-Efficient Error Control Schemes.* EWSN, Zürich, http://www.tkn.tu-berlin.de/publications/papers/ewsn06-webout.pdf (2006)

[Woo/Culler 2002]
Woo, A.; Culler, D.: *Evaluation of Efficient Link Reliability Estimators for Low-Power Wireless Networks*. UC Berkeley Technical Report, http://www.cs.berkeley.edu/~awoo/awooLinkTR.pdf (2002)

[XML-RPC 2006]
XML-RPC: *XML-RPC Homepage*. http://www.xmlrpc.com (2006)

[Zimmermann 2005]
Zimmermann, S.C.: *Online Sensor-Network Monitoring*. Semesterarbeit, ETH Zürich, http://www.btnode.ethz.ch/pub/uploads/Projects/jaws_SA_2005ss_zimmermann.pdf (2005)

[Zhao/Govindan 2003]
Zhao, J.; Govindan, R.: *Understanding Packet Delivery Performance In Dense Wireless Sensor Networks*. SenSys, Los Angeles, http://pdos.csail.mit.edu/decouto/papers/zhao03.pdf (2003)

[Zhou et al. 2004]
Zhou, G.; He, T.; Krishnamurthy, J.; Stankovic, J.A.: *Impact of Radio Irregularity on Wireless Sensor Networks*. Department of Computer Science, University of Virginia, Charlottesville, http://www.cs.virginia.edu/papers/radio_irregularity.pdf (2004)

[Zuniga/Krishnamachari 2005]
Zuniga, M.; Krishnamachari, B.: *Link LayerModels for Wireless Sensor Networks*. The Autonomous Networks Research Group, University of Southern California, http://ceng.usc.edu/~anrg/downloads/LinkModellingTutorial.pdf (2005)

[Zuniga/Krishnamachari 2004]
Zuniga, M.; Krishnamachari, B.: *Analyzing the Transitional Region in Low Power Wireless Links*. Department of Electrical Engineering, UCLA, IEEE SECON, http://ceng.usc.edu/~bkrishna/research/papers/channelmodellingSECON04.pdf (2004)

Appendix A – Project Plan

Vorgangsname	Anfang	Ende
Setup Adapterboard	Di 28.03.06	Mo 10.04.06
Inception Phase	**Mo 27.03.06**	**Fr 14.04.06**
Requirement Analalysis GUI	Mo 27.03.06	Fr 14.04.06
Determine System Architecture (Split of work)	Mo 10.04.06	Fr 14.04.06
Elaboration Phase	**Mo 17.04.06**	**Fr 28.04.06**
Simple Use Cases Analysis	Mo 17.04.06	Fr 28.04.06
Prototyping	Mo 17.04.06	Fr 28.04.06
Class Analysis	Mo 17.04.06	Fr 28.04.06
Construction Phase	**Mo 01.05.06**	**Fr 28.07.06**
Iteration 1	**Mo 01.05.06**	**Fr 12.05.06**
Use Case: View Trace	Mo 01.05.06	Fr 12.05.06
Use Case: Execute Command	Mo 01.05.06	Fr 12.05.06
Use Case: Open Terminal	Mo 01.05.06	Fr 12.05.06
Iteration 2	**Mo 15.05.06**	**Fr 16.06.06**
Use Case: View Trace	Mo 15.05.06	Fr 16.06.06
Use Case: Program Targets	Mo 15.05.06	Fr 16.06.06
Iteration 3	**Mo 19.06.06**	**Fr 30.06.06**
TestEngine	Mo 19.06.06	Fr 30.06.06
Midterm Presentation ETH	Mi 21.06.06	Mi 21.06.06
Iteration 4	**Mo 17.07.06**	**Fr 28.07.06**
Test Analysis	Mo 17.07.06	Fr 28.07.06
Transition Phase	**Mo 31.07.06**	**Mi 02.08.06**
Presentation of the system to Siemens users	Mo 31.07.06	Mi 02.08.06
WSN connectivity measurements	**Do 03.08.06**	**Do 14.09.06**
Define / Evaluate Metrics, Link Quality Test	Do 03.08.06	Mi 16.08.06
Check influencing factors	Do 03.08.06	Mi 16.08.06
Measurements	Do 03.08.06	Mi 06.09.06
DSN Bugfixing, Updates	Do 03.08.06	Do 07.09.06
Analysis, Key Questions	Do 07.09.06	Do 14.09.06
Documentation	Di 08.08.06	Di 19.09.06
Documentation Review / Corrections	Mi 20.09.06	Fr 22.09.06
Prepare for Presentation	Mo 18.09.06	Mo 25.09.06
Presentation	Di 26.09.06	Di 26.09.06

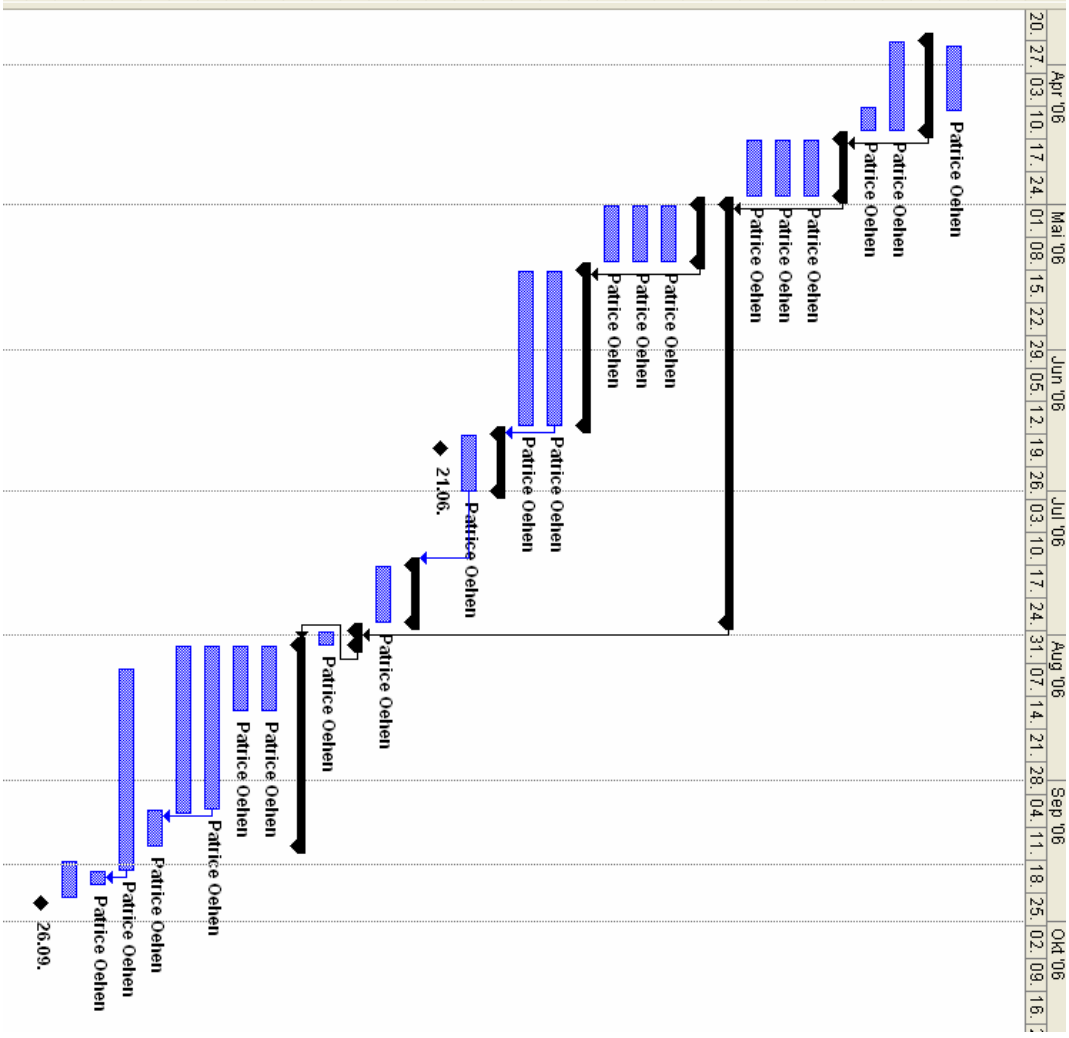

Figure 4-1: Project Plan Master Thesis

Appendix B –Definitions

Notation

Symbol	Definition	Example
(…)	Grouping	(ab)* = ab, abab, abababab
[…]	Any charter between the brackets	[EWID] = E, W, I, D
[^…]	Any character not between the brackets	[^\n] = every character but no newline
*	Zero of more occurrences	1* = nothing, 1, 1111
+	One or more occurrences	1+ = 1,111
\s	Whitespace	the\shouse = the house
\n	Newline	-
[:alnum:]	Any letter or digit	[:alnum:] = a, b, 1, 5
[:digit:]	Any digit	[:digit:] = 1, 3, 6
[:xdigit:]	Any hex number	[:xdigit:] = 1, 2, 9, a, f
x{n}	Exactly n times y	a{3} = aaa
x?	Optional	x? = nothing, x
<x>	The symbol x	If <x> is a regular expression, there is somewhere a definition of this symbol.
A | B	Alternation (Either the one or the other)	A | B = A, B

Table 4-1: Notation [McKinney 2006]

Message Definition

Message	Definition	Description
<command>	[^\n]*\n	The command that the DSN-Node has received from the GUI-Node.
<reply>	I\s[^\n]*\n	A special kind of an event which have their own log class[64] which means that they are immediately sent back to the DSN-Server.
<event>	[EWID]\s[^\n]*\n	Stored at the DSN-Node until it is requested from the DSN-Server.

Table 4-2: Message Definition

Network File Definition

[64] Every log on the DSN-Node belongs to a log class which describes the type of the log.

Message	Definition	Description
<network_file>	(<target_entry>\n)*	-
<target_entry>	<target_id>\s0\s(<position>)\s<dsn_node>?	<dsn_node> is the ID of the DSN-Node connected to the target with ID <target_id>
<target_id>	[:alnum:]+	e.g. 80440334
<position>	<pos_comp>,\s<pos_comp>,\s<pos_comp>	e.g. 240, 34.3, 4
<pos_comp>	[:digit:]*(.[:digit:]*)?	e.g. 230.7
<dsn_node>	<dsn_node_id><description>?	-
<dsn_node_id>	[:xdigit:]{2}(:[:xdigit:]{2}){5}	e.g. 00:04:3F:00:01:57
<description>	[^\n]*	e.g. Office Oehen Desktop

Table 4-3: Network File Definition

Test Definition

Message	Definition	Description
<test>	(<test_element>\n)*	The whole test.
<test_element>	<target_command> | <result_file> | <wait>	There are 4 different types of commands.
<target_command>	<target_id>\s<command>(<param>\s)*	The default test element.
<result_file>	%tracefile=[:alnum:]*	When this kind of command appears in a test file the results of the test are requested after execution and stored into a file which has the specified name as the second part of the filename.
<wait>	%wait=[:digit:]+	Wait for a specified number of milliseconds.
<target_id>	all | [:alnum:]+	“all” means all Targets, otherwise the ID of one specific Target is specified.
<param>	[:alnum:]*	A parameter of a Target command.

Table 4-4: Test Definition

Event Definition

Message	Definition
<tracefile>	<event>(\n<event>)*
<event>	<target_id>\s<log_level><log_layer>\s<log_aspect><description>
<target_id>	[:alnum:]+
<log_level>	[ewid]
<log_layer>	[spmnta]
<log_aspect>	[:alpha:]*
<description>	<general_desc>|<sender_desc>|<receiver_desc>| <test_parameters>|<test_result>|<frame_result>
<general_desc>	[:alnum:]*
<sender_desc>	[:alnum:]*\sid=<message_id>(\s[:alnum:]*)+
<receiver_desc>	[:alnum:]*\sid=<message_id>\ssource=<source_id>(\s[:alnum:]*)+
<test_parameters>	See Table 4-6
<test_result>	See Table 4-6
<frame_result>	See Table 4-6
<source_id>	[:alnum:]+
<message_id>	[:alnum:]+

Table 4-5: Event Definition

Test Result Definition

Message[65]	Definition	Description
<test_parameters>	{<tsn>,<channel>,<power>, <iteration>,<rx_antenna>, <preamble_length>, <transmit_period>, <radio_set>, <rssi_threshold>}	A special type of an event description and includes information about the parameters of a test.
<test_result>	{<tsn>,<cfr>,<ffr>(,<macfr>)? (,<micfr>)?(,<marffr>)?(,<mirffr>)? (,<marn>)(,<mirn>)}	The result of a test. RSSI values are only included if there were correct resp. faulty frames.
<frame_result>	{<tsn>,<fn>,<be>,<rs>,<rn>(,rxd)?}	The result of a frame reception. <rxd> is only included if there were errors.
<tsn>	"tsn":[:alnum:]*	The ID of the sender.
<channel>	"Channel":[:alnum:]*	The channel on which was sent.
<power>	"Power":[:alnum:]*	The power used to send.
<iteration>	"Iteration":[:alnum:]*	The number of iterations.
<rx_antenna>	"RxAntenna":[:alnum:]*	The antenna used.
<preamble_length>	"PreambleLength":[:alnum:]*	The number of 2-byte tuple in the preamble.

[65] Note that this format follows the JSON-RPC standard.

<transmit_period>	"TransmitPeriod":[:alnum:]*	The time between two frames.
<radio_set>	"RadioSet":[:alnum:]*	A special setting for frequency.
<rssi_threshold>	"RssiThreshold":[:alnum:]*	The minimum signal strength.
<crf>	“CFr:”[:alnum:]*	The number of correct frames.
<ffr>	“FFR:” [:alnum:]*	The number of faulty frames.
<macfr>	“MaRCFr”:[:alnum:]*	The maximum RSSI of the correct frames.
<micfr>	“MiRCFr”:[:alnum:]*	The minimum RSSI of the correct frames.
<marffr>	“MaRFFr”:[:alnum:]*	The maximum RSSI of faulty frames.
<mirffr>	“MiRFFr”:[:alnum:]*	The minimum RSSI of faulty frames.
<marn>	“MaRN”:[:alnum:]*	The maximum RSSI of the noise.
<mirn>	“MiRN”:[:alnum:]*	The minimum RSSI of the noise.
<fn>	“fn”:[:alnum:]*	The number of the frame.
<be>	“be”:[:alnum:]*	The number of bit errors in the frame.
<rs>	“rs”:[:alnum:]*	The RSSI of the signal.
<rn>	“rn”:[:alnum:]*	The RSSI of the noise.
<rxd>	“rxd”:[:alnum:]*	The received data in hex.

Table 4-6: Test Result Definition

Appendix C – DSNAnalyzer Readme

README

DSNAnalyzer V.1.2a

Author: Oehen, Patrice (patrice.oehen.ext@siemens.com)

0. Introduction

****

For the release notes please look at the file CHANGELOG.txt

Please note that the current release of DSNAnalyzer is in alpha status and does neither provide all the necessary functionality nor is proofed to be free of bugs.

If you find anything that does not work as intended please send an email to patrice.oehen.ext@siemens.com

The DSNAnalyzer includes a property file which is located in /etc/dsnanalyzer.properties. This file can be used to configure properties according to the network file, filters, DSN-Server, view and paths.

1. Installation & Build

****

The following two sections describe installation and the build process.

1.1 Installation

****

1. Check if JRE 1.5.0 is installed: (Note: On Siemens machines this could be done by checking: Start->Utilities->CAT Tools->Java 2 Runtime->Java Switcher)

2. Download Java 3D Run Time Environment from (http://java.sun.com/products/java-media/3D/download.html) and install it into your Java home (e.g. C:\Program Files\Java\JRE1.5.0_06). This installation places some jars into the /lib/ext folder of Java home.

3. If you are using DSNAnalyzer as stand-alone, i.e. without the DSN Server, you have to check the property "serverURL" in etc/properties.conf is commented out.

4. Double-click onto "DSNAnalyzer.jar". If this does not work, use Start -> Execute -> type "cmd" to open a "Command Window", change directory to where the DSNAnalyzer.jar file is located and type: "java -jar DSNAnalyzer.jar". If you are working with very large trace files (i.e. up to several MBs) use: "java -Xmx256m -jar DSNAnalyzer.jar" and specify enough memory.

Note: If you want to use the DSN-Server, ensure that the user "dsnanalyzer" is added. For more information please look at the DSN-Server documentation.

1.2 Ant Build

****

For building the jar files out of the source files perform the following steps:

1. Change to the root of the DSNAnalyzer project directory

2. Execute "ant" to build the jar (there is also a zip file created which resists in the dist directory, this file includes all the necessarily files to run the DSNAnalyzer on a different place than directly at the root of the DSNAnalyzer project)

1.3 Logging

****

The logging can be controlled with the file etc/logging.conf which configures log4j.

Note: The library WilmaScope does not depend on this logging properties but does logging using System.out.prinln(). There are also debug outputs enables because the application of this library is not without errors up to now.

2. Tutorial

****

The following three sections describe how to use the DSNAnalyzer.

Note that the positions of the Targets in the Analyze View and the Control View come from the .in network file which is located in /data.

2.1 Analyze View

****

To check the application functionality, you can do the following steps:

1. Run the application as mentioned in the installation part above or use "java -jar DSNAnalyzer.jar FILE" to directly open the trace file with filename FILE.

   e.g. java -jar DSNAnalyzer.jar "data/test_events.out"

2. Change to the "Analyze" tab (Now the targets are displayed in the "Map" according to their position listed in the file which is defined as "networkFilename" in the property file). The Property file is located at "etc/properties.conf".

3. Use "File -> Import Tracefile" to import a trace file (e.g. "test_event.out" which is located in /data)

   NOTE:

   The color of the message arrow is

   - Yellow if the message has log layer PHY

   - Blue if the message has log layer MAC

   - Red message has log layer NETWORK

4. Use the zoom functionality to change the time which should be displayed (Enter a number in the zoom text field, select a unit and click "Enter"). Note: The "Map" is controlled by the sequence chart, i.e., all events and messages between them which are visible in the "Sequence Chart" are also displayed in the "Map". E.g. Enter 1 and select "Second", now press "Enter". The vertical "Sequence Chart" scrollbar also controls the scrollbar of the "Log".

5. Further functionality:

   - You can click on events to highlight them in all three windows (Hold "CTRL" pressed to highlight multiple events).

   - Open "View -> Define Filter" to specify filter properties. E.g. Select the IDs of targets from which events should be displayed or disable the drawing of events which do not match the specified log properties.

   - Use "File -> Export Sequence Chart" to store the current sequence chart into a jpeg file.

   - Use "File -> Export Event" to store the content of the log into a text file.

- Right-click on the ID of a target (the boxes on the top of the sequence chart) and select "Hide Target" to put a target to the excluded list (like in the filter).

There are two ways to color events in the sequence chart using the property "coloring" in the property file:

1. If "coloring=layer" the events are colored:
   - Red if the error has log level ERROR
   - Yellow if the log level is WARNING
   - Green if the log level is INFORMATIONAL
   - Blue if the log level is DEBUG.
2. If "coloring=aspect" the events are colored according to the first character of the log aspect.
   - y for yellow
   - m for magenta
   - c for cyan
   - r for red
   - g for green
   - b for blue
   - w for white
   - k for black

2.2 Control View

****

Note: The Control View is only displayed if the property "serverURL" in the property file is set.

The Control View displays all the Target which are active, i.e. all the Target which have a running DSN-Node connected to it. This View can be used to do the following operations for all Target:

- Program Targets with new Software (includes flashing)
- Flash Targets
- Reset Targets
- Program DSN-Nodes with new Software (does not include reset)
- Reset DSN-Nodes
- Run Tests

- Run Multiple Test
- Use the Test Suite to have sophisticated test scheduling

Additionally, the user can control one or a set of Targets by selecting them with a left-click and opening the menu with the right-click.

Important things that are run in the Control View are logging in the log window.

2.3 Analyze Test View

****

In this view the user can import a result of a test previously ran in the Control View. In the Network Analysis part, the sending Target can be selected. In the Link Analysis, an arbitrary link can be selected.

"Import Extern Test Result" can be used to import the direct output of the A80 which does not have a timestamp as prefix.

"Export Test Result" can be used to export the result for all links into a format which is easy parsable by Matlab.

3. File Formats

****

Here we describe the file formats.

3.1 Trace-File Format

****

A trace file is the file that consists of events generated e.g. by a simulator and usually has the suffix ".out". A trace file has the following format which is specified in a mix of EBNF and regular expressions.

(Caution: "\n" means here exactly one whitespace)

<tracefile> = <event>(\n<event>)*

<event> = <target_id>\s<log_level>\n<log_layer>\s<log_aspect>\s<description>

<target_id> = [:alnum:]+

<log_level> = [ewid]

<log_layer> = [spmnta]

<log_aspect> = [:alpha:]*

<description> = <general_desc>|<sender_desc>|<receiver_desc>

<general_desc> = [:alnum:]*

<sender_desc> = [:alnum:]*\sid=<message_id>\s[:alnum:]*

<receiver_desc> = [:alnum:]*\sid=<message_id>\ssource=<source_id>(\s[:alnum:]*)+

<source_id> = [:alnum:]+

<message_id> = [:alnum:]+

Comments:

<log_level>: e: error, w: warning, i: informational, d: debug

<log_layer>: s: system, p: physical, m: mac, n: network, t: transport, a: application

<log_aspect>: any number of characters defining properties of the event

In the Analyze View, an arrow is drawn for every event pair which fulfills the following properties:

(A is the event that results from sending a message, B is the one that results from receiving a message)

- Event A has a description of type <sender_desc> and event B has a description of t ype <receiver_desc>
- The <message_id> of both events are equal
- The <source_id> of B and the <target_id> of A are equal

3.2 Network-File Format

****

The network file is used for displaying the Targets in the Control View and to compute the distance between two Targets in the Analyze Test View. The network file can be configured in etc/dsnanalyzer.properties and has usually the ending .in.

<network_file> = (<target_entry>\n)*

<target_entry> = <target_id>\s0\s(<position>)\s<dsn_node>?

<target_id> = [:alnum:]+

<position> = <pos_comp>,\s<pos_comp>,\s<pos_comp>

<pos_comp> = [:digit:]*(.[:digit:]*)?

<dsn_node> = <dsn_node_id><description>?

<dsn_node_id> = [:xdigit:]{2}(:[:xdigit:]{2}){5}

<description> = [^\n]*

Appendix D – Target Locations

Here we show pictures of the following Targets which are also shown in Figure 3-1: 806403f7 (see Figure 4-2), 80640124 (see Figure 4-3), 8064 0413 (see Figure 4-4), 80640407 (see Figure 4-5), 80640400 (see Figure 4-6), 80640136 (see Figure 4-7), 806403ec (see Figure 4-8) and 806403d8 (see Figure 4-9).

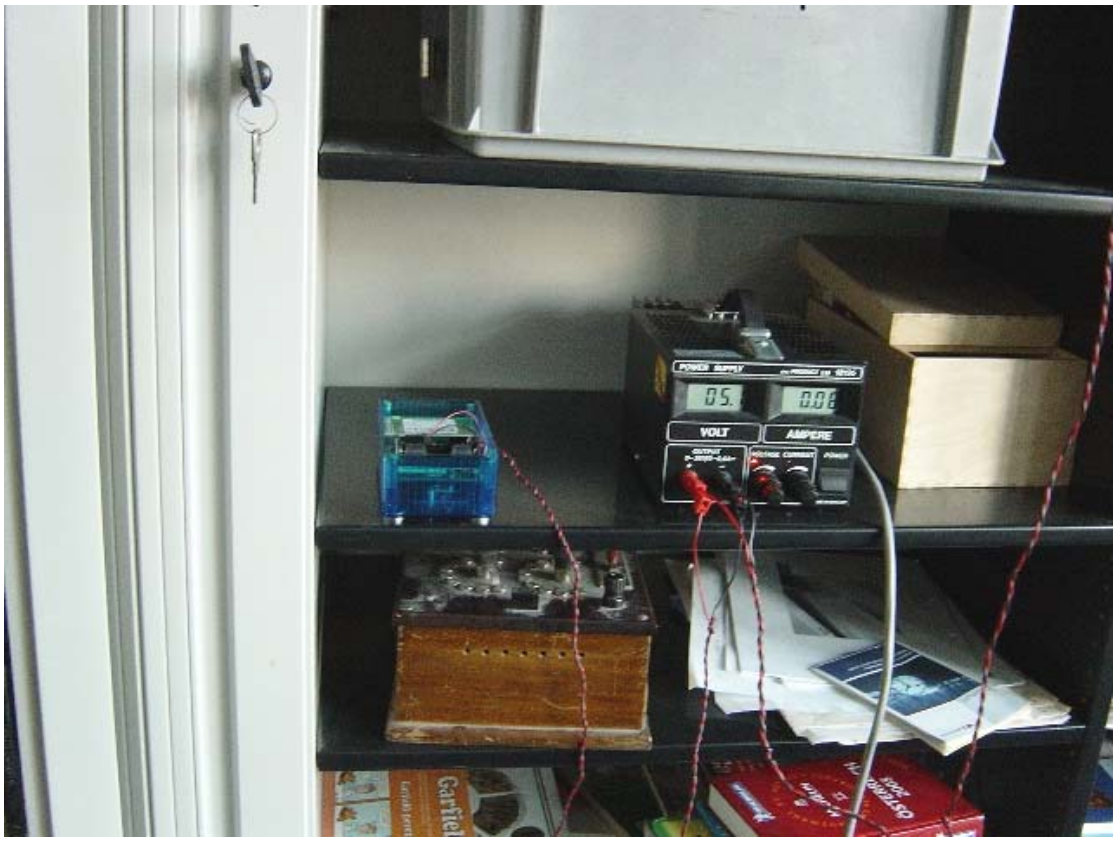

Figure 4-2: Target 806403f7

Figure 4-3: Target 80640124

Figure 4-4: Target 80640413

Figure 4-5: Target 80640407

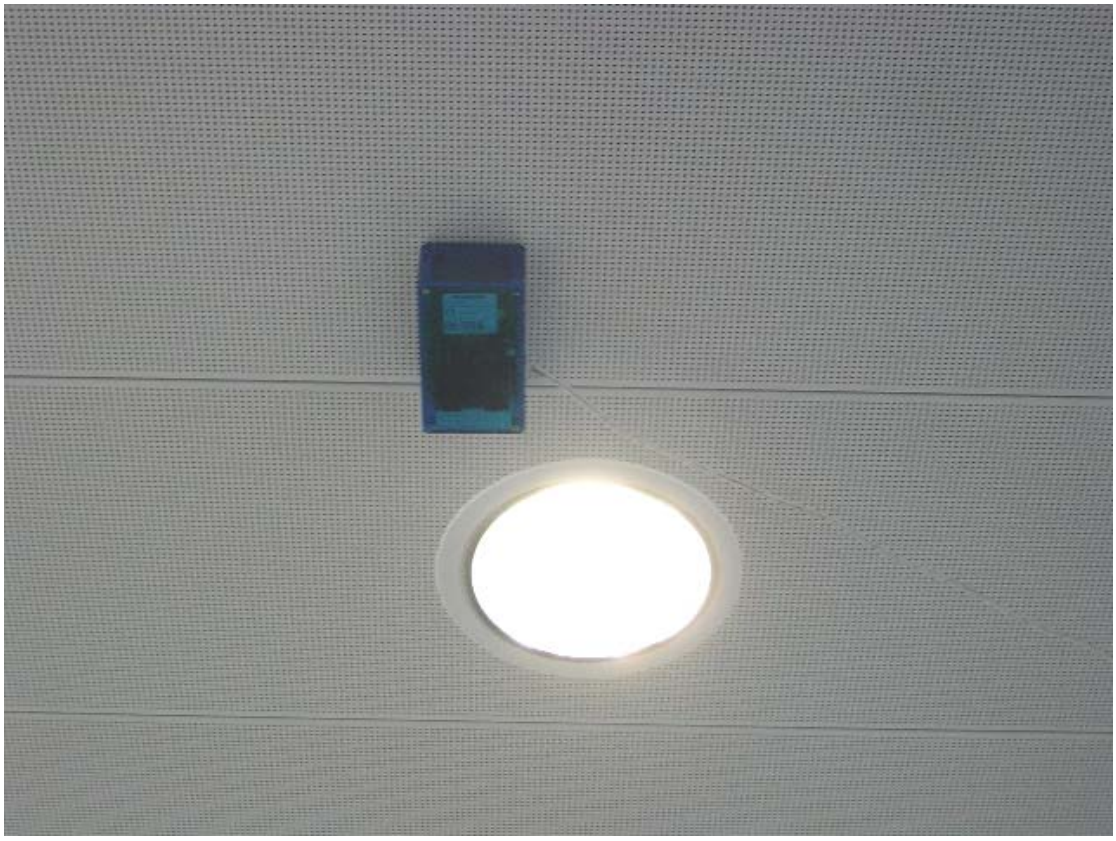

Figure 4-6: Target 80640400

Figure 4-7: Target 80440136

Figure 4-8: Target 806403ec

Figure 4-9: Target 806403d8