One world is not enough
Development of a software system for connecting ZigBee devices to an IoT gateway

Rob van Bekkum
Jason van Belzen
Hong Yi Victor Li
ONE WORLD IS NOT ENOUGH

DEVELOPMENT OF A SOFTWARE SYSTEM FOR CONNECTING ZIGBEE DEVICES TO AN IOT GATEWAY

Rob van Bekkum
Jason van Belzen
Hong Yi Victor Li

Bachelor Thesis
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
June, 2015

Presentation: Friday June 26, 2015
Supervisors: Dr. R. R. Venkatesha Prasad, R. Riemens, TU Delft
Dr. ir. F. F. J. Hermans, TU Delft
Dr. M. A. Larson, DSP Innovation B.V.

An electronic version of this thesis is available at http://repository.tudelft.nl/
This report concludes the project comprising of the software development for the SWYCS IoT gateway conducted as part of the Bachelor Computer Science program at the Delft University of Technology. The aim of this report is documenting the complete development process of the SWYCS IoT gateway software system and presenting the results of the project to the members of the Bachelor Project Committee. This project was commissioned by DSP Innovation B.V. and was carried out in the fourth quarter of the academic year of 2014-2015. During the past nine weeks, research was conducted and assessments were made to come to a design followed by an implementation taking the requirements imposed on the system into account.

First of all we would like to thank Rene Riemens, managing director at DSP Innovation B.V., for the regular and effective communication with the project team and for supplying additional hardware which eased the development of the software system. Finally, a special thanks to our TU coach, dr. Ranga Rao Venkathesha Prasad, assistant professor of the Embedded Software Group at the Delft University of Technology, for his guidance during this project and suggestions for enhancing the developed software system.

Delft, 19 June 2015

Rob van Bekkum
Jason van Belzen
Hong Yi Victor Li
SUMMARY

With the dawn of the era of the Internet of Things (IoT) more everyday objects are equipped with technology that allows them to be connected to the Internet and therefore enabling (remotely) controlling or gathering data from these objects. DSP Innovation B.V. aims to respond on this emerging development by introducing a second generation IoT gateway to be used in combination with their SWYCS energy management solutions.

During this project a software system for this new gateway has been developed, which allows several third-party ZigBee-enabled devices to connect to the gateway. The software system allows for remotely retrieving real-time device data and modifying the configurations of devices connected to the IoT gateway. Measurement data of the devices connected to the gateway is automatically stored by the developed software solution and regularly pushed to the DSP cloud servers via a secure connection. The historical data in the DSP cloud servers can be used by the already existing SWYCS software solutions to provide their customers with insights of their energy usage and automatically assisting them in achieving energy savings in their company.

To come to the design of the software system for the IoT gateway research was done into the various communication protocols that could be used and the hardware that was used. For the current application, the ZigBee communication protocol with the ZigBee Home Automation stack was found to be the most appropriate choice, even though the software system also supports connecting lighting appliances operating using ZigBee Light Link.

At the end of the project a fully functional prototype of the software system has been delivered. While the delivered system is a prototype it is not production-ready yet. An automated software installation and update process should be integrated and support for more devices should be added when the system is prepared for production and presentation to customers.
# CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>iii</td>
</tr>
<tr>
<td>Summary</td>
<td>v</td>
</tr>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1 Design Goals and Requirements</td>
<td>3</td>
</tr>
<tr>
<td>1.1 Design Goals</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Requirements</td>
<td>4</td>
</tr>
<tr>
<td>1.3 Risk analysis</td>
<td>6</td>
</tr>
<tr>
<td>2 Research</td>
<td>7</td>
</tr>
<tr>
<td>2.1 Problem Definition and Analysis</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Internet of Things</td>
<td>8</td>
</tr>
<tr>
<td>2.3 Communication Protocols</td>
<td>8</td>
</tr>
<tr>
<td>2.4 Hardware Components</td>
<td>14</td>
</tr>
<tr>
<td>2.5 Implementation</td>
<td>15</td>
</tr>
<tr>
<td>3 Architecture Design</td>
<td>17</td>
</tr>
<tr>
<td>3.1 Overall Architecture</td>
<td>17</td>
</tr>
<tr>
<td>3.2 Dynamic Structure of the Functional Modules</td>
<td>19</td>
</tr>
<tr>
<td>4 System Implementation</td>
<td>23</td>
</tr>
<tr>
<td>4.1 Overall structure</td>
<td>23</td>
</tr>
<tr>
<td>4.2 System Setup and Deployment</td>
<td>25</td>
</tr>
<tr>
<td>4.3 Gateway Cross-Compilation</td>
<td>26</td>
</tr>
<tr>
<td>5 Software Quality</td>
<td>27</td>
</tr>
<tr>
<td>5.1 Code Quality Engineering</td>
<td>27</td>
</tr>
<tr>
<td>5.2 SIG Code Evaluation</td>
<td>28</td>
</tr>
<tr>
<td>5.3 Documentation</td>
<td>29</td>
</tr>
<tr>
<td>5.4 Testing</td>
<td>29</td>
</tr>
<tr>
<td>5.5 Security</td>
<td>30</td>
</tr>
<tr>
<td>6 Process Evaluation</td>
<td>33</td>
</tr>
<tr>
<td>6.1 Agile Development Reflection</td>
<td>33</td>
</tr>
<tr>
<td>6.2 Development Resources</td>
<td>34</td>
</tr>
<tr>
<td>6.3 Encountered Issues</td>
<td>34</td>
</tr>
<tr>
<td>7 Prototype Evaluation</td>
<td>35</td>
</tr>
<tr>
<td>7.1 Evaluation of Requirements</td>
<td>35</td>
</tr>
<tr>
<td>7.2 Evaluation of Design Goals</td>
<td>36</td>
</tr>
<tr>
<td>7.3 Overall Evaluation</td>
<td>36</td>
</tr>
<tr>
<td>8 Future Work and Recommendations</td>
<td>37</td>
</tr>
<tr>
<td>Conclusion</td>
<td>39</td>
</tr>
<tr>
<td>A Project Information</td>
<td>41</td>
</tr>
<tr>
<td>A.1 Project Description</td>
<td>41</td>
</tr>
<tr>
<td>A.2 Company Description</td>
<td>41</td>
</tr>
<tr>
<td>A.3 Auxiliary Information</td>
<td>42</td>
</tr>
<tr>
<td>B SIG Code Evaluation</td>
<td>43</td>
</tr>
<tr>
<td>B.1 Feedback First Submission</td>
<td>43</td>
</tr>
</tbody>
</table>
Over the past years a new rapid development has emerged which is known as the Internet of Things (IoT) referring to “the networked interconnection of everyday objects, which are often equipped with ubiquitous intelligence” (Xia, Yang, Wang, & Vinel, 2012). DSP Innovation B.V., a company involved in the development of innovative energy and communication concepts, aims to react on this development by introducing a new energy management and communications platform. DSP Innovation B.V. has already developed a first version of this platform in the form of its SWYCS (See What You Can Save) energy management solutions.

The SWYCS energy management solutions consist of a gateway which is installed at the location of the customer and to which several ZigBee devices can be connected. The company offers their self-produced compatible devices among which are a smart-plug (which can be used for retrieving extensive energy usage data) and a central measurement unit. Further the SWYCS energy management solutions include a web portal through which customers can view real-time and online energy consumption measurements. SWYCS offers automated management of appliances with the goal of more efficient energy usage and gain of energy savings.

In this project a software system for a prototype second generation IoT gateway for the SWYCS energy management solutions has been developed. This gateway will eventually replace the first generation gateway to provide a generic solution for supporting the control and management of as many IoT ZigBee-enabled devices of third parties as possible. The aim of this report is to inform its readers of the design and development of this new SWYCS IoT gateway software system. This includes the discussion of the prior research that was conducted, the design choices that were made, the implementation of the system and evaluations on the development process and the delivered prototype.

Chapter 1 discusses the design goals and requirements that were identified in consultation with the client at the start of the project. Chapter 2 provides further defines the problem in more detail and elaborates on the relevant aspects that were researched in the research phase of the project to make an assessment of which of the various options for the development would be the best choice. Chapter 3 provides a detailed description of the architecture design of the SWYCS IoT gateway software system, that was setup keeping the goals and requirements defined earlier in mind. Subsequently Chapter 4 describes how the system was implemented according to this architecture design and which choices were made for this implementation. In Chapter 5 the different aspects of the software quality of the system are discussed, both how the software quality was borne in mind during the development process and how it was eventually evaluated. This is followed by an evaluation of the development process of the software system in Chapter 6 and an evaluation of the delivered prototype in 7. Finally in Chapter 8 some suggestions for future work are made and recommendations are provided that should be taken into account in further development of the SWYCS IoT gateway software system.
DESIGN GOALS AND REQUIREMENTS

One of the first and most important steps of the project has been identifying and defining the goals and requirements that should be taken into account in the design and implementation of the prototype. In Section 1.1 the design goals that were identified for the system are discussed. Subsequently Section 1.2 elaborates on the devised requirements that DSP Innovation B.V. puts on the prototype and which should be met at the end of the project.

1.1. DESIGN GOALS

The overall goal of the project is to develop a generic (software) solution to couple several different (third-party) ZigBee devices through the use of an NXP IoT Gateway. A generic join method should be developed that enables these devices to join the ZigBee network of the gateway, so that the IoT gateway will be able to manage and communicate with these devices. An API should be developed on the gateway that offers functionalities to retrieve data from the gateway and interface with the gateway and its connected devices. Further the gateway should store historical data that it has gathered, which should automatically be pushed to the DSP cloud server.

The remainder of this section elaborates on the main design goals concerning the security, maintainability and scalability of the software system to be developed. These are the goals that were to be borne in mind during the design and implementation of the system.

1.1.1. SECURITY

Security is one of the main design goals that has been borne in mind during the development of the software system. The software system had to be implemented in such a way that it is as secure as possible and not vulnerable to external attacks, so that the disclosure of confidential customer data is prevented. The next paragraphs briefly describe the main facets that needed to be taken into account during the development process.

ENCRYPTED COMMUNICATION

All communication involved should be secure. This includes both the communication of the IoT gateway with the connected devices, the communication of the gateway with the DSP servers to synchronize data, and the communication involved when requests are sent to the gateway.

RESTRICTED ACCESSIBILITY

The data stored on the gateway should not be accessible by unauthorized individuals. The gateway shall store data itself in local databases containing data of connected devices owned by customers. It should neither be
possible to retrieve or modify the data stored on the gateway, and therefore access to this data should be restricted.

1.1.2. **Maintainability**

Another main design goal is maintainability. It is important to bear in mind that other developers might wish to extend the produced software system in the future. Therefore it should be possible for new developers to be able to understand and easily extend the software system. Among others, it is of great importance to document the software and to let the software system output log messages so that the origin of errors can be traced back more easily. Apart from this there are other facets to be taken into account to enhance the maintainability of the software system.

**Modular Independence**

The various functionalities offered should be partitioned over independent modules where appropriate. This makes it easier to extend or adapt the implementation of one part of the system without influencing the rest of the system. When the system is divided over several modules and one part of the system fails, this would decrease the probability of the rest of the system to fail consecutively.

**Closed for Modification, Open for Extension**

One of the important goals that had to be borne in mind, is that the software system should be designed and implemented in such way that it would remain closed for modification, but always open for extension, whenever achievable. This means that it should be able to extend the existing implementation without altering the existing source code when new functionalities are required when the requirements/needs change.

1.1.3. **Scalability**

Finally the last design goal that needed to be taken into account during the development process, was that the software system should be scalable. First of all the software system should be designed and implemented in such way that support for new device types can be added easily. Further the software for the IoT gateway should be implemented in such way that the gateway will be able to handle a great number of connected devices. As the number of connected devices grows, the gateway should remain capable to store and synchronize data for each of these devices.

1.2. **Requirements**

The requirements have been devised in accordance with the desires of the product owner, Rene Riemens, managing director at DSP Innovation B.V. The requirements for the final product are the result of consultation with the product owner and the project team.

1.2.1. **Functional requirements**

The requirements regarding functionality and service are grouped under the *functional requirements*. Within these functional requirements we identify four levels of priority, where a level one requirement has the highest priority and a level four requirement has the lowest priority.

**General**

- The system should offer the possibility to pair ZigBee devices through discovery of suitable devices for pairing, triggered through a local web interface or button push on the gateway [**Priority: 1**].
- The system should offer the possibility to pair ZigBee devices using the NFC capabilities of the NXP IoT Gateway [**Priority: 4**].
1.2. Requirements

**Dimmable LED Light Lamps**

[Priority: 1]
For each dimmable LED lamp connected to the gateway, the system should:

- Offer the possibility to switch the lamp on/off.
- Offer the possibility to change the light intensity of the lamp.
- Retrieve real-time data from the lamp.
- Store historical data (e.g., on/off, power consumption, light intensity) retrieved from the lamp.

**Smartplugs**

[Priority: 2]
For each smartplug connected to the gateway, the system should:

- Offer the possibility to switch the smartplug on/off.
- Retrieve real-time data from the smartplug.
- Store historical data retrieved from the smartplug.

**CMU’s**

[Priority: 2]
For each Central Measurement Unit (CMU) connected to the gateway, the system should:

- Retrieve ‘real-time’ data from the CMU.
- Store historical data retrieved from the CMU.

**Light Sensors**

[Priority: 1]
For each light sensor connected to the gateway, the system should:

- Retrieve ‘real-time’ data from the light sensor.
- Store historical data retrieved from the light sensor.

**IoT Gateway**

- The system should retrieve the status of all ZigBee-enabled devices connected to it, and store this information in a local database [Priority: 1].
- The system should push device status information to a cloud server (managed by DSP Innovation B.V.) [Priority: 1].
- The system should remove data stored in its local database which is more than one month old [Priority: 4].
- The system should sync measurement data every half hour with the cloud server [Priority: 3].
- The system should store logging data locally on the gateway [Priority: 1].
- The system should offer the possibility to sync the logging data with the cloud server through an API call [Priority: 4].
- The system should sync the status of the devices connected to the gateway with the cloud server [Priority: 4].
- The API on the gateway should offer the following implemented functionality [Priority: 1]:

1. Design Goals and Requirements

- Reading data real-time from connected devices [PRIORITY: 1]
- Switching connected devices on/off [PRIORITY: 1]
- Linking connected light sensors and dimmable LED lamps to each other [PRIORITY: 2]
- Setting the gateway in its discovery mode (searching for ZigBee-enabled devices that could connect to the gateway) [PRIORITY: 1]
- Retrieving a list of connected devices [PRIORITY: 1]

• The gateway should offer a (local) web interface including:
  [PRIORITY: 2]
  - The possibility to connect devices to the gateway, by setting the gateway in its discovery mode.
  - The possibility to retrieve a list of devices connected to the gateway.
  - The possibility to change the name of devices (at least the name that is used by the gateway to refer to each device).
  - The possibility to disconnect devices from the gateway.

• The system should have an SSH tunnel to the DSP cloud server [PRIORITY: 3].

1.2.2. Non-Functional Requirements

Besides the provided functionality and services, design constraints need to be included in the requirements specification as well. These requirements do not indicate what the system should do, but instead indicate the constraints that apply to the system or the development process of the system.

• All communication should be encrypted, both the Zigbee-communication between the gateway and the connected devices as the communication of the gateway to the cloud server.

• The communication should be based on the ZigBee Home Automation Protocol.

• A fully functional version of the product should be delivered at 9 June 2015.

• The project should apply the Scrum methodology in its development process.

• The implementation should be thoroughly tested, preferably having at least 75% of meaningful test line coverage.

1.3. Risk Analysis

In this subsection we discuss the different risks involved in the project that could possibly affect the result of the project.

The first risk was the issue of the scarcity of the hardware. When we would not have a LED light, CMU and smartplug in time we could not program those devices. Fortunately we had some boards which could simulate some of these devices. For the Carrier Board with DR1175 Lighting/Sensor Expansion Board we already had a binary available to simulate the LED light. Other devices can be simulated using this development board, but it would require implementing this by ourselves.

Another possible, but more general risk is that one of the members of the project group could get sick. In that case we might not have had enough time to meet all the requirements.
In this chapter, the research that has been conducted in the research phase of the project has been documented. First of all the problem at hand had to be identified and analyzed. In Section 2.1 the relevant prior events in the company and current situation is identified from which the problem has originated. A further analysis is performed and an abstract overview is provided of how the company wishes to see the problem being tackled. As the project involves the implementation of a system on a dedicated IoT gateway, there should be a clear understanding of what the Internet of Things (IoT) is exactly as is discussed in Section 2.2. Several appliances should be wirelessly connected to this gateway and should communicate with this gateway. Several relevant communication protocols exist for this purpose which are thoroughly discussed in Section 2.3. Hereafter, Section 2.4 elaborates on the properties and workings of the hardware used in the project. Using all the research documented in these sections a solution is conducted, which coincides with the wishes and requirements established in consultation with the company. Finally, in Section 2.5 the plan for the implementation of this proposed solution is discussed.

2.1. Problem Definition and Analysis

Over the past years DSP Innovation B.V. has been concerned with the development of a platform for achieving energy savings with innovative energy and communication concepts supported by web portals. In November 2010 the company introduced their See What You Can Save (SWYCS) energy management solutions, which not only provides real-time, online measurements of energy consumptions, but links these to other measurements such as light intensity and temperature as well, to automatically manage the appliances connected to the system with the aim of achieving more efficient energy consumption. This automated management works by linking the retrieved information to each other and through this controlling the appliances and keeping track of the ongoing processes. DSP Innovation B.V.'s SWYCS solutions are both aimed at businesses and usage in households, and also provides its users with a communication portal through which their appliances are made accessible and controllable online.

Recently a new development, called the Internet of Things (IoT), has emerged in which more smart and self-acting devices are connected to the Internet, even more than there are people that control these devices over the Internet. Another strongly related development is Home Automation in which typical home appliances are automatically controlled or also even remotely controlled by their owners. DSP Innovation B.V. aims to anticipate on these rapid developments with the use of its SWYCS solutions. Though the main problem that arises is that the gateway that is currently used in combination with their SWYCS solutions does not comply with the standards for IoT/Home Automation and does not support communication with, and (possible) control of appliances of third-parties. Therefore DSP Innovation B.V. has recently started cooperating with NXP Semiconductors NV and is currently working on adapting the currently used hardware by deploying NXP's JN5168 wireless micro-controllers in this hardware for its SWYCS solutions. NXP Semiconductors NV is one of the sales leaders in semiconductors which has developed the family of JN516X wireless micro-controllers for bringing the Internet of Things into the home. The family of micro-controllers offers the capability of designing solutions for the Internet of Things using communication protocols including ZigBee, JenNet-IP and
other IEEE 802.15.4-based protocols (NXP Semiconductors NV, 2013a).

The project team has been assigned the task to develop a generic solution using the new generation IoT gateway to couple several third-party ZigBee devices through this gateway. ZigBee-enabled devices can then (wirelessly) be connected to this IoT gateway which will then exchange (real-time) data/information with this gateway which would otherwise not be available without the presence of this gateway. An API is to be developed on the IoT gateway that offers capabilities to retrieve an overview of the connected devices, retrieve information from these devices and gain access and control over these devices. This API shall then partly be made available through a local web interface to offer the ability to connect devices to the gateway, disconnect devices from the gateway, retrieve an overview of all connected devices or change the name of the devices. Further the API includes functionalities for retrieving real-time data, switching devices on/off or control devices in other ways.

2.2. **INTERNET OF THINGS**

Over the past 50 years the Internet has grown exponentially from a few research computers to 8.7 billion connected devices in 2012 (Soderbery, 2013). Nowadays an extended term of Internet is used increasingly, which is Internet of Things (IoT), but what does IoT mean exactly? Many definitions can be found at the Internet, but we stick to the Oxford definition. If we look up ‘Internet of Things’ in the Oxford Dictionaries, we will find that it is defined as *"The interconnection via the Internet of computing devices embedded in everyday objects, enabling them to send and receive data: ..."* (Oxford, n.d.). Internet of Things means that all everyday objects are able to communicate over the Internet. The reason for this grow of devices connected to the Internet is the miniaturization and cost reduction of electronics (Kopetz, 2011).

A lot of applications are thinkable within the concept of IoT. The European Research Cluster on the Internet of Things defined some important potential application areas. A description and research challenges of those application areas are described in Internet of Things: From Research and Innovation to Market Deployment (Vermesan & Friess, 2014). For now we only list a small overview:

- **Smart Cities**
- **Smart Energy and the Smart Grid**
- **Smart Mobility and Transport**
- **Smart Home, Smart Buildings and Infrastructure**
- **Smart Factory and Smart Manufacturing**
- **Smart Health**
- **Food and Water Tracking and Security**
- **Participatory Sensing**
- **Smart Logistics and Retail**

From here we can conclude that our project falls within the scope of Smart Home, Smart Buildings.

2.3. **COMMUNICATION PROTOCOLS**

Currently, DSP Innovation B.V. applies the ZigBee Pro communication protocol for the communication between the previous generation of gateways and devices connected to these gateways. Though with the new generation of IoT gateways the company wants to support more devices (including third-party devices) using the ZigBee Home Automation profile at the application layer. Several other communication protocols are however supported by the IoT gateway and there already exist other systems implementing yet more other communication protocols. In this section the several communication protocols are discussed, where we restrict ourselves to protocols and systems that can be used for home automation purposes.
2.3.1. ZigBee

ZigBee is an open standard for wireless communication between devices at a short range. This standard is set up by the ZigBee Alliance (The ZigBee Alliance | Control your World, n.d.) and based on the IEEE 802.15.4 standard, which defines the physical and MAC layer of the OSI model. ZigBee itself defines two layers on top: the network layer and the application layer. Figure 2.1 contains a graphical representation of the IEEE 802.15.4 and ZigBee Protocols.

![Figure 2.1: The ZigBee Architecture (Tennina et al., 2013).](Image)

The first layer we discuss is the application layer. This layer consists of the ZigBee Device Object (ZDO) and the Application Framework. The role of the ZDO is communicating information about itself and its provided services. The Application Framework contains the Application Objects 1 to 240, which are the manufacturer's applications running on top of the ZigBee protocol. These Application Objects adhere to predefined profiles, some profiles are: ZigBee Home Automation and ZigBee Light Link, which will be discussed later in this section. The complete list of profiles can be found at the ZigBee Alliance website (The ZigBee Alliance | Control your World, n.d.) under Application Standards.

The second layer we discuss is the Network Layer. The network layer is responsible for the network management, this includes procedures for joining, leaving, security and routing. The network layer depends on the ZigBee profile that is used. ZigBee defined three types of network specifications (Network Specifications | The ZigBee Alliance, n.d.).

- The first one is ZigBee PRO, which is designed to provide the foundation for IoT applications. It's a self-configuring, self-healing system.
- The second one is ZigBee RF4CE, which was designed for simple two-way-device-to-device control applications that do not require full mesh networks.
- And the last one is ZigBee IP, which is the first open standard for an IPv6 based full wireless mesh network.
We focus on the ZigBee PRO network, because that is the one designed for IoT applications. We do not treat all technical details of the network layer, but it is important to know that ZigBee defines three types of devices:

- **ZigBee Coordinator (ZC) or Gateway (GW)**, which initiates and configures the network information. This device implements the full protocol stack and is responsible for the internal networking.

- **ZigBee Router (ZR)** can also be responsible for coordinating depending on its position in the network. This device also includes the full protocol stack.

- **ZigBee End Device (ZED)** is an end device in the ZigBee network and should only have the possibility to send and receive messages from other devices. A ZigBee End Device includes only a reduced subset of the protocol stack.

With those three building blocks the following three ZigBee network topologies can exist: (a) **Star topology**, (b) **Mesh topology**, or (c) **Cluster Tree topology**. These topologies are visualized in Figure 2.2.

![ZigBee network topologies](image)

**Figure 2.2: The ZigBee network topologies (Tennina et al., 2013).**

As mentioned before it is necessary to choose a predefined application profile for ZigBee communication. The advantage of this standardization at the application level is the interoperability of different devices from different manufacturers. This is a critical difference of ZigBee in comparison with other technologies. Such a predefined profile describes which type of devices and what type of functions per device are supported. Supported profiles for the JN5168 are: Home Automation, Smart Energy, Light Link and Remote control applications, of which only Home Automation and Light Link are based on ZigBee PRO. Since Light Link is only meant for lightning applications we do not need this profile. Home Automation provides exactly what we need: controlling all kinds of devices in buildings.

### 2.3.2. JenNet-IP

JenNet-IP is a dynamic wireless network communication protocol standard using an enhanced version of the 6LoWPAN network layer which, just like ZigBee, reuses the IEEE 802.15.4 PHY and MAC layers as part of its wireless networking protocol (Farahani, 2011). A JenNet-IP system consists of two parts:

- **The WPAN domain** which contains one or more **Wireless Personal Area Networks (WPANs)** operating using the NXP JenNet protocol.

- **The LAN/WAN domain** which consists of a **Local Area Network (LAN)** that is usually connected to a **Wide Area Network (WAN)** to offer the capability of monitoring and controlling WPANs that are connected to the LAN through Border-Routers.

Further it is possible for a device in one WPAN to communicate with a device in another WPAN via the LAN/WAN domain of the JenNet-IP system. The structure of a typical JenNet-IP system with its main components is depicted in Figure 2.3. The IP host depicted in this figure is a device connected to the JenNet-IP via...
the LAN or WAN of the system to access it remotely (via a WAN, such as the Internet). It might also serve as an intermediary for remote accesses by acting as a web server for other IP hosts.

**THE 6LoWPAN protocol**

The term 6LoWPAN is an acronym of **IPv6 over Low power Wireless Personal Area Networks** and describes systems that communicate information between devices in a WPAN, called **nodes**, by means of IPv6 packets. In 6LoWPAN communication IPv6 packets are carried in the payload of IEEE 802.15.4 data frames. Though usually raw IPv6 packets do not fit the payload of these IEEE 802.15.4 frames and therefore the 6LoWPAN standard offers an **adaptation layer** on top of the MAC layer using compression, fragmentation and reassembling techniques to adapt IEEE 802.15.4 packets to IPv6, and vice versa ("JenNet-IP WPAN Stack User Guide (JN-UG-3080)", 2014). Figure 2.4 shows an overview of the layers in the 6LoWPAN protocol. Here the responsibility of the application layer is to collect data, initiating data transmissions and receiving data ("JenNet-IP WPAN Stack User Guide (JN-UG-3080)", 2014). Further the transport layer and network layer are concerned respectively with assembling IPv6 packets to be sent and disassembling received packets, and wireless network management.

![Diagram of Protocol Layers in 6LoWPAN](Farahani, 2011)
The main advantage of the 6LoWPAN protocol is network interoperability with other IP network links and so 6LoWPAN nodes are able to communicate with other IP-enabled devices even while these devices might not apply the 6LoWPAN protocol themselves (Farahani, 2011). The JenNet-IP protocol is based on the 6LoWPAN protocol, but enhances it with some supplementary layers which are described in the next part of this section.

**JENNET-IP SOFTWARE ARCHITECTURE**

The software of a JenNet-IP system is divided over three distinct parts of the system as depicted in Figure 2.5:

- The WPAN nodes
- The Border-Router(s) between the WPAN and LAN/WAN domain
- The devices in the LAN/WAN domain

![Figure 2.5: Software division in a JenNet-IP system ("JenNet-IP LAN/WAN Stack User Guide (JN-UG-3086)", 2014).](image)

Here the user application of the WPAN nodes operates over the JenNet-IP WPAN stack, communicating with the Border-Router via an IEEE 802.15.4 radio link. The main addition in the JenNet-IP WPAN stack to the 6LowPAN stack is the NXP JIP layer which uses a single communications port on a node to allow remote devices to set and retrieve data in a Management Information Base (MIB) on the node. MIBs store local variables and their values on the WPAN nodes and can be managed by using the JenNet-IP Embedded API for WPAN development.

The user application of LAN/WAN devices on the other hand operates over the JenNet-IP LAN/WAN stack, communicating with the Border-Router via an IP link. The JIP layer of this stack grants the possibility to these devices to access the nodes of the connected remote WPANs, so that it is possible to set and retrieve data from the nodes’ MIBs. To facilitate the access to the functionality of these nodes APIs for C/Java exist that can be used to develop applications for the IP Host devices.

### 2.3.3. Alternative Home Automation Protocols

There are also other home automation protocols available for smart home manufacturers. In this subsection, the most popular alternative home automation protocols will be discussed.

**Z-Wave**

Z-Wave is developed by the Z-Wave Alliance and is primarily designed for home automation, specifically for remote control applications in residential environments. The protocol uses low powered RF communications technology that supports full mesh networks without the need for a coordinator node. It theoretically supports up to 232 nodes, while ZigBee supports up to 500 nodes. The Z-Wave protocol operates on the
868Mhz-band and supports AES-128 encryption. Instead of using IEEE 802.11.4, Z-Wave uses ITU-T G.9959 standard for the PHY and MAC layer.

The Z-Wave protocol stack is closed and not open for modification. Also it can only run on their own silicon receivers. They sell standard modules that has to be built in into any product of other manufacturers. The advantage of this, is that Z-Wave guarantees compatibility between third party Z-Wave devices and backward compatibility. The drawback is that HA manufacturers cannot modify the protocol or evaluate it like ZigBee.

Routing in a Z-Wave network is not as dynamic as usual IP-networks due to reducing energy consumption. The devices in a Z-Wave network communicate through static routing tables, which are constructed by a optimisation algorithm. The primary controller asks every node in the network which nodes they are able to communicate with. Based on that information an optimal routing table will be constructed. Once a device leaves the network, this process has to start over again. Devices that are powered through AC power are able to communicate to each other through this method without intervention of the controller. This results in shorter response times.

**X10 RF**

X10 is also a home automation protocol, which is developed in 1975 by Pico Electronics. It primarily uses the wires in the power line for control and signaling, where the signals involve brief radio frequency bursts representing digital ones and zeros. This was the first general purpose domotic network technology and was leading for a long time. The main goal was to allow remote control of home devices and appliances. Later on, an RF protocol was defined in order to allow communication with key-chain controllers, wireless keypads, alarms, motion sensors, etc.

A couple of disadvantages of the X10 protocol, is that there is no encryption, there is a limitation of 6 devices per room, and only one-way communication is possible.

The protocol that is implemented in the popular KlikAanKlikUit (KAKU) devices, is a variation on the X10 RF protocol and uses the 433 MHz-band. It only communicates through RF and does not use the power line. Because it is a variation on the X10 protocol, it unfortunately shares the same disadvantages.

**KNX-RF**

KNX is the most used standard for building automation in commercial environments. This standard has protocols for power lines, networks with twisted pairs, Ethernet and wireless. The wireless protocol is called KNX-RF and operates on the 868MHz-band, like the Z-Wave protocol. It did not become as popular as the twisted-pair protocol, because KNX-RF has no encryption build in. This resulted in the standard becoming unusable for remote control applications in commercial buildings.

### 2.3.4. Comparison of Home Automation Protocols

<table>
<thead>
<tr>
<th></th>
<th>ZigBee HA</th>
<th>JenNet-IP (NXP)</th>
<th>Z-Wave</th>
<th>KNX-RF</th>
<th>X10 RF</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Year First Launched in Market</strong></td>
<td>2003</td>
<td>2011</td>
<td>2003</td>
<td>2005</td>
<td>1995</td>
</tr>
<tr>
<td><strong>PHY/MAC Standard</strong></td>
<td>IEEE 802.15.4</td>
<td>IEEE 802.15.4</td>
<td>ITU-T G.9959</td>
<td>ISO/IEC 14543-3</td>
<td>-</td>
</tr>
<tr>
<td><strong>RF Band Used</strong></td>
<td>2.4 GHz</td>
<td>2.4 GHz</td>
<td>868 MHz</td>
<td>868 MHz</td>
<td>433 MHz</td>
</tr>
<tr>
<td><strong>Maximum Data Rate</strong></td>
<td>250 kbps</td>
<td>250 kbps</td>
<td>200 kbps</td>
<td>9600 bit/s</td>
<td>20 bit/s</td>
</tr>
<tr>
<td><strong>Encryption</strong></td>
<td>AES-128</td>
<td>AES-128</td>
<td>AES-128</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Number of Nodes Supported</strong></td>
<td>500</td>
<td>500</td>
<td>232</td>
<td>64</td>
<td>256</td>
</tr>
<tr>
<td><strong>Communication Way</strong></td>
<td>Two-way</td>
<td>Two-way</td>
<td>Two-way</td>
<td>Two-way</td>
<td>One-way</td>
</tr>
<tr>
<td><strong>Topology</strong></td>
<td>Mesh, Star, Cluster tree</td>
<td>Mesh, Star, Linear</td>
<td>Mesh</td>
<td>Star</td>
<td>Peer to Peer</td>
</tr>
</tbody>
</table>

Table 2.1 shows an overview of the previously discussed home automation protocols. Scalability is important for IoT in the future, because later on, more devices and appliances will be connected to the Internet. Examples are washing machines, ovens, vacuum cleaners and many more which we cannot imagine nowadays. Nowadays ZigBee HA and JenNet-IP networks will be able to control and manage up to 500 devices in
one network, but in the future when the technology/hardware improves, the network will be able to support more devices.

Security is also important for the choice of protocol. Messages and requests between the gateway and the node should be encrypted against eavesdrops. ZigBee HA, JenNet-IP, and Z-Wave have AES-128 encryption built in their stack.

Because ZigBee is standardized, more popular, more supported by leading companies, and is better documented than JenNet-IP, we choose ZigBee over JenNet-IP as the communication protocol for the gateway.

### 2.4. Hardware Components

All of the hardware provided by DSP Innovation B.V. for this project contains a NXP JN5168 micro-controller (NXP Laboratories UK, n.d.), which allows communication according to IEEE802.15.4 on the 2.4 GHz-band, so it is compatible with ZigBee, JenNet-IP, and other IEEE 802.15.4-based protocols. In this section, we discuss all the hardware we are going to use during this project.

#### 2.4.1. NXP JN5168-RD6040 IoT Gateway

The NXP JN5168-RD6040 IoT Gateway, depicted in Figure 2.6a, is a small, compact implementation of an Ethernet gateway and is designed for low-power wireless networks (NXP Semiconductors NV, 2013b). The NXP LPC3240 ARM-9 (NXP Semiconductors NV, 2014) host controller runs a custom OpenWrt Linux operating system, which provides a platform for developing custom applications. It also has a local web server, which lets customers configure their network through a web interface.

The gateway host processor communicates with the JN5168 micro-controller through a serial connection via a UART. The JN5168 micro-controller will run as a ZigBee control bridge and communicates with ZigBee devices. The control bridge will act as a ZigBee HA Coordinator in a ZigBee network, which is described in Section 2.3.1.

#### 2.4.2. NXP DR1174 Carrier Board with DR1175 Lighting/Sensor Expansion Board

The NXP DR1174 carrier boards are development test boards which host a JN5168 module. They can be fitted with expansion boards to achieve the desired functionality. These expansion boards are mounted on top of the carrier boards. With these development boards, we are able to simulate ZigBee (Home Automation) devices.

The expansion board that we are going to use for the development of the new generation IoT gateway, is the NXP DR1175 Lighting/Sensor Expansion Board. This expansion board has dimmable LEDs, a color LED, a light...
sensor and a humidity/temperature sensor. With these features, the expansion board can act as a ZigBee HA dimmable LED, ZigBee HA color LED, etc. By flashing/programming the JN5168 chip on the carrier board, the desired test device can be simulated. The carrier board with applied expansion board is depicted in Figure 2.6b.

2.4.3. CMU

The CMU, which is depicted in Figure 2.7a, is a pulse counter developed by DSP Innovation B.V. The device can count on up to four devices simultaneously through the four different input ports. By knowing the number of pulses per consumption unit, the software can calculate the exact consumption. These pulses can be electrical pulses from a digital meter or analog pulses from the old mechanic meters (for example a flashing led or reflecting plate on the least significant rotating wheel). For the analog pulses CMU sensors are available. This CMU sensor, depicted in Figure 2.7b, can detect a flashing LED or reflected IR light from its own on board IR LED. The CMU has a JN5168 micro-controller on board as well which enables the device to connect to a ZigBee network.

![Central Measurement Unit, CMU Sensor, Smartplug](image)

Figure 2.7: Measuring hardware used by DSP Innovation B.V. (DSP Innovation B.V. Website, n.d.).

2.4.4. SmartPlug

Like the CMU, the SmartPlug, depicted in Figure 2.7c, is also developed by DSP. The smartplug contains three base components: the JN5168 micro-controller, the CS5464 (3-Channel, Single Phase Power/Energy IC) and a relay for switching. The CS5464 is an advanced integrated power measurement device which can measure instantaneous voltage, current and power IRMS and VRMS, Active, Reactive and Apparent Power (CS5464: 3-Channel, Single Phase Power/Energy IC, n.d.).

2.5. IMPLEMENTATION

Research into the relevant aspects of the problem has shown that software is needed for the IoT gateway, but also software for (the JN5168 micro-controllers of) the end-devices that DSP Innovation B.V. delivers, which are a smartplug, CMU and also a light sensor. As the gateway mainly needs to support communication with ZigBee-enabled end-devices the ZigBee PRO communication protocol will be applied. The ZigBee Home Automation profile is used with the advantage of better interoperability with devices from other manufacturers.

A generic connection solution for connecting devices to the gateway will be developed. To put the gateway into its discovery mode both a local web interface will be deployed and the possibility to push a button on the gateway. The local web interface will also provide functionalities to get an overview of all connected devices, changing the name of connected devices, and disconnect connected devices from the gateway. It will be also
possible for the DSP servers to establish an SSH connection with the gateway to push requests. An API will be available on the gateway to provide statuses, real-time and historical information of the connected ZigBee devices. This API can further also be employed to control the end-devices connected to the gateway (e.g., turning devices on/off, changing light intensity of lamps).

In addition, one might couple connected light sensors to one or more dimmable LED lamps. The measurements from the light sensors is then used to automatically regulate the light intensity of these LED lamps, possibly based on the preference of the user. In this way, when the sensors detect a change in the luminosity of a room (e.g., due to overcasts or day/night transitions) the light intensity of the lamps automatically change with this change in such way that the preferred luminosity is maintained.

To implement all these features, we will be deploying the JN-SW-4041 SDK Toolchain (Jennic, 2010) provided by NXP. This toolchain is based on Eclipse Ganymade and has build-in tools to develop, debug and flash ZigBee firmware. On top of the toolchain, the JN-SW-4067 ZigBee Home Automation SDK is installed to include ZigBee Home Automation libraries in the toolchain.
The software architecture is a high-level design of the software system describing the design decisions that are made to keep the system both maintainable and evolvable. The architecture design is the first and one of the most important stages in the software design process as it defines the main structural components of the system and how these components interact with each other. With all the goals and requirements in mind, an architecture design has been made for the implementation of the SWYCS IoT gateway software system. Section 3.1 discusses the overall architecture of the system, consisting of the (static) structure of the system’s parts and the role of the independently operating modules. Section 3.2 defines the dynamic structure of the system, which shows what happens at runtime and how the system acts in response to external or internal stimulus. It discusses how the different parts of the system act on external events and inputs, and how these parts once they are put together form the software system that meets the identified requirements.

### 3.1. Overall Architecture

The software system for the SWYCS IoT gateway should offer various functionalities among which are storing historical device data, syncing device data with the DSP server and offering the possibility to request real-time data of devices connected to the gateway. Therefore it is convenient to subdivide these system functionalities over multiple modules that can operate independently of each other. To present the structure of the software system a diagram is depicted in Figure 3.1, which shows the main components of the software system and their interrelationships.

![Diagram of the overall architecture design of the SWYCS IoT Gateway software system.](Figure_3.1.png)
A distinction is made between the software already available by default on the NXP IoT gateway and the modules part of the new SWYCS IoT gateway software system. First of all the gateway is equipped by default with the ZigBee JIP daemon which offers access to data of devices connected to the gateway using different ZigBee profiles (such as ZLL and ZHA) in a generic way. This is achieved by creating virtual devices each offering their own JIP MIBs allowing for access to the data of the connected devices. The JIP library provides accesses to these virtual devices, which in turn are converted by the ZigBee JIP daemon into requests and commands transported to the ZigBee Control Bridge over its bidirectional serial link with the gateway host. The functionality offered by this software already available on the gateway is reused by the modules of the SWYCS IoT gateway software system, which are depicted in blue in the diagram of Figure 3.1. The remainder of this section elaborates on the role of all functional modules in the software system and how these modules are related to one another.

3.1.1. **IoT Layer**

As several of the modules of the software system require access to the ZigBee network of the IoT gateway it is convenient to have a software component that allows for straightforward access to the data of the connected end-devices. The IoT layer takes the role of a common core of the software system that makes use of the JIP library to interface with the ZigBee network. It offers possibilities to retrieve data from devices, manipulate the configuration of devices, registering changes in the ZigBee network, triggering the gateway to allow new devices to join the network and accessing device data stored as entries in the databases of the gateway. The functionality offered by the IoT Layer are deployed by several of the other functional modules of the SWYCS IoT gateway software system as is discussed in the next sections.

3.1.2. **Watchdog**

The watchdog is a process that continuously runs on the gateway and monitors the ZigBee network of the IoT gateway. As soon as a change in the network occurs the watchdog reacts by executing a task corresponding to the event that occurred.

First of all the watchdog should react on the network event in which a new device joins the ZigBee network of the gateway. As soon as a new device joins the network information should be stored about the device into the device database. Data should be stored so that the joined device can later be identified (using its MAC-address) and so that can be retrieved when the device joined the network for the first time and whether it is currently connected or not. When a device joins the network of a gateway registered at the DSP cloud server, the new device itself should be registered at the DSP cloud server as well. Data should be stored in the gateway’s database that indicates whether the device has already been successfully registered at the DSP cloud server.

Further the watchdog should react on the network event in which a device which has earlier joined the network leaves the ZigBee network of the gateway. This means that the device is no longer active in the network, but might still be linked to the ZigBee network of the gateway, and therefore only the connection status of the device should be updated in the device database.

3.1.3. **Historical Data Fetcher**

The historical data fetcher collects data from all devices connected to the ZigBee network and stores it in the historical database with a time-stamp and corresponding device identifier for each device. The fetcher will be executed every five minutes, allowing the gateway to build up a history of the connected devices as long as the gateway is powered. A backup of the data will be stored in the database on the gateway for entries that were stored in the database up to 30 days back. The entries older than 30 days are removed from the database by the historical data fetcher. The data retrieved by the historical data fetcher is subsequently used by the cloud synchronizer as is explained in Section 3.1.4.
3.1.4. **Cloud Synchronizer**

The cloud synchronizer retrieves the collected data that has not already been pushed to the DSP cloud server from the historical database. The cloud synchronizer puts up a secure connection with the DSP cloud server to sent newly gathered data stored by the historical data fetcher in the database to the server every 30 minutes. When no connection could be set-up with the server, the cloud synchronizer will send the still unsent data in the next attempt to push data to the cloud server.

3.1.5. **RESTful API**

A web Application Programming Interface (API) is needed for the DSP servers to communicate, control and fetch (real-time) data from devices in the ZigBee network. Therefore the Representational State Transfer (REST) protocol is ideal for the implementation of the API. It uses standard HTTP methods (e.g. GET, POST, DELETE) which can be used to interface with the SWYCS IoT gateway system for retrieving real-time device information, modifying device configurations or triggering gateway events. The response of the RESTful API will be in JSON-format, due to the simplicity and human-readable structure.

The web API offers its users to prompt the gateway to execute one of the following actions:

- Putting the gateway in its discovery for a certain amount of time allowing for new devices to join the ZigBee network of the gateway.
- Changing the name of a device that is connected to the IoT gateway.
- Modifying settings of a device that is connected to the IoT gateway (e.g., switching devices on/off or changing the luminosity of a dimmable lamp).
- Removing a device so that it is no longer accessible through the gateway.
- Listing all devices that have joined the ZigBee network of the IoT gateway, including the devices that may not currently be active/connected to the gateway.
- Retrieving real-time data of a device that is connected to the IoT gateway.
- Switching a device that is connected to the IoT gateway on or off (if supported by the device concerned).

3.1.6. **SSH-Client**

As the gateway should be accessible through external devices, the SSH-client application on the gateway allows for external access when the gateway is connected to the Internet. The SSH-client does this by establishing an SSH-tunnel so that access from the DSP servers to each of the registered gateways is possible. This SSH-tunnel can be employed for access to the web API of the gateway and for support by DSP to their customers.

3.2. **Dynamic Structure of the Functional Modules**

With the static structure of the software system at hand defining the components and how they combine to provide the features required of the system, a description of how the system actually works is still needed. This section elaborates on the interactions of the independently operating modules and how each of these modules operate at runtime.

3.2.1. **Historical Data Fetcher**

The role of the historical data fetcher in the system is to store historical data of the connected devices. Therefore an explanation is needed of how the historical data fetcher shall gather and store the required data. The storage of data for all devices connected to the ZigBee network of the gateway shall be triggered by a scheduled task on the operating system (called a) every five minutes. As depicted in Figure 3.2 the first action that is taken by the fetcher is to request a list of all the devices that are currently connected to the gateway. The data for all devices of the same type is stored in the same device table. Therefore the device type of each
device needs to be determined, so that the data for each device is stored in the corresponding table. Subsequently the appropriate data is requested from the device and stored accordingly in the historical database of the gateway. After the data for each connected device is stored in the form of a database entry, it may be retrieved from the database again and used later on by the cloud synchronizer.

3.2.2. Cloud Synchronizer

The cloud synchronizer retrieves the earlier stored data from the database and pushes it to the DSP cloud server. Next to syncing device data the cloud synchronizer also syncs the device statuses of all devices with the cloud server. The syncing of the measurement data and device statuses is triggered by a task scheduled to be executed every 30 minutes by the operating system.
As depicted in Figure 3.3 the first step that is taken is retrieving the device statuses of all the devices that are linked to the gateway. These statuses may correspond to newly joined devices that might not yet be registered at the DSP cloud server. In this case the device will be registered at the DSP cloud server and its corresponding status should be synced. However these statuses may also correspond to devices that were removed but were registered at the cloud server before. In this case the device concerned will be unregistered from the DSP cloud server as the device should no longer be of interest. For all devices that were already registered and not removed the status of the device should be synced with the DSP cloud server. Next to this real-time data of each connected device is sent to the DSP cloud server, which is not depicted in the sequence diagram of Figure 3.3. This data is sent to the cloud server, so that the SWYCS portal can use this cache data when real-time data is requested.

The next step that is taken is retrieving a list of all connected devices. This device list is needed as measurement data of removed devices needs to be filtered out of the data to sync with the cloud server. For each of the devices the new measurement data that has not yet been synced, is retrieved from the corresponding database tables and pushed to the DSP cloud server.

### 3.2.3. Watchdog

The watchdog of the SWYCS IoT Gateway system monitors the network and acts in response to network change events in the ZigBee network of the gateway. The sequence diagram of Figure 3.4 provides a simplified overview of how the watchdog takes care of both devices joining the network and devices leaving the network.

When a device joins the network it can be either a new device that has joined the ZigBee network of the gateway for the first time, or a device that has joined the network again after having left the network. In the first case a new entry should be inserted into the device database with the device info of the new device. Also the new device should be registered at the DSP cloud server so that real-time access through the cloud server will be directly possible. Otherwise if a device joins the network that has left the network earlier, only the connection status of the device needs to be updated both in the local database and at the DSP cloud server.

When a device leaves the network its connection status should be updated in the database so that it indicates the device is inactive/not connected to the gateway at the moment. Subsequently this change in connection status is synced with the cloud server as well, so that it is update of the statuses of all connected devices of the gateway.
3.2.4. **Gateway Web API**

The SWYCS IoT gateway software system should allow for several requests to be send to the gateway in the form of HTTP requests. This section summarizes how the gateway shall respond on the various requests described in Section 3.1.5. Further implementation details of the web API are discussed in Section 4.1.4.

**Putting the Gateway in Discovery Mode**

When a request is made to put the gateway into its discovery mode, the software system will interface with the gateway’s Control Bridge in order to trigger the gateway to permit new devices to join the network. The request can specify the amount of time in seconds the gateway should permit new devices to join the network. If no time is specified, the default discovery time is used which is set to 30 seconds.

**Changing device names**

The system shall allow users to assign the desired descriptive name to each of the devices connected to their gateway. This name is stored in the device database of the IoT gateway and may be changed through a request via the web API. One can specify for which device the name should be changed and which name should be assigned to this device.

**Modifying device configurations**

Several of the devices that can be connected to the gateway allow for modifiable configurations. For example, one might want to change the intensity or color of a lightbulb connected to the gateway. Depending on the device type one can change configurations of all connected devices through a request via the web API. The request will trigger the Control Bridge to send a command to the device concerned, so that its configuration is updated to the users’ wishes.

**Removing devices**

When devices should be no longer accessible through the gateway they can be removed through a request to the web API. The device will then be marked as removed in the device database and measurement data will no longer be stored on the gateway or synced with the DSP cloud server.

**Retrieving a list of all devices**

When a request is made to retrieve a list of all devices, a response will be sent back that includes a collection of device information for each device. Data is retrieved for each entry in the gateway’s device database and therefore retrieves information of connected, disconnected, and removed devices as well. A complete overview is thus provided of all devices that have been connected (at least once) to the ZigBee network of the gateway.

**Retrieving real-time device data**

One can retrieve real-time device data for a device connected to the gateway. Therefore should be specified for which device the data of interest should be retrieved. The response that is sent back includes the appropriate requested data for the specified device.

**Switching devices on/off**

If supported by the device, one can switch a specified connected device either on or off. When the specified device does not support switching, no action will be performed, and a response will be sent back indicating the device does not support the requested action.
After having carefully thought about the design of the software system, it has been implemented according to the architecture design. This chapter describes how the system is implemented and what choices have been made while implementing. In Section 4.1 the overall implementation of the components in the system is discussed. Section 4.2 describes how the components in the system are deployed and set up. Finally, Section 4.3 discusses how the system is cross-compiled for the gateway.

4.1. OVERALL STRUCTURE

The system has multiple components with different functionality. In this section the implementation of each component in the system is discussed, providing a more in depth overview of which choices were made regarding the actual implementation of the system.

4.1.1. CORE / IOT LAYER

The components in the system require some certain general functionality, e.g. message logging, parsing configuration files, communication to the network, communication to a local database. All these functionalities are bundled in a common core, which each component of the system makes use of. In this subsection, the common core components of the system are discussed.

Message Logging

Message logging is important in our system, because events that occur in ZigBee networks can be unpredictable and logging messages can be useful while debugging, but on run time as well. A custom logger has been build, which can log messages with different log levels to the standard output, system log and/or the cloud of DSP. This logger is a singleton, which will be initialized once and can be configured with the desired options (e.g., the log level to apply, the name of the application in the logs).

End Device

The ZigBee devices in the network are represented as EndDevice instances. In these EndDevice instances, it is possible to read and manipulate variables of an EndDevice based on its device type. The end devices in the network are all identified through their unique MAC address. This is also stored as a property in the instance. As ZigBee devices have different device types and properties, it would have been possible to create a basic EndDevice class and make classes for each type of ZigBee devices. Instead device type definitions in JSON format are made for each device type, which describes the properties of each device type. The general EndDevice has now getters and setter based on the key-value principle. In this way, new device types can be easily added to the gateway by adding JSON files without modifying the implementation of the system.
**Network Client**

The network client initializes the communication between the ZigBee network and the different components. It establishes a connection to the border router and is for example able to fetch EndDevices from the network or trigger the discovery mode so that new ZigBee devices may join the network. The settings are fetched from a configuration file, which can be modified to the desired configuration.

**Database Managers**

The database managers arrange the communication with the databases. A division has been made between the historical database manager and device database manager. The SQLite3 database software library has been used to accomplish this due to its zero-configuration, server-less SQL database engine. A method is written for each query that the system would like to execute. The manager also uses the device type JSON files to create tables dynamically for each device type based on its properties.

4.1.2. **Historical Data Fetcher**

The historical data fetcher will be executed every five minutes by a cronjob and fetches all the connected devices in the network. The historical database manager processes the list of EndDevices retrieved from the network and puts the data of the devices with the current timestamp in the corresponding tables for each device type. If a device type is unknown, the gateway will ask DSP for the device type definition of this unknown device. The new device type definition will be automatically added to the collection of known device types in JSON format. It also deletes data records older than 30 days, to prevent shortage of memory storage. Because the data is pushed every 30 minutes to the cloud, the old data should be already pushed.

4.1.3. **Cloud Synchronizer**

To push the data to the DSP servers, the system uses the web API provided by DSP. Therefore the system uses cURL, which is a third-party library for transferring files with URL syntax. The advantage of this library is that it supports HTTPS/SSL, so cURL can ensure that the connection is secure using certificates and that it is only communicating with DSP.

When the cloud synchronizer starts, it first syncs all known devices in the ZigBee network to DSP. For each record in the device database it checks if the field "registered_at" is NULL. If that is the case, the synchronizer will push the new device to DSP and set the timestamp for that moment in the "registered_at" field.

Once all the (new) devices have been registered to the DSP server, all new measurements of devices that have not been deleted will be pushed. Each table of the measurements of each device type has a field named "pushed_at". If there are records where the value of this field is NULL, the records will be pushed. Because each device type has different properties, the system sends the request with the local id as parameter, along with the timestamp of the measurement and the measurement itself (in JSON format corresponding to the device type JSON descriptions). The local ID identifies the device type and the DSP server will handle the request accordingly.

The cloud synchronizer also handles the message logging to the cloud of DSP and will be called by the custom logger from the core (Section 4.1.1). The logger function in the cloud is also specified in the API of DSP.

4.1.4. **Gateway Web API**

To implement the RESTful web API on the gateway, a web server is required which supports scripting languages. OpenWrt already provides a uHTTPd web server and supports cgi, php5, perl and lua scripts. Because the API fetches the data from the IoT Layer, which is written in C++, the most convenient way to communicate with it, is with cgi-scripts written in C++. The standard output of the cgi-program will form the response in JSON format.

The output in JSON format of every API call includes a key-value pair that indicates whether the request was successful. When the request was successful, the value of the success key will be true, and the rest of the JSON structure will be as specified in the API. When the request was not successful, the value of the success key will
be false, and then there will always be an error key with a descriptive message of the error as value. In Listing 4.1, an example response is given of a request to get the real-time values of a color LED lamp.

```
HTTP/1.1 200 OK
{
  "fields": {
    "local_id": 2,
    "time_stamp": "2015-06-09 15:55:42",
    "mode": 1,
    "luminosity": 190,
    "hue": 214,
    "saturation": 78
  },
  "success": true
}
```

4.1.5. Watchdog

The JIP library provides a function to create a "network monitor" thread. When the watchdog starts, it calls this function with a callback method as parameter. When a device joins or leaves the ZigBee network, the monitor thread will call this callback method. The watchdog handles these events in the ZigBee network and saves the states of the devices in the device database and automatically adds new devices. New devices will also be automatically registered at the DSP server. The device database is needed, because the response time to fetch the status of all the devices in a large ZigBee network could take quite some time. Also, it is necessary to know which devices are currently offline but have been in the network before.

4.1.6. SSH-Client

To forward the web port (80) and SSH port (22) of the gateway to the DSP server, an SSH tunnel will be set up between the gateway and the DSP server. The SSH-client makes use of an SSH-utility (already available on OpenWrt) called dropbear. Once dropbear has forwarded these ports of the gateway, a DSP server can access them to access the API on the gateway or the shell of the gateway.

4.2. System Setup and Deployment

This section describes how the components are deployed and set up in the system. The start-up, first run and periodically executed jobs of the system will be discussed in the subsections.

4.2.1. Start up

At start up, two processes will be started at the end of each boot process: the watchdog (Section 4.1.5) and the SSH-client (Section 4.1.6). These processes will run continuously waiting for an event to act on. OpenWrt uses the rd.c system to start-up its programs while booting. It executes start-up scripts of each start-up program with a runlevel specified, where the program will be started once the runlevel has been reached. To run the new components at start-up, scripts for the new components are added in the rd.c folder.

4.2.2. First run

When the gateway boots for the very first time, the gateway needs to register itself at the DSP cloud servers. The system will automatically send a register request when the first attempt to communicate to the server has been made. Using the unique gateway address, which is the MAC address of the gateway, it makes a register request to the DSP server along with its hardware and software version. Once verified by DSP, the gateway receives a unique login token. This is needed to make various requests to the DSP web API. The token will be included in each request as parameter to authenticate itself.
4.2.3. **PERIODICALLY**

The historical data fetcher (Section 4.1.2) and the cloud synchronizer (Section 4.1.3) will run periodically. Therefore the system uses *Cron*, which is a time-based job scheduler in Unix systems. Once a cronjob has been configured, the associated shell command will be executed when the configured time has passed. In the case of the gateway, the historical data fetcher and the cloud synchronizer will be executed each 5 and 30 minutes respectively.

4.3. **GATEWAY CROSS-COMPILATION**

The gateway runs on an ARMv5 based processor, which is a different architecture than on the build platform where the software is developed. Due to the limited processing power and storage on the gateway, it was not possible to compile the code on the gateway itself. The code of the gateway is therefore cross-compiled using the *arm-linux-gnueabi* compiler on the build machine.

To manage the internal and external dependencies for compilation for the build machine or the gateway, the build system *CMake* has been used. It generates make-files based on the compiler independent configuration files. These configuration files are highly customizable and states how targets are going to be build and with which dependencies.

The components from the common core discussed in Section 4.1.1 are compiled as an object library. This library will be linked at compile-time to each of the components in the system. Other external libraries are dynamically linked with the shared libraries compiled for the gateway.
In this chapter the different aspects of the software quality will be elaborated. In Section 5.1 the engineering of qualitative code during the project is discussed. This code quality depends on different metrics, dependencies, sizes and formatting. In Section 5.2 the feedback on the code, received from the Software Improvement Group (SIG) is evaluated. Section 5.3 describes the documentation of the source code. What code and how is the code documented? In Section 5.4 the tests and testing process is discussed. What source code is tested and how is it tested? Finally Section 5.5 explains the security aspect of the software. It discusses how secure the software is and what is done to prevent the system from being intruded or hacked.

5.1. Code Quality Engineering

In this section we discuss the quality of the code. There are several aspects that contribute to the code quality. The following aspects will be discussed: the source code metrics, the design principles to be followed to reach a good score on these metrics, the readability of the code and logging as debug tool.

Code of good quality is necessary for companies who want to maintain or extend the software in the future. SIG, defined six source code metrics based on the ISO/IEC 9126, focused on software maintainability (Baggen, Correia, Schill, & Visser, 2012), namely:

- **Volume**: a small system takes less effort to understand and maintain, since there is less information to take into account compared to a large system.
- **Redundancy**: changes to duplicate code need to be done at multiple places.
- **Unit size**: big units need more effort to understand in a way of working.
- **Complexity**: less complex systems are easier to understand.
- **Unit interface size**: high number of parameters probably indicates bad encapsulation.
- **Coupling**: interdependence between components, high coupling means high resistance to changes.

One of the goals was to develop good maintainable software. During the development these six metrics were taken into account. There has been attempted to keep the total volume of the system small. One example of what is done to achieve this, is to only write one class that represents all the devices instead of writing a class for each of the devices, because lot of different devices exist within the protocol specification. Where possible redundancy is avoided and duplicated code is deleted and moved to one module. The size of the units are kept small by splitting bigger units in multiple smaller ones, which also reduced the complexity of the units most of the time. To keep the size of the unit interfaces low, units with five or more parameters are avoided and split up into multiple units. Coupling between components is kept small by following design principles explained in the next paragraph. During the project we got feedback from SIG twice about the maintainability metrics. The evaluation and processing of this feedback is explained in Section 5.2.

In Object Oriented Software, the SOLID design principles contribute to a good outcome of those metrics. SOLID includes the following principles: Single Responsibility, Open-Closed, Liskov Substitution, Interface segregation, Dependency inversion. The following are applicable in our system: Single Responsibility, which means that one component only has one task. This principle reduces the complexity. Open-Closed means that the source code should be open for extensions, but closed for modifications. A good example of this principle applied to the source code is the EndDevice class, described in Section 4.1.1, which is extensible.
with new devices, without changing the current code. Dependency inversion means that components should depend upon abstractions and not abstractions upon concrete implementations.

A second important aspect of code quality is readability and the format of the source code. It is easy to understand own written code, but when working in teams and writing code that others need to understand, it is good practice to use one code style in the whole project, so it is easier to read for all the project members. To reach this goal we used Vera++. Vera++ is a programmable tool for verification, analysis and transformation of C++ source code (Verateam, n.d.). Vera++ is run manually, parses all the code, and gives warnings about unusual alignments, statements that are too long and other conventional style rules. The complete list of parsing rules can be found at: Available Vera++ Rules. Solving those warnings make sure, that the code is in general and readable format.

Another tool that is used is Cppcheck. Cppcheck is a static analysis tool that is run at compile time. This tool detects syntax errors and type of bugs that compilers normally do not detect. This useful tool is an addition to the compiler and is run at compile time. Some features of Cppcheck: out of bounds checking, memory leaks checking, detecting possible null pointer dereferences, checking for uninitialized variables, checking for invalid usage of the STL, checking exception safety, warning if obsolete or unsafe functions are used, warning about unused or redundant code and detecting various suspicious code indicating bugs (Cppcheck - A tool for static C/C++ code analysis, n.d.).

Unlike other programming languages, C++ does not provide a stack trace when an exception occurs. Due the absence of this stack trace debugging is hard. A configurable logger class (discussed in detail in Section: 4.1.1) is used to log many useful messages at different log levels. When an error occurs, it is easy to follow the trace from the crash back to the initialization of the program. This is used at debug and release time to trace bugs.

5.2. SIG Code Evaluation

During the project the source code is evaluated two times by SIG. From the first submission, SIG evaluated the source code and concluded that the code has an average score for maintainability. The highest possible score was not achieved due a lower score for Duplication, Unit Size and Module Coupling. The complete feedback for this submission can be found in Section B.1.

The first feedback from SIG is evaluated and processed. A common notice was that there was no clear distinction between company code and external code. All external code is moved to the ext folder and the includes of external code are re-factored. The first aspect that lowered the score, was some code duplication in the CGI `.cpp`-files. These network-related parts that were duplicated, had to be utility methods. A `network_utility` is introduced and the duplications are removed. The second aspect that lower the score was the unit size of some methods. For example the `DeviceDatabaseManager::deviceJoinNetwork`-method contained a comment like `// Check if creating the query has succeeded and execute the query` and this indicates that there is an autonomous piece of functionality. Fragments like these are traced and split up in multiple methods. The last aspect that lowered the score was module coupling. Modules which are often invoked might be split up in different modules to increase stability. The `NetworkClient` was a relative large module and contained various functionalities. Some methods are moved to the `network_utility` module.

After re-factoring and extending the code, a second submission was done. According to SIG the average code score is increased from 3 stars to almost 4 stars. The complete feedback for the second submission can be found in Section B.2. Especially code duplication is decreased. The unit sizes are also decreased, but new large methods are added. Probably the `CloudSynchronizer` is mentioned here, but a clear explanation can be found when we take a look at this class. A lot of methods build a `Map`, which contains all the data for the post or put request to the API. Inserting a variable takes one line of code. An example is depicted in Listing 5.1. Building this map (line numbers 6-10) is different for each API call and cannot be abstracted to one single method. Also the error handling for each API call is different (line numbers 18-26) and therefore cannot be abstracted and moved to one method. This makes the methods relatively large, while it is hard to split them up.

The distinction between company and external code is much more clear now. The test code is still too small, but it is hard too test, because much code depends on the SQLite3 and JIP library. A detailed explanation of the testing is given in Section 5.4.
```cpp
/**
 * Registers the gateway at the DSP cloud server.
 */

int CloudSynchronizer::gatewayRegister() {
  //post values
  std::map<std::string, std::string> post;  
  post.insert(std::pair<std::string, std::string>("system_id", getUniqueGatewayAddress()));
  post.insert(std::pair<std::string, std::string>("software_version", SOFTWARE_VERSION));
  post.insert(std::pair<std::string, std::string>("hardware_version", HARDWARE_VERSION));
  post.insert(std::pair<std::string, std::string>("timezone",
    ConfigReader::getInstance()->getString("Cloud.Timezone", "Europe/Amsterdam")));

  //do request and parse result
  Logger::getInstance()->informational("Registering gateway...");
  Json::Value root = postRequest("gateway/registration", post);
  if((root.compare(Json::Value())) == 0)
    return -1;

  if (root.get("success", false).asBool()) {
    login_token_ = root.get("login_token", "").asString();
    ConfigReader::getInstance()->writeString("Cloud.Login_Token", login_token_);
    Logger::getInstance()->informational("The gateway is successfully registered at the DSP Cloud Server.");
    return 0;
  } else {
    Logger::getInstance()->critical("Could not register the gateway at the API. Error: %s",
      getReadableJsonError(root).c_str());
    return -1;
  }
}
```

### 5.3. **DOCUMENTATION**

Documenting source code is important for the maintainability of software, because it will be easier for non-authors to extend the source code in the future. For a software engineer it is easier to understand functions with a short description and an explanation of the parameters and return value. All functions of the system are well-documented in Javadoc style. The documentation is generated by Doxygen, the standard tool for C++ to generate documentation from Javadoc- and Qt-style documented source code. A snapshot of the Doxygen documentation is depicted in Figure 5.1a. The RESTful API which is described in Section 3.1.5 is documented with apiDoc, an inline documentation tool for RESTful web APIs(apiDoc - Inline Documentation for RESTful web APIs, n.d.). This documentation can be used by DSP Innovation to connect their cloud portal to the gateway. A snapshot of the apiDoc documentation is depicted in Figure 5.1b. The complete documentation of the software consists of the Doxygen documentation, the apiDoc documentation and the description of the architectural design and system implementation, which are part of this report (Chapter 3 and 4).

### 5.4. **TESTING**

To deliver high qualitative code it is important to test the code. Unit Testing is chosen to test the different modules of the code. Unfortunately it was not possible to mock all our classes, because of the dependencies on shared C libraries, which could not be mocked. All the database-related classes depend on the sqlite library, the CloudSynchronizer depends on the cURL library and the NetworkClient depends on the JIP library. Therefore these classes could not be easily tested using conventional unit tests.

Another important thing to take into account is that in C/C++ memory management is done by the programmer. The programmer needs to reserve memory when needed and free the memory if it is not used anymore. When a programmer forgets to deallocate used memory, memory leaks may occur. To avoid memory leaks,
Valgrind is used to track this kind of bugs. Valgrind is a tool that runs programs under Memcheck’s supervision. During the check, all read and write operations to the memory are checked. It detects wrong memory accesses, wrong memory frees, uses of uninitialized variables, memory leaks, bad frees, and more (Valgrind - Valgrind’s Tool Suite, n.d.). Using this tool in an appropriate way makes sure that the software contains no memory leaks.

5.5. Security

Security is an important aspect of the quality of the code. Different security aspects are taken into account. The first one is user input. A lot of database queries are executed, some with direct and some with indirect user input. All variables that are put into the queries are bound with the sqlite3_bind function, provided by the sqlite3 library, to make sure that all (user)input is escaped and no SQL-injections are possible.

The second aspect is the communication with the DSP Innovation cloud servers. The API server supports an HTTPS connection which is a fully encrypted connection. The CloudSynchronizer is the only module that makes calls to this server. It is forced to check the server’s certificate by enabling the option verify_peer and providing the corresponding certificate authority bundle. This makes sure that no man-in-the-middle-attack can take place without knowing the server’s private key. Communication in the other direction, from server to gateway, is done through a SSH tunnel which is set up by the gateway. This SSH connection is public-key authenticated and therefore not vulnerable to man-in-the-middle attacks (abb, 2010).

The last part of the security aspect is the communication between the devices and the gateway. The ZigBee HA stack has a build-in AES-128 encryption (Table 2.1), so there is no need for additional security in the application layer.
5.5. SECURITY

(a) Doxygen documentation: The source code was fully documented allowing a complete documentation to be generated.

(b) apiDoc documentation: A documentation was composed for the developed web API of the IoT gateway.

Figure 5.1: Documentation of the SWYCS IoT gateway software system.
In this chapter the development process of the IoT gateway software is being discussed. In Section 6.1 the evaluation of the Scrum methodology during the project is done. Section 6.2 describes the development tools which are used. And finally in Section 6.3 the encountered issues, and the solutions the project team came up with are discussed.

6.1. Agile Development Reflection

During the project, a light-weight variant of Scrum is used as agile software development methodology for managing the product development. One-week sprints are done. The sprint planning was done each Tuesday, where sprint tasks are extracted from the product backlog and assigned to the members of the development team. A daily Scrum was done all office days to inform each other about the progress of the tasks for that sprint. An example of a sprint board is depicted in Figure 6.1.

![Sprint 1 - Task board.](image)

The first two sprints Scrum did not work out very well. The gateway and the essential libraries were a completely new concept for the team. Tasks were underestimated and there were too much dependencies between tasks, which resulted that tasks of different members could not be done completely parallel. Once the team was used to the gateway and knew which libraries could be used, Scrum became a very helpful and handy tool. From the third Sprint and on Scrum could be effectively applied.
6.2. Development Resources

Because there was no whiteboard available that could be used the entire project, an online software package called Trello is used to maintain the progress of the Scrum sprints. This gave the project team the possibility to gain insight into the process of the tasks of each of the developers.

For version control Github is used, since the company reserved some private repositories for the project. Git is useful when there is an emphasis on speed, data integrity, and support for distributed, non-linear workflows.

6.3. Encountered Issues

During the project, there was an issue encountered regarding the hardware that should be delivered by DSP Innovation. At the beginning of Sprint 3, the company’s smartplug and CMU were still not delivered. Only the lamps and development boards were available at that moment. This issue is solved by buying some third-party smartplugs and temperature/light/humidity sensors, which are depicted in Figure 6.2.

Detailed manuals with all technical and software aspects were delivered together with the devices. Unfortunately there were some issues. The project team quickly succeeded in switching on and off the smartplug, but the measurement cluster of the plug is not part of the Home Automation stack. Instead of HA, the plug uses the ZigBee Smart Energy Stack for sending measurement to the gateway. After some research and contact with the product owner, the decision is made to partial disregard the smartplug for the project.

Another issue that was encountered had to do with the multisensor. The gateway was able to read the sensor, but requests to the device took about seven seconds. The device does contain a battery instead of having constant power, therefore ZigBee low energy profile was used and that causes slow responses. The control bridge was not prepared for these slow devices, because it can not execute parallel requests to the ZigBee network and other requests had to wait seven seconds when the multisensor was read. To solve this issue, the control bridge had to be partially rewritten, but there was not enough time to do that during the project. Therefore the multisensor is disregarded for the project.

![ZigBee Smartplug](a) ZigBee Smartplug

![Zigbee Multisensor](b) Zigbee Multisensor

Figure 6.2: Third-party ZigBee devices.
7

**Prototype Evaluation**

7.1. **Evaluation of Requirements**

As discussed in Section 6.3, unfortunately the prototype do not meet all the requirements. The requirements we did not met were related to the lack of hardware, which has been taken into account as risk as stated in the project plan and in Section 1.3. In cooperation with the client, some third-party ZigBee Home Automation devices could be ordered, so that the project still could move on and make up for the missing hardware. Unfortunately, neither the smartplug and the multisensor did meet the expectation of the team of how the products should have worked.

The measuring cluster of the smartplug is part of the ZigBee Smart Energy profile, which does not fall under the scope of this project. However multiple attempts have been made to still make it work for the gateway. After research, the Smart Energy profile requires an additional layer security on top of the default security of ZigBee PRO. Security certificates and link keys from official certificate authorities are needed for the additional security layer and to fetch the measuring data extra securely. It would require a lot of extra time to make the measurement cluster work which we did not have given the limited time of the project.

The multisensor has great features and sounded very promising. Due to the reduction of energy consumption of the battery, the responses times of the sensor were too slow for the control bridge on the ZigBee chip to handle. We tried to solve this problem on the control bridge by increasing the timeout value. Unfortunately the response times were that slow, that waiting for the response to return blocks the whole main thread and the control bridge could not handle other messages anymore. While planning this project, it has not taken into account that the control bridge had to be modified or to be rewritten. The expectation in the begin of the project were that the ZigBee Control Bridge should work without modification for every ZigBee device. Therefore the control bridge did not fall under the scope of the project and it would cost a lot of extra time besides the time for the project.

The issues of both devices mentioned above has been discussed with the client and has been disregarded for the project.

However none of the functionalities of the prototype has been compromised due to the lack of Home Automation hardware. The main goal of this project is to create a platform on the gateway that lets various types of Home Automation devices easily join. Also compatibility for new types of Home Automation devices should be easily added to the platform. The prototype is currently only extensively tested with mono lamps and colour lamps, but it is designed to be compatible with every ZigBee Home Automation device. By simply adding a JSON file with the properties of a new device type (Section 4.1.1) the platform will instantly support it without modifying the system.

The last requirement that was not met, is the requirement that the Gateway could pair with ZigBee devices by using NFC technology. Unfortunately, neither the gateway nor the ZigBee devices used in this project had NFC capabilities.
7.2. Evaluation of Design Goals

In the research phase of the project, three design goals have been set for this system (Section 1.1). The three design goals were: Security, Maintainability and scalability. In this section, the prototype is evaluated regarding the design goals.

The first main design goal was Security. During implementation of the system, there has been a lot of attention payed to making sure the system and all its connections were secure. The connections on the ZigBee network were already secure due to the build-in encryption in the protocol stack, so there was no need to pay attention there. The communication from the gateway to the DSP server is encrypted by HTTPS/SSL connections and the communication from the DSP server to the gateway goes through the established SSH tunnel. Also the prevention of SQL injections has been implemented on the API of the gateway, therefore no unauthorized access can be granted for intruders. Besides communication, the access to the gateway is restricted for unauthorized individuals. Reading and modifying data on the gateway is not possible by others. Therefore this goal has been reached in this project.

The second design goal was Maintainability. To let the system be maintainable, the system is divided into different components. Each component is modular independent and is well documented. Modifications to one component does not lead to modifications in other components. However, during development there has been aimed to develop a software system that is closed for modification, but open for extension. There should be no need to alter the existing source code to add new functionality to the current system. Therefore this goal has been reached in this project.

The last design goal was Scalability. In the developed system, the support of new device types can be easily added by the gateway (Section 4.1.1) and the gateway should be able to handle every IoT device that has ZigBee Home Automation. However, during development it turned out that the control bridge is running every request sequential (Section 6.3). The development or modification of the control bridge did not fall under the scope of this project, so we were not able to improve the control bridge given the limited time for this project. Due to the scarcity of Home Automation devices, it was not possible to test the scalability and the performance of the gateway with a large amount of devices in the network. Our expectation is that the gateway could scale up but limited. However, the system for the gateway is ready to scale and it only depends on the performance of the control bridge how much it could scale. Therefore this goal has been reached in this project.

7.3. Overall Evaluation

Although the prototype did not meet all the requirements due to scarcity of hardware and the some unexpected shortcomings of the control bridge, the prototype is definitely a success. The main challenge of this project was to develop a generic solution to couple different (third-party) ZigBee devices to the gateway. With that in mind, the system has been designed to easily adopt new device types without altering the original system implementation. Also the main design goals security, maintainability, scalability have been maintained during the development. The software system on the gateway is ready to handle lots of ZigBee devices on the network and to adopt new device types with ease.
FUTURE WORK AND RECOMMENDATIONS

Although a lot of work is done, the gateway is not production-ready yet. As finishing touch, some additional features should be integrated before the gateway can be sold to customers. First of all an update process should be added, which can update software on the host and the ZigBee Control Bridge, so software on the gateway can always be updated by the DSP Innovation cloud servers.

Another feature that should be added is the support for DSP Innovation’s own hardware, namely the Smartplug and CMU, which could not be integrated during this project (as described in Section 6.3). One of the goals of DSP Innovation is to connect as many devices as possible to the gateway, so it is recommended to integrate more popular ZigBee devices, which can be found at ZigBee Alliance Product List.

Once the gateway is production-ready, DSP Innovation should integrate the new gateway with its portal. The web portal at swycs.com should be extended to support lamps, sensors and other devices that are supported by the gateway.

One last recommendation is to rewrite the default software that interfaces with the control bridge. As explained in Section 6.3, the bridge could not perform parallel requests. Some devices have high response times and requests to them may take several seconds, which blocks all other requests on the bridge. The bridge should be adapted so it can process multiple requests at the same time. Even if we think away slow-response-devices, it is not desirable to have a sequential processing bridge. Think of an environment with, for example 50 smartplugs, which are read at this moment by the historical-fetcher. Another person tries to switch one of the smartplugs. Then there is a chance that first all smartplugs are read and afterwards the smartplug switches. Sequential processing introduces big delays and should be avoided.
Before this project, DSP Innovation B.V. had developed a second generation gateway with the goal to support as many IoT devices of third parties as possible. A generic software solution was needed to couple different (third-party) ZigBee Home Automation devices to the new gateway. The need for a software system emerged that was secure, maintainable and scalable. The project team gained the unique opportunity to develop and deliver a working prototype for DSP Innovation B.V.

When the team started to work on this project, two weeks of research were conducted to the problem, the gateway and different solutions. It became clear that developing this system would require much knowledge of the multiple aspects around the system and within the current system itself. Understanding lots of details within the context of the whole project was important and was very educational. An architecture design has been made with all the requirements and design goals in mind.

During six weeks of developing and testing the software system, the team has stumbled upon multiple unexpected challenges. Due the scarcity of Home Automation hardware and unexpected shortcomings of the control bridge, it was not feasible to meet some requirements. However, none of the required functionalities on the prototype have been compromised.

At the end of the project, a fully functional software system on prototype has been delivered. The system has been made generic due to dynamic device type definitions, which can be easily added without altering the current system. The software system on the gateway is ready to handle lots of ZigBee devices on the network and to adopt new device types with ease. Even though the prototype is not production-ready yet, the project was definitely a success.
A.1. **Project Description**

Op korte termijn zijn er meer slimme zelf acterende apparaten aan het internet verbonden, dan mensen die via het internet apparaten bedienen. Deze snelle ontwikkeling heet IoT (Internet of Things).

Een voorbeeld IoT project heeft DSP innovation bv al ontwikkeld in de vorm van haar SWYCS (See What You Can Save) energiemanagement oplossing. SWYCS meet realtime en online energieverbruik van gas, water en elektra, en koppelt dit aan andere informatiebronnen zoals lichtsterkte, temperatuur, opwekking van energie, enz. Hierdoor worden apparatuur automatisch gemanaged teneinde efficiënter om te gaan met energie.

DSP innovation bv heeft een 2e generatie hardware ontwikkeld om zoveel mogelijk IoT apparatuur ook van derden te ondersteunen.

De uitdaging in dit project is om via de nieuwe generatie IoT gateway op generieke wijze allerlei ZigBee apparatuur (van derden) te koppelen. Denk hierbij aan thermostaatknoppen, thermostaten, rolluiken, draadloze schakelaars, color LEDs, etc.

De studenten doen onderzoek naar de verschillende bestaande ZigBee apparaten op de markt. Het doel van dit onderzoek is om erachter te komen hoe deze verschillende apparatuur communiceert en welke informatie zij uitwisselen en hoe dit geïmplementeerd kan worden als API op de gateway. Daarnaast zal een werkend prototype moeten worden opgeleverd. Dit houdt in dat de door DSP aangeleverde IoT apparaten, gekoppeld aan de nieuwe IoT gateway volledig functioneren.

A.2. **Company Description**

DSP innovation BV is een onderneming die op basis van innovatieve energie en communicatieconcepten, ondersteund door web portalen, energiebesparing bewerkstelligt en een communicatieplatform biedt. DSP Innovation bv is een jonge, dynamische, maatschappelijk betrokken onderneming met passie voor innovatieve oplossingen.

DSP innovation bv heeft een energiemanagement en communicatieoplossing ontwikkeld dat onder de naam SWYCS (See What You Can Save) projectmatig op de zakelijke markt wordt uitgebracht.

SWYCS is een realtime en online energiemanagement oplossing. Gas, water, elektra, licht, luchtvochtigheid en temperatuur worden realtime gemeten en gemonitord. Apparatuur en processen kunnen waar ter wereld online worden gemonitord, ingeregeld en/of bijgestuurd.

De SWYCS oplossingen zorgen voor:

- Allocatie van energiestromen en -kosten
- Kostenbesparing voor de organisatie
A. Project Information

- Verbeterd inzicht in processen
- Bijdrage aan milieudoelstellingen en MVO
- Verbeterde communicatie met medewerkers, huurders en/of andere stakeholders

Door middel van het IoT platform van DSP innovation bv is het mogelijk om naast de SWYCS ook meetapparatuur, sensoren of andere hardware/apparatuur van derden te ontsluiten en aan te sturen.

A.3. Auxiliary Information

DSP stelt ten minste 1 gateway beschikbaar en de volgende ZigBee apparaten: smartPlug, pulsteller, color LED, en een thermostaat.

De studenten krijgen een vergoeding van 200 euro per maand.
B.1. Feedback First Submission

De code van het systeem scoort drie sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code gemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Duplication, Unit Size en Module Coupling.

Het eerste wat opvalt is dat het lastig te bepalen is welke code door jullie zelf is geschreven. Zo lijkt het erop dat er code van sqlite3, een JSON parser en een JIP library aanwezig is. Deze code staat naast de code die door jullie zelf is geschreven. Om problemen in de toekomst te voorkomen is het goed om duidelijk te documenteren welke code aangepast moet worden en welke code van externe partijen is, bijvoorbeeld door externe code in een aparte directory te plaatsen.

Voor Duplicatie wordt er gekeken naar het percentage van de code welke redundant is, oftewel de code die meerdere keren in het systeem voorkomt en in principe verwijderd zou kunnen worden. Vanuit het oogpunt van onderhoudbaarheid is het wenselijk om een laag percentage redundantie te hebben omdat aanpassingen aan deze stukken code doorgaans op meerdere plaatsen moet gebeuren. In dit systeem is er duplicatie te vinden tussen de verschillende `.cpp` bestanden binnen `src/cgi/src/device/`. Delen van deze files lijken utility methoden te bevatten die ook op een enkele plek gedefinieerd kunnen worden. Het is aan te raden om dit soort duplicaten op te sporen en te verwijderen.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de langere methodes in dit systeem, zoals bijvoorbeeld de `DeviceDatabaseManager::deviceJoinNetwork`-methode, zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes. Commentaarregels zoals bijvoorbeeld `// Check if this device already has a local id in the database` en `// Check if creating the query has succeeded and execute the query.` zijn een goede indicatie dat er een autonoom stuk functionaliteit te ontdekken is. Het is aan te raden kritisch te kijken naar de langere methodes binnen dit systeem en deze waar mogelijk op te splitsen.

Voor Module Coupling wordt er gekeken naar het percentage code dat relatief vaak wordt aangeroepen. Normaal gesproken zorgt code die vaak aangeroepen wordt voor een minder stabiel systeem omdat veranderingen binnen dit type code kan leiden tot aanpassingen op veel verschillende plaatsen. In dit snapshot is bijvoorbeeld de `NetworkClient` relatief groot. Nadere inspectie wijst uit dat deze file specifieke netwerk functionaliteit bevat (e.g. `monitorNetwork`), maar ook utility functionaliteit (e.g. `getIPAddress`). Om zowel de grootte als het aantal aannroepen te verminderen zouden deze functionaliteiten gescheiden kunnen worden, wat er ook toe zou leiden dat de afzonderlijke functionaliteiten makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden worden.

Over het algemeen scoort de code gemiddeld, hopelijk lukt het om dit niveau te behouden of te verbeteren tijdens de rest van de ontwikkelfase. De aanwezigheid van test-code is in ieder geval veelbelovend, hopelijk zal het volume van de test-code ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt.
B.2. Feedback Second Submission

In de tweede upload zien we dat zowel het codevolume als de score voor onderhoudbaarheid zijn gestegen. Jullie scoren nog steeds 3 sterren, maar jullie zitten nu dicht tegen de 4-sterrengrens aan.

Van de drie deelscores die tijdens de eerste analyse werden aangemerkt als verbeterpunt zijn jullie vooral op Duplication vooruit gegaan. Bij Unit Size zien we ook verbeteringen, maar die zijn grotendeels weer teniet gedaan omdat jullie nieuwe code weer een aantal grote methodes kent.

Het is goed om te zien dat jullie de code-indeling hebben aangepast, waardoor nu een stuk duidelijker is welke code door jullie wordt onderhouden.

Wat betreft de testcode zien we wel een stijging in de hoeveelheid, maar de testcode blijft nog steeds meer dan een factor tien kleiner dan de productiecode. Echt test-driven werken is hierdoor niet mogelijk, waardoor de voordelen van unit tests niet echt benut worden.

Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie deels zijn meegenomen in het ontwikkeltraject.
Title of the project: One world is not enough
Name of the client organization: DSP Innovation B.V.
Date of the final presentation: 26-06-2015

Description
This bachelor project is commissioned by DSP Innovation B.V., a company that uses innovative energy and communication concepts, supported by web portals, as a basis for achieving energy saving and offering a communication platform. DSP Innovation B.V. is a young, dynamic, socially involved company with passion for innovative solutions.

The goal of this project was to develop, with the new generation IoT (Internet of Things) gateway, a generic software solution to couple several different (third-party) ZigBee devices. Some examples might be thermostat knobs, thermostats, shutters, wireless switches, dimmable LED lights, color LED lights, etc. Coming to this ‘generic solution’ was the most challenging part of the project. During the research phase, the different communication protocols and ZigBee devices have been analyzed. For the current application, the ZigBee Home Automation stack was found to be the most suitable solution. Scrum was used as software development methodology to manage the process. Especially in the first two weeks it was hard to apply Scrum, since the required effort of the tasks was hard to estimate due to the fact we had no experience in working on an embedded platform and therefore it took some time to setup the project. The final product is an operational IoT gateway, which can be controlled by a RESTful API. The gateway can discover ZigBee devices, store historical and retrieve realtime measurements of the devices and synchronize these measurements with the cloud. The gateway connects via a Secure Shell to the cloud server, so it can be accessed by the company’s web portal. The gateway is a prototype and therefore is not production-ready yet. A software update process must be integrated and support for more devices should be added.

Members of the project team
Rob van Bekkum
Role and contribution: Scrum Master, All-round developer, API, Database-related.

Jason van Belzen
Role and contribution: All-round developer, Database-related, Cloud-related.

Victor Li
Role and contribution: All-round developer, API, ZigBee-related.

Contact information
Name and affiliation of the client: Mr. R. Riemens, DSP Innovation B.V.
Name and affiliation of the coach: Dr. R. R. Venkatesha Prasad (VP), ES Group, TU Delft
Contact person 1: Rob van Bekkum, rob_van_bekkum@hotmail.com
Contact person 2: Jason van Belzen, jasonvanbelzen@gmail.com
Contact person 3: Victor Li, victor_li1994@hotmail.com

The final report for this project can be found at: http://repository.tudelft.nl
ACRONYMS

API  Application Programming Interface. 3, 8, 19, 24
CMU  Central Measurement Unit. 5, 15, 37
IoT  Internet of Things. 1, 4, 7, 17, 33
JIP  Jennet IP. 18, 28
JSON  Javascript Object Notation. 19, 23, 35
MIB  Management Information Base. 12, 18
NFC  Near Field Communication. 4, 35
REST  Representational State Transfer. 19, 24
SDK  Software Development Kit. 16
SIG  Software Improvement Group. 27
SQL  Structured Query Language. 24
SSH  Secure Shell. 6, 16, 19, 25, 30
SWYCS  See What You Can Save. 1, 7, 17
**Glossary**

**cronjob** A periodically run task by the Cron utility, which is a time-based job scheduler. 19

**cURL** A shared C library that can perform HTTP requests. 24, 29

**daemon** A daemon is a program that runs a background task. 18

**Github** Github is an online version control system that uses Git. Git is useful when there is an emphasis on speed, data integrity, and support for distributed, non-linear workflows. 34

**HTTPS** The HTTP protocol with a Secure Socket Layer (SSL) on top of it. 30

**JenNet-IP** JenNet-IP is an IP-based networking protocol that enables 'Internet of Things' devices to connect to all Smart Devices in the home and other buildings. 7

**OpenWrt** A Linux based firmware for embedded devices like routers and gateways. 24

**Scrum** Scrum is an interactive, incremental and agile software development method. 33

**SQLite3** SQLite3 is a zero-configuration, serverless Structured Query Language Database library, which is often used at embedded devices. 24, 28

**stack trace** Report of the active stack frames in a program during the execution. Each subroutine in a program has its own stack frame containing its local variables. 28

**Trello** Trello is an online organization tool that organizes projects into flexible boards. 34

**Unit Testing** A software testing method by which each individual unit of source code is tested. 29

**ZigBee** ZigBee is a low-power, wireless mesh network standard. This standard is targeted at wireless control and monitoring applications. 1, 3, 7, 18, 23, 27, 34, 35

**ZigBee Control Bridge** The JN5168 chip in the gateway that is programmed as network coordinator. The control bridge is serially connected to the gateways CPU. 18, 35, 37
REFERENCES


