Degree project

Improvement of the Architecture and Communication Protocol of a Sensor-based IT System

Author: Illia Klimov and Oryna Podoba
Supervisor: Rüdiger Lincke
External Supervisor: Per-Olov Thorén
Date: 2013-12-09
Course Code: 5DV00E, 30 credits
Level: Master
Department of Computer Science
Acknowledgements
First of all we would like to thank our supervisor Dr. Rüdiger Lincke for his time and expertise. Then, we would also like to thank our customers who have provided us with an interesting topic, Maksym Ponomarenko who supervised the detailed development of the web applications, as well as Ruslan Gederin and Victor Mazepa who worked on developing the primary version of the system our efforts were building on.
Abstract
Communication protocols based on SMS and e-mail are simple and straightforward to implement yet they possess a significant communication overhead and delay. In a large sensor network this would increasingly lead to performance issues and decrease scalability of the overall system, since SMS and e-mail include a significant transmission delay and overhead. Additionally, a communication architecture, where clients communicate directly to the central server requires these to be online all the time, thus decreasing battery time.

We replaced the initial SMS and e-mail based communication protocol with a TCP based protocol. Furthermore, we selected a new architecture where sensor groups could communicate to a master node in that group, which in turn communicates to the central server.

These changes were implemented in an emulator, since the real sensors could not be reconfigured. We evaluated the improvements and could show that we can reduce the communication overhead and transmission delay as well as that the average battery time for all sensors in a group is increased.

Disclaimer: this paper is written under a non-disclosure agreement. This is why certain details are omitted in the thesis, but available to the project partner.

Key words: sensor, SMS, e-mail, TCP, communication protocols, architecture
6.6 Improvement protocol

6.5 Smartphone

6.4 Tablet

6.3 Web

6.2 Sensor Emulation

6.1.4 Checking Input Parameters

6.1.3 Receive Messages

5.3 Communication

5.2 Emulator A

5.1 Server Architecture

4.2.4 Summary

4.2.2 Individual Sensors

4.2.1 Sensors in Group with Master Node

4.1.2 Logic Clusters

4.1.1 Single Server

4.1 Sensor Communication Architecture

4.2 Sensor Communication Architecture

3.6 Summary

3.5 Requirements for Emulator

3.4.4 Hardware Problems

3.4.3 Emergency Situation Detected by Server

3.4.2 Emergency Situation Detected by Sensor

3.4.1 Data Acquisition

3.4 Use Cases

3.3 Messages

3.2 Types of Users

3.1 System Overview

3 Features & Requirements

2.9 SVN

2.7 Android

2.6 Android

2.5 Google Web Toolkit (GWT)

2.4 MySQL

2.3 Java Database Connectivity (JDBC)

2.2 MySQL

2.1 Netty Framework

2 Background

1.6 Project Constraints

1.5 Goals and Criteria

1.4 Problem and Motivation

1.3 Project Constraints

1.2 Goals and Criteria

1.1 Problem and Motivation

1. Introduction
7 Conclusions and Future Work ................................................................. 32
7.1 Conclusions .................................................................................. 32
7.2 Future Work ................................................................................. 32
References ......................................................................................... 33
Table List

Table 3.1: Users ................................................................. 6
Table 3.2: Messages ............................................................. 6
Table 3.3: Data acquisition scenario ........................................ 7
Table 3.4: Emergency situation detected by sensor scenario ........... 8
Table 3.5: Emergency situation detected by server scenario .......... 9
Table 3.6: Hardware problems scenario ...................................... 9
Table 5.1: Communication protocol ......................................... 17
Table 5.2: Protocol structure .................................................. 17
Table 6.1: Time for message sending ......................................... 30


1 Introduction
A company in south Sweden, which does not want to be mentioned at the time of the
thesis publication wants to propose new solutions in the sensor-based IT systems for
data acquisition and monitoring. Sensors are typically used for monitoring sensor values
and for reporting deviations from expected values.

Our work is performed in close collaboration with the customer, a second group of
students focusing on a different topic, as well as the experienced developers of
Softwerk, a consulting company managing our work.

Work on the project are distributed between students as following:
1. Oryna Podoba and Illia Klimov – Work with a study of the concept of the
   system and to identify weaknesses. The main task is not to develop new systems
   but improving existing ones. To do this, a lot of time was spent in close
   cooperation with the other group. The main points of attention that was put to
   this: Future communication protocols between sensors and server,
   persistence layer, alternative architectures, and mobile clients.
2. Ruslan Gederin and Viktor Mazepa – e-mail/SMS communication between
   sensors and server, SMS gateway, emulator of sensors.

1.1 Problem and Motivation
Currently the bespoken company develops a sensor-based IT system for continuous data
acquisition and monitoring (initial version). The system development is in an early
stage. Due to a limited budget and hard time-constraints it is not possible implement the
ideal solution right from the beginning. At the moment was created the initial version of
the system with basic functionality. One of the problems is the transformation of the
system with improvements. We need to increase the speed, minimize delays on the
transport layer and make the system scalable.

To influence evolution of the system under construction at an early stage, and to save
costs in the long run, the customer would like to know what alternative protocols and
system architectures might be suitable alternatives for the current (trivial) choices. Yet,
he wants to avoid implementing different alternatives, for comparison, to facilitate tests
and evaluations, right away due to the associated costs in particular regarding altering
the sensor design and implementation. Therefore an alternative low-cost solution is
required, and thus the problems to be solved by this thesis is to:
(a) propose an alternative architecture and communication protocol for the sensors,
(b) to implement an advanced prototype for evaluation, and
(c) to provide all necessary means to facilitate an evaluation of the proposed
   improvements.

This is a challenging problem to solve, since the current development of the sensor-
based IT system is not completed and all requirements are not known. Furthermore,
there is no complete testing/management infrastructure. It is impossible (due to budget
constraints) to create/program new hardware sensors according to new specifications.

1.2 Goals and Criteria
We define the following goals and criteria for solving the above problem:
1. The first goal is to evaluate different alternatives for the overall server
   architecture. This goal is reached if we describe and compare at least two
   different alternative architectures.
2. The second goal is to evaluate different alternatives for sensor organization. This
   goal is reached if we describe and compare three different alternative
   architectures/organizational patterns.
3. The third goal is to propose and implement an alternative communication protocol. This goal is reached if we describe and implement a protocol which will increase the speed of operation, system performance and reduce equipment costs.

4. The fourth goal is to implement and evaluate the most promising architecture and protocol alternative (advanced prototype) using an emulator and any required server components for facilitating the testing. And provide client application based on Android OS.

5. The last goal is to review the current vision of the system to develop and propose improvements.

1.3 Project Constraints
Developing of system has several important constraints. We should use the same technology stack as the current system is implemented with. This includes: Linux as server operation system, Java as programming language, GWT for web-based GUI, MySQL with Hibernate or JDBC as RDBMS, Apache Tomcat as web server and servlet container, Apache Maven for build-automation.

1.4 Structure of Report
Section 2 describes the technology which we used in our program during development. Section 3 describes the requirements for developing product. Section 4 describes the analysis of architectures that were considered. Section 5 describes the architecture of the complete system, which we have chosen and developed. Section 6 describes the system implementation and experiments. Section 7 describes the interaction within the team during the implementation process and writing the thesis. In Section 8 we present the conclusion and future work.
2 Background
In this section we will write about technologies, frameworks, and tools which we use during system development.

2.1 Netty Framework
There are many Java New Input/Output frameworks in the field of data transferring [17]. The essence of these frameworks do not improve the speed of data transfer or make it more secure but to provide a user-friendly interface design and useful frequently-used functions [17]. One of these frameworks is Netty. Since the customer company has a great experience in developing client-server applications using this framework, we decided to use it in our project.

Netty is a client server framework that allows to quickly and easily develop networking systems based on the protocol interaction between the server and the client. This greatly simplifies and speeds up network programming, based on the TCP and UDP socket servers [2]. Netty found a way to achieve simplicity of design, performance, stability, and flexibility without loss.

We used the Netty framework for ensuring communication between the server and the sensors, also for message encoding and decoding to / from a binary package [1].

![Netty architecture](image)

2.2 MySQL
MySQL is a free relational database management system. Normally MySQL is used as a server that is accessed by local or remote clients [3]. We use MySQL because it is free and optimal for our needs.

2.3 Java Database Connectivity (JDBC)
JDBC is a platform-independent industry standard allowing Java programs to interact with different databases. It is implemented as a package `java.sql` part of Java SE [4]. We decided to use JDBC for implementation database connection because it withstand high load and it is easy to synchronize database access.

2.4 Hibernate
Hibernate is a library for the Java programming language, designed to solve the problems of object-relational mapping. It is a free software and open source distributed under the GNU Lesser General Public License. This library provides an easy-to-use framework for mapping an object-oriented data model to traditional relational database. During the experiment revealed that Hibernate, under high load, works slower than JDBC[6].
2.5 Google Web Toolkit (GWT)
GWT is a technology on the basis of Java and developed by Google. This technology allows creating web-applications using almost only the Java language [7]. This greatly simplifies development of web-applications. Web-based applications written in GWT translate into optimized JavaScript and HTML. You can also debug code, which is a plus compared to the development on JavaScript [8]. It was the choice of implementation for the initial version of the system.

2.6 Android
Android - is the operating system for smart phones, tablets, electronic books, watches, laptops, Google Glass and other devices based on the Linux kernel and its own Java implementation from Google. Originally developed by Android Inc. Android allows you to create Java-based applications that control the device via Google-designed library. Android Native Development Kit allows you to port the library and application components written in C and other languages [9].

2.7 SVN
Subversion (SVN) is a free centralized version control system, which has been released in 2004 by CollabNet Inc. It is comfortable to use svn, because you always have your source code in the Internet, you also can view the history of commit and you can do backups to any revision [10].

2.8 Apache Maven
Apache Maven is a framework for automation build projects, specified in the XML-based language POM. Maven, as opposed to another collector projects Apache Ant, provides a declarative and not a mandatory assembly project. That is, in the files of the project pom.xml contains its declarative, not individual teams. All processing tasks to perform file Maven plug-ins [11].

2.9 Communication
The most common protocol in the world - a stack of TCP/IP, which traces its history from the network ARPAnet. The protocol got its name from a couple of protocols: Protocol IP network layer model OSI, which provides delivery of data between nodes, and the TCP transport layer, which makes the delivery of reliable. In addition to these two protocols, the stack includes many other protocols.

TCP/IP - Internet Protocol core. TCP/IP stack for dozens of years of development has incorporated a lot of other protocols: it is the protocols for the operation of services of hypertext WWW - HTTP, and email protocols SMTP and POP, and custom protocols to encrypt and decrypt transmitted data "on the fly", SSL [19].

TCP/IP stack supports a comfortable addressing system, has the ability to fragment packets, that is able to adjust the size of their transmission through networks that are based on different technologies. TCP/IP is supported by the overwhelming majority of modern operating systems. TCP/IP - the central record for the most popular desktop operating system that is different for different implementations of Windows and Unix.
3 Features & Requirements
In the following we provide a short overview of the core features of the first version of the sensor-based IT system for data collection and monitoring.

During our developing process we continuously communicated with the customers. This allowed us to defined complete requirements for the system and the emulator, such as: types of users, type of messages, use-cases, requirements for emulator, etc.

We used Scrum methodology allowing us to manage frequent changes in requirements keeping a focuses developing process [20]. In this chapter we describe only use cases and requirements which are refer to our part of thesis work.

The implementation of the advanced prototype should resemble the core requirements, but with an improved architecture and communication protocol.

3.1 System Overview
The system is a client-server application. The main function of the collection, processing and storage of data from the sensors. The system aims at improving the existing emergency alert.

Advantages:
Unification of the sensors. No matter what parameter measures the sensor, the system will understand it. The main requirement is compliance data exchange protocol.
Ability to display an image from a place of emergency. Prevents false alarms. Saves time to respond to an emergency, do not depend on the human factor.
Detailed history, allows to keep statistics and automatically raise an alarm if any of the parameters out of range.

Disadvantages:
At the moment, only one disadvantage is increases of power and intelligence sensors fluency on their cost.

The system provide automatization for reacting on an emergency situation. Basic use case describe communication between the system and people which are responsible for the reacting on emergency situation. The system produces a filter to avoid fake emergency alarm and saves time, money and efforts. A company which installed our equipment will be able to reduce the cost of the security system through actions automation. At an emergency situation the system will automatically call emergency services. In case of fake alarm, our system will not cause the rescuers, but nonetheless disconnect the alarm is only possible for a responsible person that was attached to the place. The system allows to monitor the location and provides convenient mobile interface for responsible persons. Each company has access only to the sensors that are registered with the name of the company. Each responsible person can get access to the last updated information about the state of the sensors and get sensors history.

Actors and technical specifications have been received from the customer.

3.2 Types of Users
The content of this section is similar to the content of section 3.1 in the master thesis of Ruslan Gederin and Viktor Mazepa [13].

All users in the system divided into three types: administrator, manager of a company and technician.

Administrator is a user with highest possibilities into all system. This type of users can: add/remove sensors to/from the system, add/remove new company to/from the
system, add/remove users to/from the system, attach/detach sensors to/from a manager of a company/technician, see all information in the system (messages, sensors, user, etc), change privileges for another types of users.

**Manager of a company** is a user with highest possibilities into single company. This type of users can: add/remove technician to/from the system, attach/detach technicians to/from his company’s sensors, see all information related to own company (messages, alarms, sensors, etc). Also manager of a company could be a responsible person for each sensor in own company. That means that this user will receive all notification messages (via TCP/IP) related to his sensors from the system.

**Technician** is a user which work with sensors (manager of a company define these sensors). Technician is responsible for sensor service and support. Technician receives all notification messages related to his sensors from the system.

Table 3.1 provides an overview of the users and their abilities.

**Table 3.1: Users**

<table>
<thead>
<tr>
<th>Administrator</th>
<th>Manager of a company</th>
<th>Technician</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add/remove sensors.</td>
<td>Add/remove technician.</td>
<td>Responsible for sensor service and support.</td>
</tr>
<tr>
<td>Add/remove company.</td>
<td>Attach/detach technicians.</td>
<td>See all information related to sensors which is</td>
</tr>
<tr>
<td>Add/remove users.</td>
<td>See all information related to own company.</td>
<td>attached to him.</td>
</tr>
<tr>
<td>Attach/detach sensors to/from manager/technician/company.</td>
<td>Could be a responsible person for each sensors in company.</td>
<td>Receives all notification messages related to his sensors.</td>
</tr>
<tr>
<td>See all information in the system.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change privilege for other types of users.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**3.3 Messages**

Table 3.2 provides list of messages with short description.

**Table 3.2: Messages**

<table>
<thead>
<tr>
<th>Name</th>
<th>Sender</th>
<th>Recipient</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECHO</td>
<td>Sensor</td>
<td>Server</td>
<td>Ping message</td>
</tr>
<tr>
<td>SENSOR_INFO</td>
<td>Sensor</td>
<td>Server</td>
<td>Information about device</td>
</tr>
<tr>
<td>REGULAR_SENSOR_DATA</td>
<td>Sensor</td>
<td>Server</td>
<td>Measured values</td>
</tr>
<tr>
<td>STOP_WORK_MESSAGE</td>
<td>Sensor</td>
<td>Server</td>
<td>Device has stopped</td>
</tr>
<tr>
<td>ALARM_NOTIFICATION_RESPONSE</td>
<td>Sensor</td>
<td>Server</td>
<td>Device in alarm mode</td>
</tr>
<tr>
<td>PAUSE_ALARM</td>
<td>Sensor-Button</td>
<td>Server</td>
<td>Command to put device in pause mode</td>
</tr>
<tr>
<td>STOP_ALARM_BUTTON</td>
<td>Sensor-Button</td>
<td>Server</td>
<td>Command to stop alarm</td>
</tr>
<tr>
<td>BUTTON_REGULAR_RESPONSE</td>
<td>Sensor-Button</td>
<td>Server</td>
<td>Measured values</td>
</tr>
<tr>
<td>ECHO</td>
<td>Sensor-Button</td>
<td>Server</td>
<td>Ping message</td>
</tr>
<tr>
<td>GET_INFO</td>
<td>Server</td>
<td>Sensor</td>
<td>Command to get information about device</td>
</tr>
<tr>
<td>START_WORK</td>
<td>Server</td>
<td>Sensor</td>
<td>Command to start work</td>
</tr>
<tr>
<td>ALARM</td>
<td>Server</td>
<td>Sensor</td>
<td>Command to start alarm</td>
</tr>
<tr>
<td>PAUSE_ALARM</td>
<td>Server</td>
<td>Sensor</td>
<td>Command to pause alarm</td>
</tr>
</tbody>
</table>
### 3.4 Use Cases

The content of this section is similar to the content of section 3.3 in the master thesis of Ruslan Gederin and Viktor Mazepa [13].

Functional requirements to the system are formulated as set of use cases. Each use case has actors, precondition, post condition and successful scenario. Also each use case could have numbers of alternative scenarios.

In this section shows use cases which are related to our field of work on this system. It means that here we describes only features that emulator of sensors should support. These use cases defines with high level of abstraction (only important steps without detail). Also these use cases will be used as test cases for testing emulator and system works.

### 3.4.1 Data Acquisition

The content of this section is similar to the content of section 3.3.1 in the master thesis of Ruslan Gederin and Viktor Mazepa [13].

This use case (see Table 3.3) describes requirements to the data acquisition and transfer feature. This is the base use case in the system. It describes rules and scenarios for periodical sending Regular messages from the sensor with measured values.

<table>
<thead>
<tr>
<th>Use case name</th>
<th>Data acquisition.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors</td>
<td>Sensor, server, responsible persons.</td>
</tr>
<tr>
<td>Precondition</td>
<td>Server is running. The sensor is configured, operable and turned on; all information about it is stored in the database. The communication flow between server, sensor and database works.</td>
</tr>
<tr>
<td>Post condition</td>
<td>Obtained measured values are saved into the database.</td>
</tr>
</tbody>
</table>
| Successful scenario    | 1. Periodically sensor sends Regular message with measured values.  
                          2. Server receives new incoming message and recognizes it as Regular message.  
                          3. Server parses Regular message content and saves values from message into the database. |
| Alternative scenario #1| 1.1. Server does not received Regular message in time (in specified period).  
                          1.2. Regular message does not received after one minute.  
                          1.3. Server sends Regular request message to the sensor.  
                          1.4. Regular message received. |

Table 3.3: Data acquisition scenario
1.5. Go to the item 3 in successful scenario.

**Alternative scenario #2**
(Second regular message delay)

1.1 Server does not received *Regular message* in time (in specified period).
1.2 *Regular message* does not received after one minute.
1.3 Server sends *Regular request message* to the sensor.
1.4 *Regular message* does not received after two minutes.
1.5 Server sends *Turn on functional alarm* to this sensor.
1.6 Server sends *Hardware fail notification message* to the responsible persons.
1.7 Server saves information about alarm in the database.

### 3.4.2 Emergency Situation Detected by Sensor

The content of this section is similar to the content of section 3.3.2 in the master thesis of Ruslan Gederin and Viktor Mazepa [13].

This use case (see Table 3.4) specifies requirements for system work in case of emergency situation in monitored area which was detected by sensor. Server provides reaction on this situation.

**Table 3.4: Emergency situation detected by sensor scenario**

<table>
<thead>
<tr>
<th>Use case name</th>
<th>Emergency situation detected by sensor.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors</td>
<td>Sensor, server, responsible person.</td>
</tr>
<tr>
<td>Precondition</td>
<td>Server is running. Sensor is configured, operable and turned on; all information about it is stored in the database. The communication flow between server, sensor and database works.</td>
</tr>
<tr>
<td>Post condition</td>
<td>Emergency situation is finished. All information about emergency situation stored into the database. Responsible persons are notified.</td>
</tr>
</tbody>
</table>
| Successful scenario               | 1. Sensor detects emergency situation in monitored area.  
                                        2. Sensor turns on emergency alarm.  
                                        3. Sensor sends *Emergency situation detected message via* TCP/IP.  
                                          4. Sensor sends *Emergency situation detected message via* TCP/IP.  
                                          5. Server receives new message and recognizes it as *Emergency situation detected message*.  
                                          6. Server sends *Emergency situation detected notification message* to all responsible persons.  
                                          7. Server saves information about *emergency alarm* into the database.  
                                          8. Emergency situation is finished, all needful actions are performed.  
                                          9. Server sends *Turn off emergency alarm message* to the sensor.  
                                         10. Sensor receives this message and turn off emergency alarm.  
                                         11. Sensor sends *Confirmation message* to the server. |

### 3.4.3 Emergency Situation Detected by Server

The content of this section is similar to the content of section 3.3.3 in the master thesis of Ruslan Gederin and Viktor Mazepa [13].

This use case (see Table 3.5) specifies requirements for system work in case of emergency situation in monitored which was detected by server.
In this kind of emergency situation main role belongs to the server (unlike in previous use case). Sensor does not detect this kind of emergency situation – it is only sending measured values to the server. And server checks if this value higher or lower than corresponding threshold.

**Table 3.5: Emergency situation detected by server scenario**

<table>
<thead>
<tr>
<th>Use case name</th>
<th>Emergency situation detected by server.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actors</strong></td>
<td>Sensor, server, responsible person.</td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
<td>Server is running. Sensor is configured, operable and turned on; all information about it is stored in the database. The communication flow between server, sensor and database works. This sensor attached to the responsible person.</td>
</tr>
<tr>
<td><strong>Post condition</strong></td>
<td>Emergency situation is finished. All information about emergency situation stored into the database. Responsible persons are notified.</td>
</tr>
</tbody>
</table>
| **Successful scenario**    | 1. Server receives new incoming message and recognized it as Regular message.  
2. Server parse Regular message and obtain measured values.  
3. Server compares obtained values with the corresponding thresholds (both min and max value).  
4. One or several values are out of limits.  
5. Server saves information about emergency alarm into the database.  
6. Server sends Emergency situation detected notification message to all responsible persons.  
7. Server sends Turn on functional alarm message to the sensor which sent Regular message with value which is out of limits.  
9. The sensor send Confirmation message to the server.  
10. Server receives new incoming message and recognizes it as Confirmation message.  
11. Emergency situation is finished, all needful action are performed.  
12. Server sends Turn off functional alarm message to the sensor.  
13. Sensor receives this message and turn off functional alarm.  

#### 3.4.4 Hardware Problems

The content of this section is similar to the content of section 3.3.4 in the master thesis of Ruslan Gederin and Viktor Mazepa [13].

This use case (see Table 3.6) describes requirements for system work in case of detection any hardware problems. In this use case server receives message about hardware problems, stores information into the database and notifies responsible persons about them.

**Table 3.6: Hardware problems scenario**

<table>
<thead>
<tr>
<th>Use case name</th>
<th>Hardware problems.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actors</strong></td>
<td>Sensor, server, responsible person.</td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
<td>Server is running. Sensor is configured, operable and turned on; all information about it is stored in the database. The</td>
</tr>
<tr>
<td><strong>Post condition</strong></td>
<td>Functional alarm is finished. All information about it stored into the database. Responsible persons are notified.</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>

### 3.5 Requirements for Emulator

The content of this section is similar to the content of section 3.4 in the master thesis of Ruslan Gederin and Viktor Mazepa [13].

Besides the above scenarios the following requirements are defined:

- The emulator should
  - be a part of main web application (as a tab).
  - provide possibility to chose sensors for emulation.
  - visualize each sensor.

- Each sensor should
  - visualize as separate box and should include all equipment as the real sensor.
  - emulate all behavior of real sensor.
  - update his own state every 3 seconds.
  - visualize measured values in readable form.

- The emulator should have output console for showing all information about emulator work.
- The output console should have check box for filtering Regular messages.
- The output console should have buttons for managing scrolls and clear console.

*Main requirement for emulator – it should fully copy behavior of the real devices.*

### 3.6 Summary

After extensive testing and research on the basis of the results of testing of prototypes and the speed of the prototype applications running on server equipment of company was selected second case of the architecture as the most optimal for the initial stage of development of the concept. Using SMS or Email to transmit data from the sensor to the server is not optimal and want to improve this part, replace it with the TCP/IP. To transfer data, we have developed a special protocol for data store in the messages and transmit it over the network.
4 Architecture Analysis
This section describes and analyses different alternative architectures for the server and sensor implementation, which all fulfill the requirements stated in Section 3.

4.1 Server Architecture
In server developing were taken into account three models of sensor deployment:

4.1.1 Single Server
This architecture is focused on use of a single server for all tasks. The server hosts the application to control the sensors, database and web site.

Sensors connected directly to the server and transmitting it to the information processing [14]. Server keeps all logic in one place Figure 4.1.

Advantages:
1. Easy and quick to implement. The logic and data are in a location that facilitates the task of data processing.
2. Easy to support, low costs for service. Use one physical server that stores all the information. Rent only one machine reduces costs.

Disadvantages:
1. Low scalability. Under a heavy load productivity can be increased only through increasing the power of the equipment.

![Figure 4.1: Single server](image)

4.1.2 Logic Clusters
The architecture divides into two types Figure 4.2: the cluster for the exchange of data with the sensor and the cluster for processing the data.

Advantages:
1. High scalability. System performance easily increased by redirecting tasks with busier server to a less loaded.
2. High speed. The load distribution can increase performance.
3. Great potential. Load sharing allows more efficient use of power equipment.

Disadvantages:
1. The high complexity and slow development. Additional logic is costly.
2. The high cost of support. Support requires additional costs as number of hardware and software complexity increases.
4.2 Sensor Communication Architecture

We are taking into account three models of sensors behavior.

4.2.1 Sensors in Group with Master Node

Sensors are associated with each other in the group Figure 4.3. In each group we have master sensor which know about all sensors in its group. Each sensor in the group, except master-sensor sends the response to the command or the current parameters as a message to sensor-master. Sensor-master analyzes the message and sends aggregated data to the server. Also, the sensor-master receives messages for its group, and distributes them to each sensor.

All sensors go to sleep after sending a message. Sensor-master does not fall asleep and never closes the connection to the server and, if it necessary wakes of other sensors on group.
**Advantages:**
1. Reducing the load on the server side, because server communicates with only one sensor of the group (sensor button).
2. Sensors are becoming more intellectually that simplifies the development of server-side.

**Disadvantages:**
1. The high cost of software development for sensors, as the sensors have to be smart.
2. Fast battery consumption at the master-sensor, because it never goes to sleep and does not close the connection to the server.

**4.2.2 Individual Sensors**

Figure 4.4 each sensor communicates directly with the server and not combined into a group with other sensors. Each sensor sends its current message to the server and waits for a response. After receiving the response processes the received command and sends back the current message to the server.

![Diagram of individual sensors](image)

**Advantages:**
1. Low cost of development for Sensors.

**Disadvantages:**
1. Fast battery consumption of sensors, as they do not close the connection to the server, and do not stop communication with it.
2. In the event of a temperature detection of one sensor, the server cannot determine the nearest sensors to it, and cannot send turn of alarm to all of them.
4.2.3 Individual Sensors in Group

Each sensor sends a request to the server and then gets a response, closes the connection, processes the information, sleepy at T time. Next, the sensor wakes up, connects to the server and sends a prepared message.

The server receives the request, reads it, finds the message in the model state of sensor to the previous request and sends it. Server then processes the data, generates and stores the answer in the model for this sensor, and when sensor connect next time server will send it.

It is as in previous variant, but also, each sensor associated with the group in the database, sensors do not know anything about the group, only the server coordinates their work together.

![Diagram of sensor communication](image)

**Figure 4.5:** Independent sensors in groups

**Advantages:**
1. Low cost of development software for Sensors, because sensors are not responsible for the communication within the group.
2. Keeping battery consumption, because sensor sleeping and close connection with server.
3. When detecting a threshold violation, server can easily identify a group of sensors using the database.

4.2.4 Summary

In the end we decided to implement third variant of sensors, because we should resolve problems with energy consumption and the cost of manufacture of sensors.
The level of sophistication of sensors
Due to the need to form groups of sensors, the logic of this process has been implemented on the server side, and the sensors are made as easy as possible, which reduced the power consumption and cost.

Battery consumption
We studied various aspects and schemes of the sensors interaction. Different ways of power and the possibility of charging. It was decided that the sensor spends most of his time in power-saving mode, and only some time in working mode. About 750 milliseconds sensor sleeps and 250 runs. During an emergency, these parameters change.
5 Architecture
This section describes the candidate architecture of the server and sensor network as well as the communication protocol.

5.1 Server Architecture
We selected "Single Server" Figure 5.1 as our approach, since this architecture allows a short time to create a workable server. On the server, the database is located under the control of the MySQL database and a Java virtual machine that runs the basic logic.

Interaction scheme was implemented called "bulletin board". This type of communication allows you to make asynchronous requests to the server, which saves battery by changing the mode of operation of sensors.

With the development of server utilized a popular pattern MVC. Use of this pattern it possible to achieve high distribution logic, much reduces the cost of supporting the server.

![Server Architecture Diagram](image)

Figure 5.1: Server architecture

5.2 Emulator Architecture
Since we do not have actual sensors we can easily reprogram for performing our tests, we implement an emulator. Sensors in our program will be presented by simulator. We must develop simulator for showing behavior of future implementation of real sensors. Simulator will be developed in GWT, it was the condition of the customer. GWT application will be a web-application, where user can control and change sensor behavior. And server side of web-app will communicate with main analysis server of the sensor monitoring system Figure 5.2 and change UI dependent of receiving commands from main server.
5.3 Communication
We propose to replace the SMS and e-mail based protocol with the following TCP based protocol.

5.3.1 Communication Protocol
We decided to develop communication protocol as format of sending and receiving data in messages [15]. Its description and structure is illustrated in Table 5.1 and Table 5.2.

Definitions:
1. length – total bytes number of package
2. pVersion – version number of protocol
3. sVersion – version number of sensor type
4. messageType – message version number
5. preValidation – check sum for headers
6. data – message data. Contains sensor id, message description, some parameters.
7. postValidation – check sum for message data

<table>
<thead>
<tr>
<th>№</th>
<th>Bytes size</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1-4</td>
<td>length</td>
<td>Total bytes number of package</td>
</tr>
<tr>
<td>2</td>
<td>5-8</td>
<td>pVersion</td>
<td>Version number of protocol</td>
</tr>
<tr>
<td>3</td>
<td>9-12</td>
<td>sVersion</td>
<td>Version number of sensor type</td>
</tr>
<tr>
<td>4</td>
<td>13-14</td>
<td>messageType</td>
<td>Message version number</td>
</tr>
<tr>
<td>5</td>
<td>15-18</td>
<td>preValidation</td>
<td>Check sum for headers</td>
</tr>
<tr>
<td>6</td>
<td>19-n</td>
<td>data</td>
<td>Message data. Contains sensor id, message description, some parameters</td>
</tr>
<tr>
<td>7</td>
<td>n-n+4</td>
<td>postValidation</td>
<td>Check sum for message data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>length</th>
<th>pVersion</th>
<th>sVersion</th>
<th>messageType</th>
<th>preValidation</th>
<th>data</th>
<th>postValidation</th>
</tr>
</thead>
</table>
5.3.2 Communication Messages
In the system we have implemented messages for communication between server and sensors. Here is the list of messages that used in last version of the system [16].

- ECHO
- SENSOR_INFO
- REGULAR_SENSOR_DATA
- STOP_WORK_MESSAGE
- ALARM_NOTIFICATION_RESPONCE
- PAUSE_ALARM
- STOP_DEVIATION_ALARM_BUTTON
- BUTTON_REGULAR_RESPONSE
- GET_INFO
- START_WORK
- DEVIATION_ALARM
- PAUSE_ALARM
- STOP_ALARM
- ALARM_NOTIFICATION
- REGULAR_SENSOR_DATA_RESPONCE
- SYSTEM_ALARM
- REPEAT_RESPONSE
- ERROR_MESSAGE
- STOP_SOUND

Messages contain headlines and content, headlines read by server then content converted to the desired message and sent to the processing.

5.4 Database
During the implementation of the database were considered two ways to store data. Information in the system is conventionally divided into two types, the first - is the data received and used for the sensors, the second - storing user data (information about users and companies)

Storage of data in the same database. Information about users and data from sensors stored in one database.

**Advantages:** Easy and quick to implement

**Disadvantages:** Low scalability.

Storage of user data and sensor data in different databases.

**Advantages:** High scalability. Load on the database does not block access to user data.

**Disadvantages:** Difficulty in implementation, high cost of support

We decided to implement first schema of DB. It has the low cost to support and implement.
6 Implementation & Experiments
This section describes the implementation details and evaluation of the selected architecture and communication protocol.

6.1 Server
After considering several architectures, we describe the architecture on which we stopped and realized in this thesis, briefly describe the process of sending and receiving messages, as well as how the server detects an emergency situation.

6.1.1 Blackboard Structure
To implement the message processing mechanism has been developed structure "bulletin board". Each message is received and processed separately.

The sensor sends a request to the server and receives the response without delay. The information is processed, then a delay sensor on the T time. After a delay, the sensor sends a new request.

The server receives the request, it reads from the stream and sends a response to a previous request or an empty response if the first request. Then processes the information on the last request and stores the response in memory to be sent later when the new request.

The result was selected last scheme in connection with the problem of power consumption by the sensor.

6.1.2 Send Messages
When a message is received from the sensor, the server depending on the input parameters or received message generates the response message and stores it in the model for future sending. When the sensor is re-connected for him in the model gets message and sends to the output stream. Using the tools provided by the framework Netty we override the method writeRequested() in class SimpleChannelHandler:

```java
@Override
public void writeRequested(ChannelHandlerContext ctx, MessageEvent e) {
    Message responseMessage = (Message) e.getMessage();
    addToRemind(responseMessage);

    ServerTools.saveAlarm(responseMessage);
    ServerTools.saveData(responseMessage);

    byte[] data = responseMessage.initProtocol().getData();
    ChannelBuffer buf = ChannelBuffers.buffer(data.length);
    buf.writeBytes(data);
    Channels.write(ctx, e.getFuture(), buf);
}
```

where we extract the object o for Message from MessageEvent, save in the model sent message. In the event of a deviation we save information about this incident. Save the sending data of response message to the database. From the message construct the protocol in accordance with the established structure, and extract from it a set of bytes for sending.

Data is stored in class Protocol:
public class Protocol {
    public int length = 0;
    public int version = 1;
    public int sensorType = 0;
    public short messageType = 0;
    public int preValidation = 871;
    public byte[] data = null;
    public int postValidation = 947;
}

// extracting a set of bytes from package for sent
public byte[] getData() {
    byte[] length = null;// int
    byte[] version = null;// int
    // /
    byte[] sensorType = null;// int
    byte[] messageType = null;// short
    byte[] preValidation = null;// int
    byte[] data = null;
    byte[] postValidation = null;// int
    length = Converter.intToByte(this.length);
    version = Converter.intToByte(this.version);
    sensorType = Converter.intToByte(this.sensorType);
    messageType = Converter.sortToByte(this.messageType);
    preValidation = Converter.intToByte(this.preValidation);
    data = this.data;
    postValidation = Converter.intToByte(this.postValidation);
    int packageSize = version.length + messageType.length
        + sensorType.length + preValidation.length + data.length
        + postValidation.length;
    length = Converter.intToByte(packageSize);
    ByteBuffer buffer = ByteBuffer.allocate(packageSize + length.length);
    buffer.put(length);
    buffer.put(version);
    buffer.put(sensorType);
    buffer.put(messageType);
    buffer.put(preValidation);
    buffer.put(data);
    buffer.put(postValidation);
    return buffer.array();
}

6.1.3 Receive Messages
To receive messages server reads all data from the socket stream. After that, the server
parses the byte stream at the headers and the message itself. Let's look on details on
converting the byte stream data. Initially the server generates a class of protocol using
the method:

```java
public static Protocol initProtocolFromStream(byte[] bPackage) {
    Protocol p = new Protocol();
    ByteBuffer bb = ByteBuffer.wrap(bPackage);
    p.version = bb.getInt();
    // /
    p.sensorType = bb.getInt();
    p.messageType = bb.getShort();
    p.preValidation = bb.getInt();
    int dataSize = bPackage.length - UtilVariable.INT_SIZE
```
```java
//
- UtilVariable.INT_SIZE - UtilVariable.SHORT_SIZE
- UtilVariable.INT_SIZE - UtilVariable.INT_SIZE;
byte[] temp = new byte[dataSize];
bb.get(temp);
p.data = temp;
p.postValidation = bb.getInt();

if (checkValidation(p.preValidation, p.postValidation, p.version)) {
    return p;
} else {
    return null;
}
}

After the creation of a protocol object is passed to the method for the determination of messages:

```
message = new StartWorkMessage();
break;
case MessageType.DETRACTION_ALARM:
description = new String(buffer.array());
message = new DeviationAlarmMessage(description);
break;
case MessageType.STOP_ALARM:
description = new String(buffer.array());
message = new StopAlarmMessage();
break;
case MessageType.REGULAR_SENSOR_DATA_RESPONSE:
description = new String(buffer.array());
message = new RegularResponseMessage();
break;
case MessageType.ALARM_NOTIFICATION_RESPONSE:
description = new String(buffer.array());
message = new RegularResponseMessage();
break;
case MessageType.ALARM_NOTIFICATION:
description = new String(buffer.array());
message = new AlarmNotificationMessage();
break;
case MessageType.BUTTON_START:
description = new String(buffer.array());
message = new ButtonStart();
break;
case MessageType.STOP_DEVIATION_BUTTON:
description = new String(buffer.array());
message = new ButtonStopDeviation();
break;
case MessageType.BUTTON_REGULAR_RESPONSE:
description = new String(buffer.array());
message = new ButtonRegularResponse();
break;
case MessageType.REPEAT_RESPONSE:
description = new String(buffer.array());
message = new RepeatResponseMessage();
break;
case MessageType.STOP_WORK_MESSAGE:
description = new String(buffer.array());
message = new StopWorkMessage();
break;
case MessageType.SYSTEM_ALARM:
description = new String(buffer.array());
message = new SystemAlarmMessage();
break;
default:
    System.err.println("Message.createMessage messageType= "
        + messageType);
}
message.sensorId = id;
return message;

With the help of polymorphism at the output we get the object of the class Message that contains data definition of a message that was transmitted from the sensor.

6.1.4 Checking Input Parameters
When receiving regular reports from the sensor, the server needs to compare the values with threshold values from the database:

```java
public static void regularDataReceiver(Message receiveMessage) {
```
RegularSensorDataMessage regularMessage = (RegularSensorDataMessage) receiveMessage;
    double battery = regularMessage.getBattery();
    ...
    // gets all parameters
    int sensorId = receiveMessage.sensorId;
    setSensorGroup(sensorId);
    
    SensorEntity sensorEntity = SensorsModel.INSTANCE.getSensorById(sensorId);
    if (sensorEntity != null && sensorEntity.getMinTemp() == Double.NaN) {
        double[] temp = QueryManager.getThresholdsTemp(sensorId);
        if (temp != null && temp.length == 2 && temp[0] != Double.NaN) {
            sensorEntity.setMinTemp(temp[0]);
        ...
        // sets min and max values for all parameters
        if they are not already set
    }
    // save regular data to DB
    long time = System.currentTimeMillis();
    QueryManager.addTemperature(sensorId, time, String.valueOf(regularMessage.getTemperature()));
    ...
    Message sendMessage = null;
    if (temperature > sensorEntity.getMaxTemp() || deviation > sensorEntity.getMaxDeviation) {
        receiveDeviation(regularMessage);
        receiveAlarm(regularMessage);
    } else if (battery < UtilVariable.MIN_BATTETY_LEVEL) {
        sendMessage = new SystemAlarmMessage(sensorId);
        initSensorEntityMessage(sensorId, sendMessage);
    } else {
        sendMessage = new RegularResponceMessage(sensorId);
        initSensorEntityMessage(sensorId, sendMessage);
    }
}

From received message we invoke the input parameters, set min and max values for sensor if they are not already set, save input parameters to database and depending of threshold values server makes decision about sending message. If server receives high value of temperature or any other deviation, server will format Deviation message for all sensors in group. If server receives high value of battery, server will format SystemAlarmMessage message for sensor. If server receives normal values of input parameters, server will format RegularResponceMessage message for sensor.

6.2 Sensor Emulation
This module was implemented with the following technologies:
a) Programming Technology GWT (Google Web Toolkit)
b) HyperText Markup Language HTML (HyperText Markup Language)
c) Cascading Style Sheets CSS (Cascading Style Sheets)
d) Testing With JUnit unit testing
The structure of the module:

a) org.sensor.client - in this package locate classes display the necessary widgets and user reactions

b) org.sensor.datatransfer - in this package are classes for adoption and send messages to and from the sensor

c) org.sensor.onesensor - in this package describes the interface for one sensor

d) org.sensor.server - in this package is the interface and interaction the functionality of the sensor.

Main features of the emulation-sensor:

a) Modeling of temperature increase

b) Modeling of any other deviation

c) Simulation of a low battery

d) Modeling delay sending messages

e) Simulations situation of stop deviation alarm (with control-buttons)

f) Simulation of the situation of deviation pause (with control-buttons)

g) Modeling on and off sensors

h) Modeling the behavior of sensors communication from one group via server

In this thesis work should simulate behavior when using two types of sensors:

a) Sensor - sensor determines the state of the environment (temperature and any other deviation) and the level of the battery. These now he must sent to the server for further processing.

b) Control buttons - controls the sensors behavior from one group in alarm situation.

Used to indicate that the alarm stopped (stop alarms in sensors) or deviation in pause (put the alarm and flashing of sensors in pause, which recovers over time.

The algorithm of the sensor behavior

When you turn on the simulator, the sensor sends Echo message to server, which contains a unique serial number of the sensor, its gps location and type. If the message is a lossless server asks for more information about the sensor and the sensor sends it. After receiving permission to work, the sensor sends the value of temperature, other parameters and charge its battery. Simulator with UI contains four sensors and one sensor-button belongs to the same group of sensors.

To run the sensor you must click "Start" after this sensor sends its first message to the server.

For simulation of high temperature, press "Deviation" button and then the sensor will send temperatures above the permissible value.

For simulation of a deviation situation, press "Deviation 1" button and then the sensor will send the value of the deviation above allowable.

To simulate low battery of sensor, press "Battery" button and then the sensor will send a low battery level on server.

In order to simulate the situation of data loss when sending message, press the button "Error" and the following message will send with wrong checksum.
To posted image of environment near sensor click the button "Image" and the following message will send image to server.

To stop the sensor you must click on "Stop" button, and the message will send to server and server will delete this sensor from its active model.

View of one not running sensor simulator in Figure 6.1. The right and left lists are show sent message from the simulator to the server and received messages from the server.

![Figure 6.1: One sensor view](image)

The algorithm of the control buttons Figure 6.2

Press the "Stop Deviation" button for stopping flashing and alarm on sensors in the group. That event will send a message to the server to terminate deviation, then the server determines the sensors in the group and sends to them command to turn off the alarm and blinking. You can also pause the sound and flashing on sensors on certain time and leave only the flashing on the sensor that sent a high value of temperature or any other deviation, to determine the source of the deviation. To put the "Deviation" to pause, press the button "Pause Deviation".

View of control buttons on the simulators.

![Figure 6.2: Control-buttons](image)
Simulation of Emergency Situations
Sensor simulator in normal mode, running sensors and one sensor-button to control deviation Figure 6.3.

Figure 6.3: Sensor's work

When you click on "Deviation 1" or "Deviation 2" at the first sensor, the server determines the high value of parameter and sends to all the sensors in the group command Start Deviation. And all sensors in the group change their color to red, see Figure 6.4.

Figure 6.4: Detecting deviation, turn on alarm
When you click "Pause Deviation" on the button-sensor all sensors, except the deviation started, returning to usual status. Sensor, which detects deviation situation changes color to yellow, which means stopping the deviation alarm, but continued blinking, see Figure 6.5.

![Figure 6.5: Pause alarm](image)

After 10 seconds, all the sensors again turn on alarm. When you click button "Stop Deviation" on the sensor-button all sensors returned to their normal mode.

### 6.3 Web-application
Displays information about the system and it is essential for managing the sensors (incl. emulated sensors) for testing and production.

1. You can view all users and their details Figure 6.6: name, phone number, email address, the status of the user, whether the user is certified and the company to which the user belongs.

   ![Figure 6.6: User management](image)

2. You can view all the registered companies in the system, as well as information about them Figure 6.7: the name of the company, the phone company and its address.
3. You can see all sensors and information about them Figure 6.8: the user which responsible for the group and its contact telephone.

4. You can view information about all the messages received from sensors in the system Figure 6.9. To do this, select the period of time for received messages, and group. Then you will see all messages from all sensors of the group for that period. You can also choose specific sensor in the group, and then you will see all messages from this sensor.

5. You can view information about yourself Figure 6.10.
Add / Edit information
1. You can edit an existing user in popup window by clicking on the Edit button in Users page, in front of an existing user Figure 6.11. You can also add a new user by clicking Add button.
2. You can register a new company in popup window Figure 6.12 by clicking on Add button on the page Companies or edit information about an existing company by clicking on the Edit button in front of an existing company.

![Add new company dialog](image)

Figure 6.12: Add new company dialog

3. You can also update your details by editing the fields on the page My Settings Deleting information.

You can delete any information user or companies by clicking on the Delete button on the appropriate page.

### 6.4 Tablet-application

To interact with the system was designed app for tablets. The application allows you to log in and see the details of the sensors. The application allows you to edit information in the system, so, for example, if the sensor triggers an alarm, the user can check the real state of the sensor and change the signal to normal.

### 6.5 Smartphone-application

The application for smart phones also allows you to log in, but it does not allow edit. This is done in order to divide users on those who are responsible for the performance of the system and for those who are looking at the state of the system. Users of the application have access to all the information about the system, and so they have the opportunity to order a status report that will be sent by email.

### 6.6. Improvement protocol communication

Following the development of the thesis was conducted the test case. During which measured the time of transmission between the server and the sensor, and the amount of lost messages.

We draw a brief comparison of the results with the existing system.

**Table 6.1: Time for message sending**

<table>
<thead>
<tr>
<th>APEA Sensor Network version 1</th>
<th>APEA Sensor Network version 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 – 30 seconds</td>
<td>0–1 second</td>
</tr>
</tbody>
</table>

The next challenge was to improve the reliability of sending messages and taking action on loss of data in the message.

In the first version APEA Sensor Network loss of messages when sending 100 messages was 6%. When using TCP/IP we have not situation of message loosing. Because TCP - «guaranteed» transport mechanism that gives confidence in the
correctness of the data, the data re-requests in the event of loss and eliminates duplication of data.

TCP allows you to adjust the load on the network, as well as to reduce the waiting time of data over long distances. More over, TCP guarantees that the data was sent in exactly the same sequence. We have also provided the functionality for re-query the data from the sensor in case of lost messages.
7 Conclusions and Future Work
In this chapter, we summarize our work and reflect on the goals that were set for the development of the diploma and how they were resolved. We also tell you about our future work that we would like to modify this system.

7.1 Conclusions
The problems to be solved by this thesis were to:

(a) propose an alternative architecture and communication protocol for the sensors,
(b) to implement an advanced prototype for evaluation, and
(c) to provide all necessary means to facilitate an evaluation of the proposed improvements.

We solve the problems by reaching the stated goals as following:

1. The first goal is to evaluate different alternatives for the overall server architecture. This goal is reached if we describe and compare at least two different alternative architectures. We did this in Section 4.1 and selected for evaluation purposes to implement a single server approach, even though a cluster of servers has in the long run more scalability.

2. The second goal is to evaluate different alternatives for sensor organization. This goal is reached if we describe and compare three different alternative architectures/organizational patterns. We did this in Section 4.2 where we selected a grouped approaches next step since it is less effort and cost for updating the existing sensors, but in future the master node configuration should the development target due to its advantages of low average battery consumption.

3. The third goal is to propose and implement an alternative communication protocol. This goal is reached if we describe and implement a protocol different (and more efficient) to the current implementation being based on SMS and e-mail message exchange. We did this in Section 5.4 where we selected a TCP based protocol replacing the existing SMS and e-mail based protocol. We also made comparison of communication protocol with first version in Section 6.6.

4. The fourth goal is to implement and evaluate the most promising architecture and protocol alternative (advanced prototype) using an emulator and any required server components for facilitating the testing. We did that as described in Section 5 (Architecture) and Section 6 (Implementation).

5. The last goal is to review the current vision of the system to develop and propose improvements. We did that in Section 6, where we proposed different improvements.

7.2 Future Work
Of course in the future work can be finished some features. For example we can extend the Web-and Smartphone applications, adding new capabilities to monitor the behavior of the system.

It was also possible to assign a precise GPS coordinates of each sensor location and display on map each sensor in the building. In the event of an emergency the sensor, which discovered the deviation could be illuminated red or display of the sensor with low battery - flashes on the map [17]. This functionality will allow remotely on the map see the behavior of the sensors.
References


[8] All that you need for GWT. Available at http://www.gwtproject.org/ [Accessed 17 August 2013]


