FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

KNX to MQTT/AMQP

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Felix Walcher
Matrikelnummer 01427555

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Ao.Univ.Prof.Dr Wolfgang Kastner

Wien, 3. März 2019

Felix Walcher	Wolfgang Kastner

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

KNX to MQTT/AMQP

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Felix Walcher
Registration Number 01427555

to the Faculty of Informatics
at the TU Wien
Advisor: Ao.Univ.Prof.Dr Wolfgang Kastner

Vienna, 3rd March, 2019

Felix Walcher	Wolfgang Kastner

Erklärung zur Verfassung der Arbeit

Felix Walcher
Kaserngasse 28, 1230 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 3. März 2019

Felix Walcher

Danksagung

Herzlichst bedanke ich mich bei Herrn Professor Kastner für die richtungsweisende und tatkräftige Unterstützung und der das Gelingen meiner Bachelor Arbeit ermöglicht hat. Bedanken möchte ich mich auch bei denjenigen, die mich während meines Studiums begleitet und unterstützt haben.

Acknowledgements

I want to thank professor Kastner for the indicatory support during my work and the possibility for this thesis. Furthermore I want thank everybody who supported me during my study days.

Kurzfassung

Ziel dieser Arbeit war es, die Kommunikation zwischen Geräten aus dem IoT (Internet der Dinge) und Gebäudeautomationssystemen mithilfe eines IoT-Brokersystems zu ermöglichen. Der erste Teil der Arbeit gibt einen groben Überblick über Gebäudeautomation und dem IoT. Weiters werden MQTT und AMQP, zwei der meist verbreiteten IoT Kommunikationsprotokolle, vorgestellt und miteinander verglichen. Außerdem wird KNX vorgestellt, ein internationaler Standard für Gebäudeautomationssysteme. Der zweite Teil der Arbeit besteht aus der Dokumentation des entworfenen und implementierten IoT-Brokersystems.

Abstract

The aim of the work was the design and implementation of an IoT protocol broker in order to enable communication between IoT devices and building automation systems (BAS). The first part gives a short overview on BAS and the IoT. Furthermore, MQTT and AMQP, two common IoT communication protocols are discussed and compared. Next, a short overview of KNX is given, an international standard communication system for BAS. Finally, the work presents a documentation of the implemented IoT broker.

Contents

CHAPTER 1

Introduction

By 2020, more than 25 billion devices are expected to be connected through the Internet of Things (IoT) [1]. The IoT expected to find usage in smart infrastructure, traffic management, healthcare or logistics. In the sector of smart infrastructure, there is a trend toward intelligent homes and buildings. These systems require a high device integration and a high degree of communication [2].

Because of the increasing usage of the IoT, it is important to integrate building services into the IoT. The underlying building automation systems (BAS) could benefit from this integration by offering new features. For example, it could improve maintenance and be useful for the realization of applications related to smart grids (e.g. demand side management). Furthermore, it enables data monitoring for IoT devices [3].

This bachelor thesis shows a simple integration of a BAS into the IoT. MQTT and AMQP, two very common IoT communication-protocols are discussed and compared. Furthermore, there will be a short overview of KNX, a communication system used for BAS. The second part is directed to the design and implementation of a Java application, that bridges a KNX system and the IoT with MQTT and AMQP as communication protocols.

CHAPTER 2

Building Automation Systems and the IoT

2.1 Building Automation Systems

A building automation system (BAS) is a system installed in a building that is responsible for the controlling and monitoring of building services [4]. In large functional buildings, typical services for BAS are, for example, heating, cooling, air conditioning, ventilation, lighting and shading, life safety and alarming. The benefits of the use of BAS are the reduction of energy consumption costs, simplification of building operations, monitoring and better maintenance. For example, the information of temperature, CO_2 concentration or building occupancy influence the building control in order to reduce resources and maintain comfort. Costs can be also downsized by centralized monitoring and control centres [3]. With the help of these centres, faulty conditions can be detected in an early stage with less personnel effort. Though BAS will result in higher construction and engineering costs, the long-term benefits result in a better economic behaviour. Because of this, it is important to choose a building concept that ensures the optimal life-cycle costs contracting the concept with the lowest initial costs.

2.1.1 Architecture

BAS are distributed systems that are orientated on a computerized control and management of building services. The architecture of a BAS can be organized by three layers (Figure 2.1).

Field Layer:
The Field Layer is the lowest layer where the interaction with the physical world takes place. Sensors are collecting environmental data, for example, temperature values which get transformed into a processible representation. On the other hand, actuators influence

the behavior of the physical process (e.g. thermal or visual control).

Middle Layer:
The Middle Layer is also known as the Automation Layer of the system, where all kinds of autonomous tasks are done with the prepared data received from the Field Layer.

Top Layer:
In the Top Layer or also called Management Layer, information through the entire system is accessible. Values of the Middle Layer are accessible in order to change parameters like for example for schedules. Alerts can be also created for unusual conditions or technical faults. Furthermore, long-term historical data storage is a part of this layer.

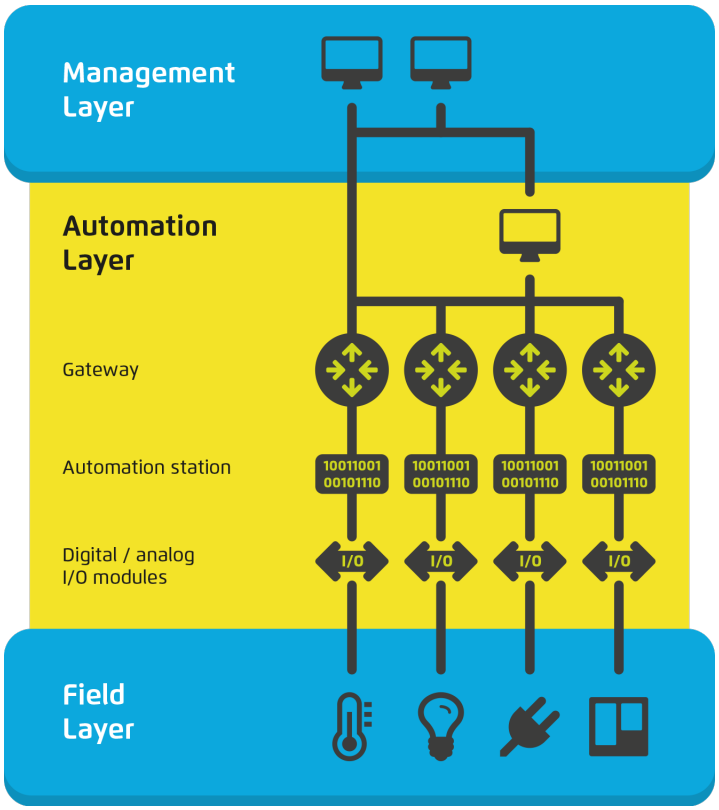

Figure 2.1: BAS architecture [5]

2.2 Internet of Things

The Internet of Things (IoT) consists of devices that communicate with each other over the Internet [1]. These devices collect and exchange data following a machine-to-machine communication. IoT devices are used in industry, transportation, logistics, healthcare,

smart environments, etc. By 2020, more than 25 billion devices are expected to be connected to the IoT.

2.2.1 Architecture

The architecture of the IoT can be generally described into six layers (Figure 2.2).

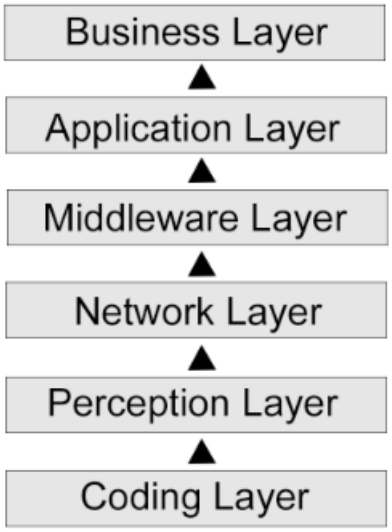

Figure 2.2: IoT architecture adapted from [1]

Coding Layer:
The Coding Layer is the foundation of IoT that offers identification. Each device is associated with a unique ID.

Perception Layer:
The Perception Layer is the device layer of the IoT and it gives physical meaning to objects. It can consist of sensors in different forms like RFID tags or IR sensors for temperature, speed or location of the objects. The Perception Layer bundles the sensor information of the linked devices and converts the information into digital signals which are then sent to the Network Layer.

Network Layer:
After receiving the digital signal information from the Perception Layer, the Network Layer transmits this information to the processing system in the middleware. This transmission can be done through transmission media like Wi-Fi, Bluetooth, GSM, 3G, or with protocols like IPv4, IPv6, MQTT, DDS, AMQP.

Middleware Layer:
This layer processes the information from the sensor devices. It includes technologies like Cloud computing or Ubiquitous computing, that enables access to databases to store all necessary information in it.

Application Layer:
The Application Layer represents the application of IoT for all kinds of industry, based on the processed data. IoT related applications could be smart homes, smart transportation, smart city, etc.

Business Layer:
The Business Layer manages the applications and services of IoT devices and generates business models for business strategies.

2.3 Integration of BAS into the Internet of Things

One way to integrate BAS into the IoT realm is by means of a centralized server (gateway) [6]. The server hides the complexity of the BAS from the IoT and provides a translation between two worlds. Another way would be to connect all field devices in the Field Layer to the IoT. The advantage is that no gateway is needed, but it is very costly to connect each device alone to the IoT.

Besides the exchange of runtime information, the integration of BAS into the IoT is useful for the device maintenance [7]. For example, if a device reports a faulty state, the relevant supplier can be automatically contacted to arrange a service. With the help of BAS in combination with the IoT, smart grids can be realized. For example, with the help of external services, BAS could know when much renewable energy is viable or the electronic grid is not overloaded. With this information, BAS can plan to shift energy consumption of HVAC devices to times when it is most efficient.

CHAPTER 3

IoT Communication Protocols

IoT protocols enable the communication between IoT devices. For this, many different communication protocols have been developed. In this section, MQTT and AMQP, two of the most frequently used protocols, will be discussed. Advanced Message Queueing Protocol (AMQP) is an open standard and currently available in two different versions (1.0 and 0.9.1) [8]. Both protocols are completely different and not compatible. AMQP 0.9.1 and AMQP 1.0 have to be seen as completely different protocols and because of this they will be discussed separately.

3.1 MQTT

Message Queuing Telemetry Transport (MQTT) is a lightweight transport protocol, which efficiently uses the network bandwidth with a 2-byte fixed header [9]. Otherwise MQTT works on TCP and can assure the delivery of messages [10]. MQTT was released by IBM in 1999. It was developed to send data taking into account long network delays and low bandwidth and has been recognized as standard by the Organization for the Advancement of Structured Information Standards (OASIS).
MQTT is based on a publish/subscribe paradigm. Any connection in MQTT involves two different kinds of agents (MQTT clients and an MQTT public broker) (Figure 3.1). Messages that are sent with MQTT are typically application messages. Hardware or software that is connected with MQTT and is exchanging application messages, is called an MQTT client. An MQTT client can be a publisher or a subscriber. An MQTT client publisher publishes application messages and MQTT subscribers are requesting application messages. An MQTT broker can be a device or program, that interconnects MQTT clients, accepts and forwards application messages. Usually, devices like sensors are MQTT clients. When an MQTT client has information to broadcast, it publishes the data to an MQTT broker. The MQTT broker receives the message and delivers it to other clients that subscribed to it. An MQTT broker is responsible for the organization

and collection of data. In MQTT, the complexity is concentrated only in the broker. Publishers and subscribers are isolated from each other and do not have to know of the existence or the underlying application of each other. A topic in MQTT offers the routing

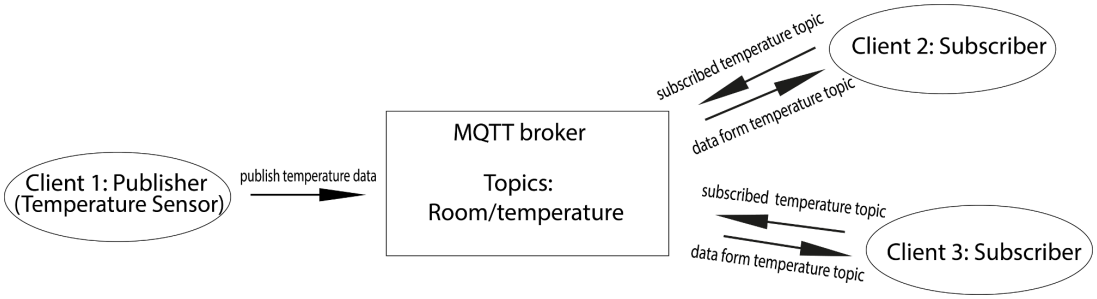

Figure 3.1: Simple MQTT transmission

information. Each topic consists of a topic name and topic levels associated with it. It is possible, that a topic tree consists of multiple topic levels and these are separated by "/". Wildcard characters, like + and #, are used to match multiple topic levels [9].

Before the transmission of the application messages, control packets have to be exchanged based on the Quality of Service (QoS) associated with them. An MQTT control packets contains a fixed header, a variable header, and the payload. Some of the control packets for the exchange between clients and broker are CONNECT, CONNACK, PUBLISH, PUBACK, PUBREC, PUBREL, SUBSCRIBE and SUBACK (Figure 3.2).

Connection setup:
To enable a connection between an MQTT client and an MQTT broker, control packets have to be exchanged. The MQTT client, that wants to connect to an MQTT broker needs to send a CONNECT packet to the broker specifying its identifier, flags and protocol level. A broker accepts the connection with a CONNACK packet and with a return code to specify the status of the connection.

Publishing:
A client needs to send a PUBLISH packet to the broker, if it wants to be a publisher. The PUBLISH packet contains details about the QoS level of the transmission, a topic name and the payload. Generally, MQTT supports 3 levels of QoS. With QoS0 the MQTT client does not receive any acknowledgment for the published packet from the broker. With the use of QoS1, the client receives back a PUBACK packet including the packet identifier. 4 Packets are exchanged with the use of QoS2. The broker acknowledges the PUBLISH packet with the PUBREC packet. Next, the MQTT client sends the PUBREL packet. Then, the broker sends the PUBCOMP packet back to the client and this indicates the publishing of a message on the given topic.

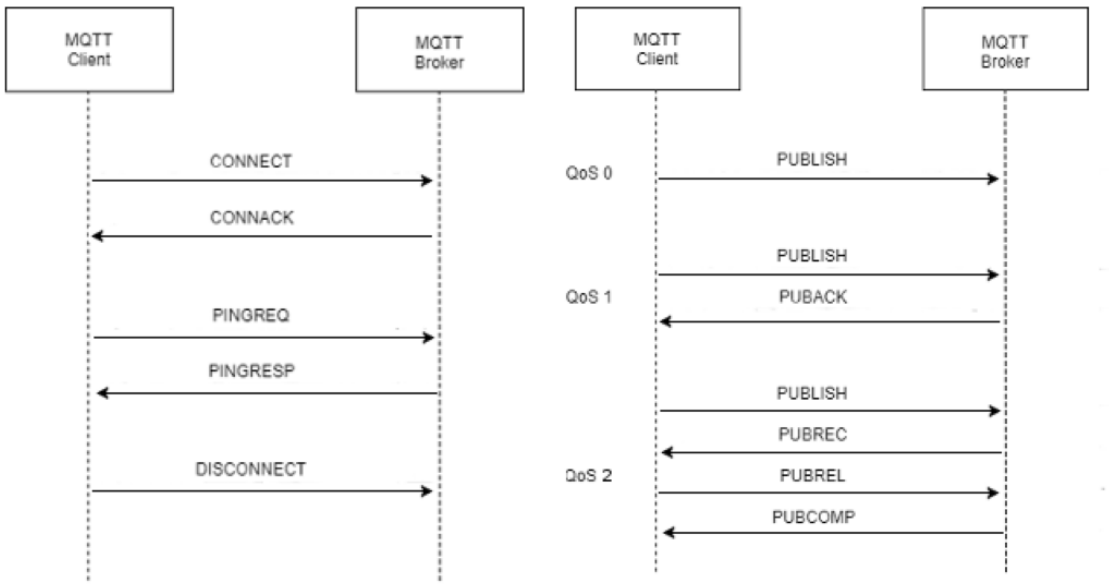

Figure 3.2: MQTT sequence diagram

Subscription:
To subscribe a topic, an MQTT client has to send a SUBSCRIBE packet to the MQTT broker including the topic name in UFT-8 encoding. The broker accepts the subscription with a SUBACK packet including a return code denoting the status of the request. If the subscription has been successfully setup, the broker messages the client constantly with the maximum QoS to maintain the connection. To cancel a subscription, a client sends an UNSUBSCRIBE packet to the broker. The Broker then sends an UNSUBACK packet back to the client in order to confirm a cancelation of the subscription.

Maintaining a connection:
A connection has to be maintained, because a connection between a client and a broker is terminated after a while. To maintain a connection, the MQTT client has to send a PINGREQ packet to the server. An MQTT broker sends then a PINGRESP packet back to the client and maintains the connection alive.

Connection close:
To terminate a connection, the MQTT client sends a DISCONNECT packet to the server. After this, the MQTT broker does not acknowledge this packet and the client gets disconnected from the broker.

3.2 AMQP 0.9.1

AMQP was developed for financial transactions such as for trading and banking systems, which require high reliability, scalability, and manageability. The protocol is designed for interoperability and this allows that different platforms, implemented in different languages can exchange messages.
AMQP 0.9.1 follows, like MQTT, the publish/subscribe paradigm [11]. The main part in AMQP 0.9.1 is the broker which consists of exchanges and message queues. Unlike in MQTT, publishers do not directly send the messages to the queues. The messages are sent by the publishers to "exchanges" that route an incoming message to a dedicated queue whose binding key matches with the message's routing key. If more subscribers are interested in a message, the message can be duplicated and sent to more queues. A message stays in a queue until it is received by a subscriber (Figure 3.3).

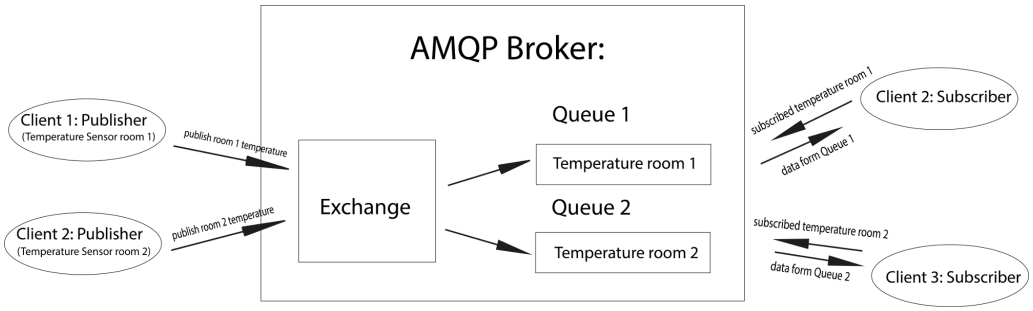

Figure 3.3: Simple AMQP transmission

3.2.1 Queues

A message queue stores messages and distributes them to one or more clients [12]. Here it is important to mention, that each queue is independent. A queue has various properties like private or shared, durable or temporary, broker-named or client-named. Because AMQP queues are not formally defined it is possible to create custom queues by combining different properties. Usually subscription queues are temporary, broker-named and private to one client. For example, a private subscription queue holds messages from various publishers and sends the messages to only one subscribed client.

3.2.2 Exchanges

Exchanges receive messages from publishers and route them to message queues following pre-arranged criteria, which are called "bindings". In AMQP, exchanges do not store messages. They inspect the received messages with the help of binding tables. In addition, messages are assigned to specific queues. The protocol defines some standard exchange types, which are sufficient for common message delivery. Furthermore, AMQP allows developers to create their own exchange instances. Additional to exchanging messages,

AMQP can act as an intelligent agent that accepts messages and even produces messages if needed.

Usually an exchange examines the message properties, the header field, the body content and additional data from other sources to decide, where a message has to be routed. Often the routing is defined by a key field which is called the “routing key”. The routing key is a virtual address, that an exchange uses to get the destination of the route. In most of the implementations, the routing key for point-to-point routing is the name of a message queue. For topic routing, the routing key is the topic name, like in MQTT. To enable more complexity, the routing key can be combined with the header fields and/or its message content.

3.2.3 Messages

A new message is created with the AMQP client API by a publisher. After this, the publisher sets the content and additional message properties. Furthermore, the publisher has to set the routing information before the message is sent to an exchange on the broker. If a message is unrouteable, the exchange either deletes the message or returns it to the publisher. This depends on the chosen properties of the publisher. A single message can exist on different queues. This can be done by copying the message or with the use of references. A copied message on a message queue is identical to the original. When a message arrives at a message queue, the broker immediately tries to pass the message to a subscriber. If the subscriber is not ready to receive a message, the queue stores the messages. Without an existing subscriber, the broker is able to return the message back to the publisher.

The principle of AMQP 0.9.1 is similar to an email system. All messages sent from a producer are sent to a single point, the exchanges. The routing to the queues is hidden from the producer. Subscribers on the other hand are able to create or destroy message queues and define how message queues are filled. Furthermore, a subscriber can select different exchanges which can lead to changing routing semantics.

3.2.4 QoS

AMQP uses TCP for reliable transport and this provides 3 levels of QoS. QoS0 works without confirmation on message reception. On the other hand, QoS1 ensures that messages will arrive at subscribers. So the subscriber needs to send an acknowledgment to the broker. If it does not arrive in a defined time period, the message will be re-sent. QoS2 ensures that a message will be delivered exactly once without a duplication. AMQP provides security mechanisms for data protection with the use of TLS for the encryption of messages. For authentication, SASL (Simple Authentication and Security Layer) is common.

3.3 AMQP 1.0

AMQP 1.0 is defined as an efficient binary peer-to-peer protocol for transporting messages between two clients [13]. It was an important big step to a standardized AMQP version, that is universally applicable. The older version, AMQP 0.9.1 is still more popular and used by many message brokers.

3.3.1 Structure

The AMQP network consist of nodes that are connected with the help of Links. The network nodes are also called entities which are responsible for the storage or delivery of messages. A Link can be seen as a unidirectional route between two nodes. Links are attached to a node at a Terminus and a Terminus can be either a "Source" or a "Target" (Figure 3.4). The two Termini of a Link are responsible for tracking the state of a message stream. A Source Terminus tracks outgoing messages, while a Targets tracks incoming messages. All entry criteria have to be added at the Source before a message can be sent. During a message transfer, the responsibility for storage and delivery of messages is passed through the entered nodes.

Figure 3.4: AMQP 1.0 peer-to-peer link

In AMQP 1.0, containers are used. Each node exists within a container and each container can contain many nodes. Nodes can be for example Producers, Consumers or Queues. Queues are the part of a Broker that are responsible for the storage and distribution of messages. On the other hand, Producers and Consumers are part of a client application that generates and processes messages. Containers can be brokers or client applications (Figure 3.5).

3.3.2 Transport

AMQP 1.0 is, as already mentioned, defined as a peer-to-peer transportation protocol. But this concerns only the node to node communication and not the internal work of AMQP 1.0. Connections enable the communication between containers. AMQP offers a

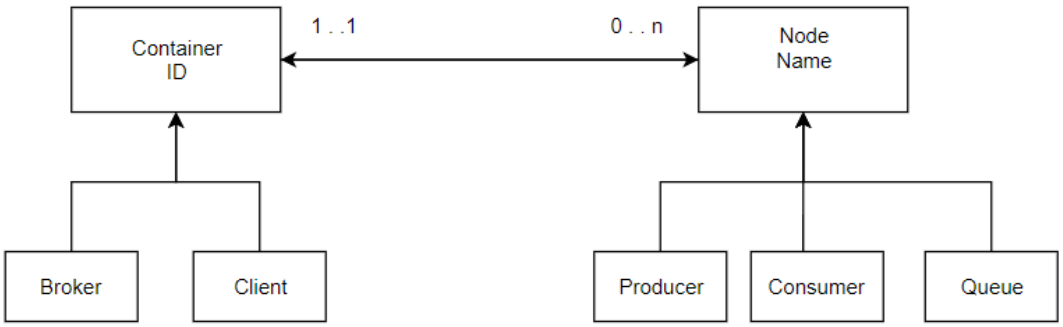

Figure 3.5: AMQP 1.0 structure

full-duplex, reliably ordered sequence of frames. An active connection requires all prior frames to be transferred, before the next frame can be transferred. If an unknown number of frames get lost, the connection will fail. A connection gets divided into a negotiated number of independent unidirectional channels. Frames contain the channel number indicating their parent channel. The frame sequence for each channel is multiplexed into a single frame sequence for the whole connection.

Sessions provide the context for the communication and with the help of a session, the link protocol is used to establish a link between Source and Targets.

3.4 MQTT and AMQP 0.9.1 Comparison

3.4.1 Similarities

MQTT and AMQP 0.9.1 have some similarities [7]. First of all, both are based on the publish-subscribe paradigm and are applying a message queuing scheme. Furthermore, both protocols are working in an asynchronous manner and are supporting cloud computing. They both need only a minimal set of configurations and are based on TCP/IP.

3.4.2 Dissimilarities

The MQTT protocol fits for small devices that are connected on low bandwidth networks. On the other hand, AMQP can be used for all kind of devices and for any bandwidth. For the frame optimization, MQTT does not use fragmentation, but AMQP supports it. MQTT also uses a stream-oriented approach and encoding and decoding of frames

is easy for low memory devices. Instead of a stream-oriented approach, AMQP uses a buffered oriented approach. For the response, MQTT supports basic acknowledgments. AMQP supports different acknowledgments for different use cases.

3.4.3 MQTT Advantages

MQTT is a lightweight protocol, that is suitable for low bandwidth and devices with low computing power [15]. Because of this, MQTT is useful for devices that need to guarantee a long battery life. Another advantage of MQTT is that it is very easy to use, due its simplicity. Furthermore, MQTT message header consists of only 2 Byte. MQTT is used by big companies like Facebook, IBM, Casio and Amazon [14].

3.4.4 MQTT Disadvantages

MQTT is the most popular communication protocol for the IoT, but it has some disadvantages. By default, the MQTT protocol provides username and a password for an authentication and many brokers facilitate their own security mechanisms. Often brokers offer additional security based on TLS, but TLS affects the performance of the broker. MQTT does also not support priority of messages. For example, a fire alarm is more important than temperature data and so it should be delivered also with higher priority. Important in the IoT is the resending of messages, which got lost during a transmission. MQTT offers the guarantee of delivery, but ensures not maintaining the right message order. MQTT supports only messages with a payload up to the maximum size of 256 MB.

3.4.5 AMQP Advantages

The message size of AMQP is unlimited and negotiable. Thus, it is suitable for messages with a high payload. Unlike MQTT, AMQP only offers three QoS levels, nevertheless it is also a very reliable protocol. The protocol supports higher security mechanisms than MQTT. AMQP is preferred for businesses, because it offers services like reliable queuing, topic-based and subscribe messaging, flexible routing and transactions. AMQP has been used in the world's biggest projects like OceanographyS monitoring of the Mid-Atlantic Ridge, NASAS Nebula Could Computing, and Indias Aadhar Project.

3.4.6 AMQP Disadvantages

AMQP requires higher computing power and resources than MQTT, in oder to offer provisioning and reliability. Furthermore, AMQP requires higher bandwidth and and causes higher latency.

3.4.7 Comparison Conclusion

AMQP and MQTT are both common IoT protocols and have their different advantages and disadvantages. Because of its lightweight nature, MQTT should be used for large

networks of small devices that need to be controlled by a back-end server on the Internet and not for a device-to-device transfer of multicast data. On the other hand, AMQP should be used for devices with higher computing power, where security is more important.

CHAPTER 4

KNX

In conventional building installations, each system like lighting, heating and conditioning is implemented by different companies. This is why sensors and actuators are usually connected via point-to-point connections. This increases the planning effort, the wiring and respective installations and declines maintenance. Furthermore, advanced features require more wires to be laid while the use of more wires causes higher fire risks.

KNX is a building control system that connects devices such as sensors, actuators, operating terminals, and controllers by a common network [16]. The KNX technology is included in the electrical installations of buildings and is decentrally organized without any need of a central control station. Worldwide, KNX claims to be the only open standard for buildings which means that devices from different manufacturers can communicate with each other.

For each building domain, there are devices with KNX support. This reduces the planning and implementation effort and offers superior functionality, flexibility and comfort. By setting of parameters, the functionality can be changed anytime enabling higher flexibility.

A simple example for a communication in a KNX system is a lighting control. In this case, a signal is created by a push button when, for example a switch is pressed. After this, the switch sends the signal via the network as a data frame to an actuator. The actuator receives the data frame and takes an action, for example turning the light on.

A disadvantage of KNX is that KNX supported devices are far more expensive than devices that are used in conventional installations. If more installations need to be implemented in a building, it decreases the costs, because the already available KNX system just has to be extended.

4.1 Transmission

KNX supports different types of transmission media to transfer data from one device to another.

KNX.TP:
The most commonly used transmission medium is the Twisted pair (KNX.TP) cable. It is used for new buildings or in existing buildings if an additional cable can be installed without problems. KNX.TP is the cheapest way to use KNX. The classic KNX line contains four wires twisted with each other. The red and black pair of wire is used for power and the transmission of data. The yellow and white pair is used for an addition bus line or additional energy if required. The data transmission is carried out byte by byte and in an asynchronous manner. The data transfer speed is about 9,6 kBit/s.

KNX.PL:
Powerline communiction is used in buildings, where it is not possible to install a second wire, examples for this could be landmark buildings. In KNX.PL, the data signals are added to sinusoidal voltage of the energy supply network.

KNX.RF:
With the use of radio frequency transmission, no additional bus lines are required to be installed, because the transmission is wireless [17]. To be independent from the energy supply network, KNX.RF devices are often provided with batteries. Solar energy panels are also common alternatives. The choice of the right radio frequency influences the transmission quality. By default, KNX.RF uses 868,3 MHz for the transmission and has a maximal transmission power of 25mW. This is enough for a single family house, but for huge transmission networks, retransmitters have to be installed. The transmission speed in KNX.RF systems is with 15 kBit/s faster than in systems that use KNX.TP. Due to the utilization of wireless connections, KNX.RF needs to send more information.

KNXnet/IP:
KNXnet/IP is used to connect devices with IP networks of automated buildings. Optical fiber is used to connect buildings over longer distances. A positive side effect of using optical fiber is, that there is no need for lightning protection or overvoltage protection.

The KNX communication over the Ethernet is defined as KNXnet/IP. The Ethernet is an open and powerful network following the IEEE 802.3 standard [18]. Nowadays, the Ethernet is the worldwide number one network.

4.2 KNX Gateway

A KNX Gateway is defined as a standardized interface between KNX networks and other information technology systems [19]. It is capsuled in a gateway device which is able

to communicate with the KNX network and a connected IT system (Figure 4.1). The Gateway has to support encoding standards and message exchange protocols to enable a remote access to the KNX network via the Internet. The KNX Gateway device is just an abstract concept and is not restricted to any hardware requirements.

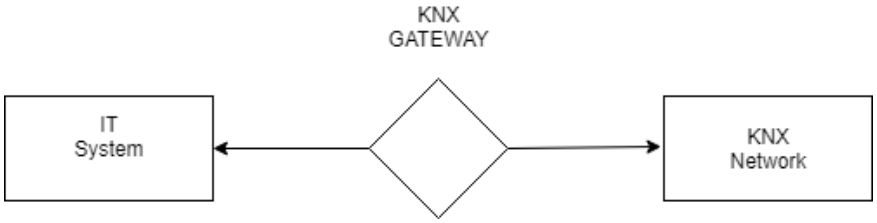

Figure 4.1: KNX Gateway

A KNX Gateway basically needs three interfaces to be implemented. First of all, the KNX information model interface, which defines and offers the KNX Gateway a representational structure of the KNX Network. The second interface is the KNX Web interface that connects the KNX Gateway and the remote system which describes the available access points and the structure of the data that are provided and expected by services of the KNX Gateway. Finally, the KNX Network access interface is used to connect the KNX Gateway with the KNX network to enable message exchange and routing requests from the connected IT system.

4.3 KNX Information Modeling

To offer access from an IoT system to a BAS, information of the used BAS needs to be made available. There are many different BAS technologies and many integration solutions, because of that, an abstract modeling concept is required. To enable machine-to-machine communication, this modeling concept needs to cover relevant semantics. This can be done by a model-driven approach supporting semantic modeling by a list of tags that are summarized in a vocabulary according to the publication “Modeling framework for IoT integration of building automation systems” [2].

4.3.1 Meta-model

In Model-driven engineering (MDE) a system, in our case a BAS, is usually an instance of a meta-model describing a DSL. In this case, the concept has to be extended by a common meta-meta-model in order to declare a notation of the meta-models. A meta-model is required, which specifies the tags of the DSL and their composition to class-like structures. This principle can enable powerful vocabularies and comprehensive system models.

In this concept, a Tag includes the description of entities (Entity) in form of features (Feature) (Figure 5.2). A Tag consists of a unique name and an optional description. There are three type of Tags, a Basic tag has a simple data type corresponding to a

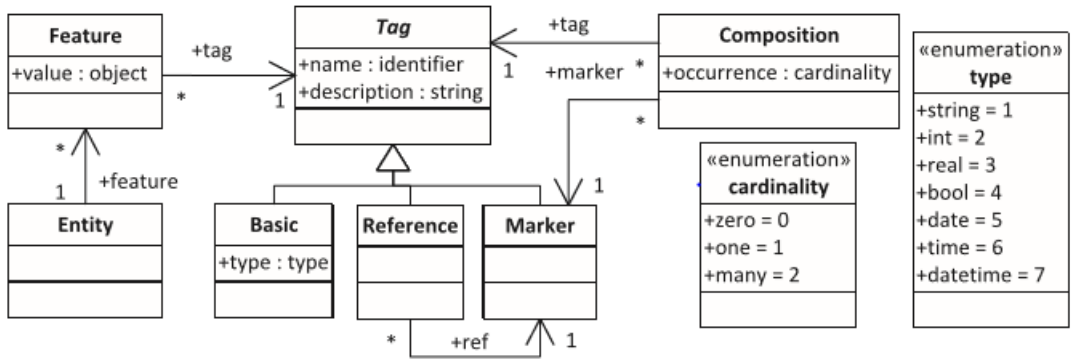

Figure 4.2: Meta-model [2]

type enumeration like string, int, etc. On the other hand, a Marker Tag is used to express an "is-a" relationship and does not have any value. Finally, Reference Tags are used to describe relationships between Entities. With the help of the Composition class, compositions can be created. The Composition class contains the enumeration cardinality and this defines the number of allowed appearances of the associated tag within a single entity. This meta-model provides the basics of the modeling concept, that is encoded in XML.

4.3.2 Vocabulary

A core set of important tags is already predefined as vocabulary (Figure 4.3). Tags are differentiated between physical elements and their arrangements in the context of building, functionality, or topology. Furthermore, the semantic modeling concept requires units and enumerations to define datapoints or functional blocks.

Generally, basic tags are illustrated located in the middle of the Figure 5 like id, name, description and locale. In block 1, tags are located that are describing devices. For example, each device has a manufacturer and a serial number. The block 2 contains Tags that are based on general Marker Tags for functionality, topology and building part. On the other hand, the block 3 contains Tags for units and enumerations. The Tags in the block 4 are used for modeling a datapoint type that consists of a set of values.

This modeling concept represents only static information and no runtime information in the form of process data is modelled. Finally, the modeled BAS has to be mapped to a KNX gateway to enable the communication.

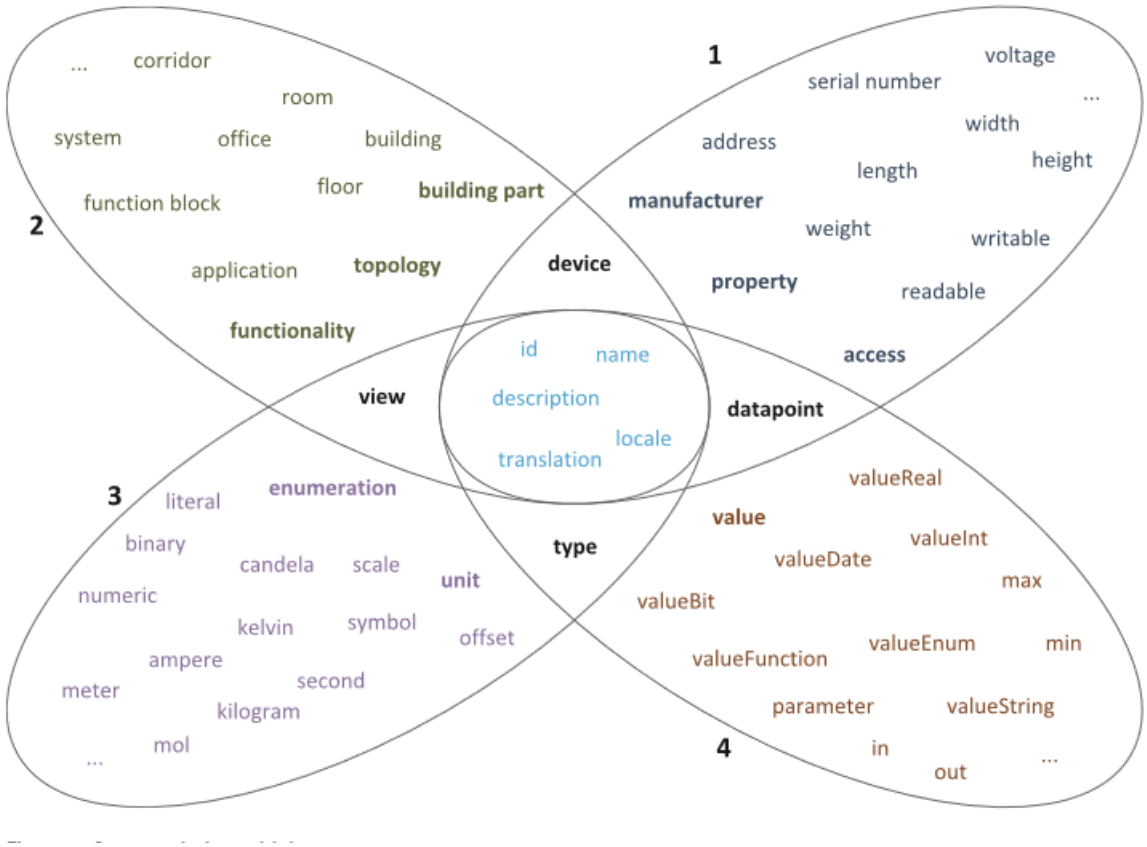

Figure 4.3: Predefined Vocabulary [2]

CHAPTER 5

Practical Part

The practical part of this thesis is an implementation of an application that enables a communication between a KNX system and the IoT for BAS (Figure 5.1). The application supports MQTT 3.1 and AMQP 0.9.1. The test case consists of IoT devices monitoring lab equipment which are connected via KNX. The software uses a KNXnet/IP Gateway to access the connected KNX system.

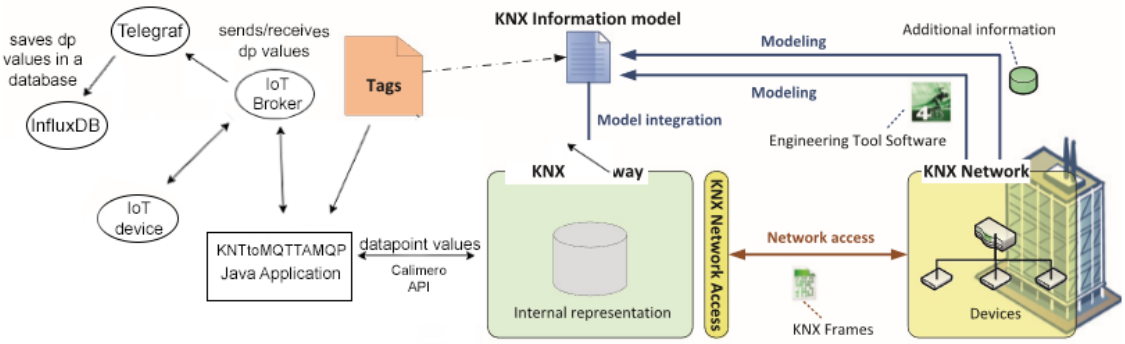

Figure 5.1: Project sketch

5.1 Applications Architecture

5.1.1 Use case diagram

The KNXtoMQTTAMQP application has two main use cases (Figure 5.3). The KNX System transmits information of its datapoints to an IoT broker. Furthermore, IoT devices can change datapoint values over an IoT protocol of the BAS.

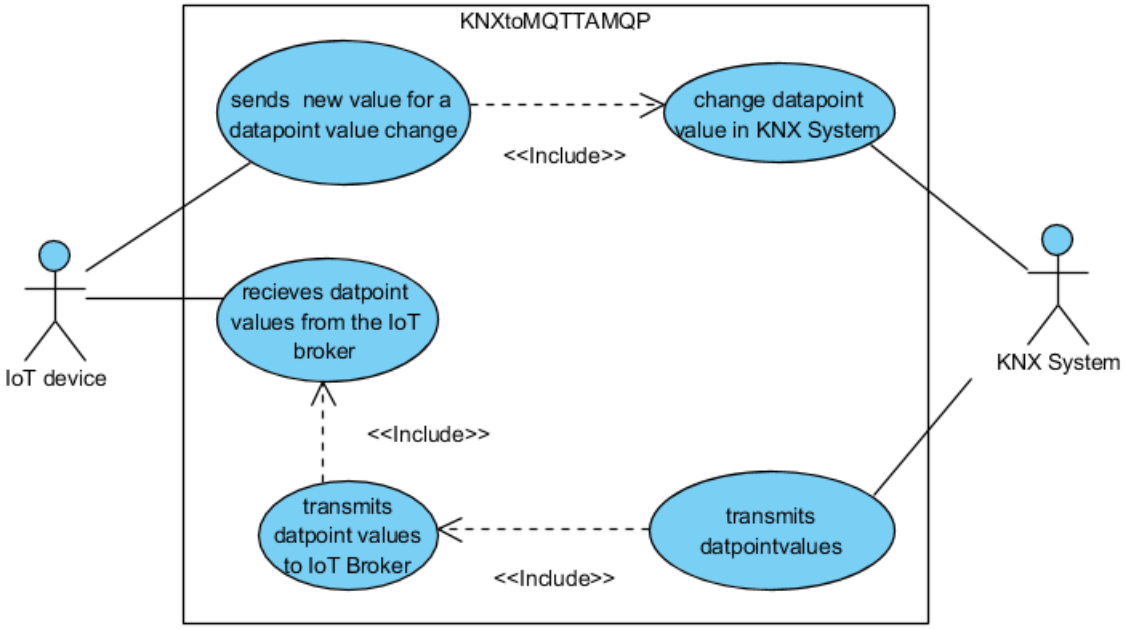

Figure 5.2: KNXtoMQTTAMQP Use case diagram

5.1.2 Class diagram

After the user runs the “Main” class, the class creates a new “XML_Reader” object, that imports all KNX datapoint entities of the “knx_input_model.xml” file and a “PropertiesManager” object, that manages all settings of the “KNXtoMQTTAMQP” configuration file (Figure 5.2). The Main class creates also a “DatapointManager” object. The Constructor of the “DatapointManager” class creates for each valid datapoint entity a datapoint object. Furthermore, the “DatapointManager” creates depending on the chosen IoT protocol an IoT Communication and IoT Listener object.

5.1.3 Sequence diagram

After the user starts the application, the application reads all datapoint entities from the “knx_input_model.xml” file and starts a connection to a KNX network and an IoT broker (Figure 5.4). In a loop the application reads all KNX datapoint values in a frequency of one second and broadcasts them to an IoT broker.

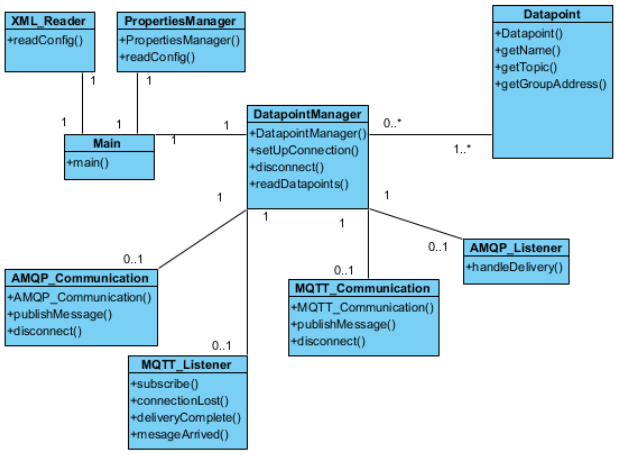

Figure 5.3: KNXtoMQTTAMQP class diagram

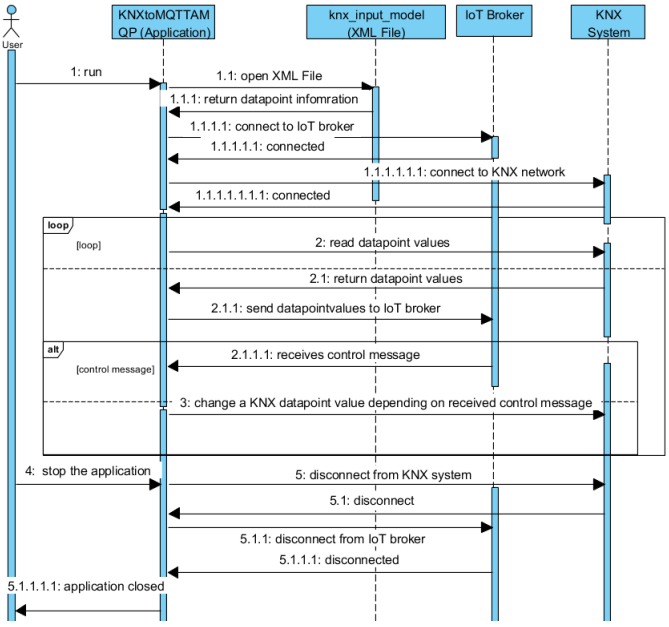

Figure 5.4: KNXtoMQTTAMQP Sequence diagram

When the Application receives a control message from the IoT broker, it changes the KNX datapoint value according to the received control message. If the user enters “q” into the KNXtoMQTTAMQP console, the application closes all opened connections to the IoT broker and the KNX network. After this the applications terminates.

5.2 Components

5.2.1 Application

The IoT broker is implemented as Java application (Figure 5.5). The project is set up with Maven and written with JDK 9.0. The communication between the application and the KNX network is provided by the Java library Calimero v2.4. The Java library Paho 1.2 is used for the communication between the application and an AMQP broker. Furthermore, the RabbitMQ client library 5.4.1 enables the communication to an AMQP Broker. For the test scenario a Mosquitto server is used as an MQTT Broker and RabbitMQ as an MQTT 0.9.1 broker.

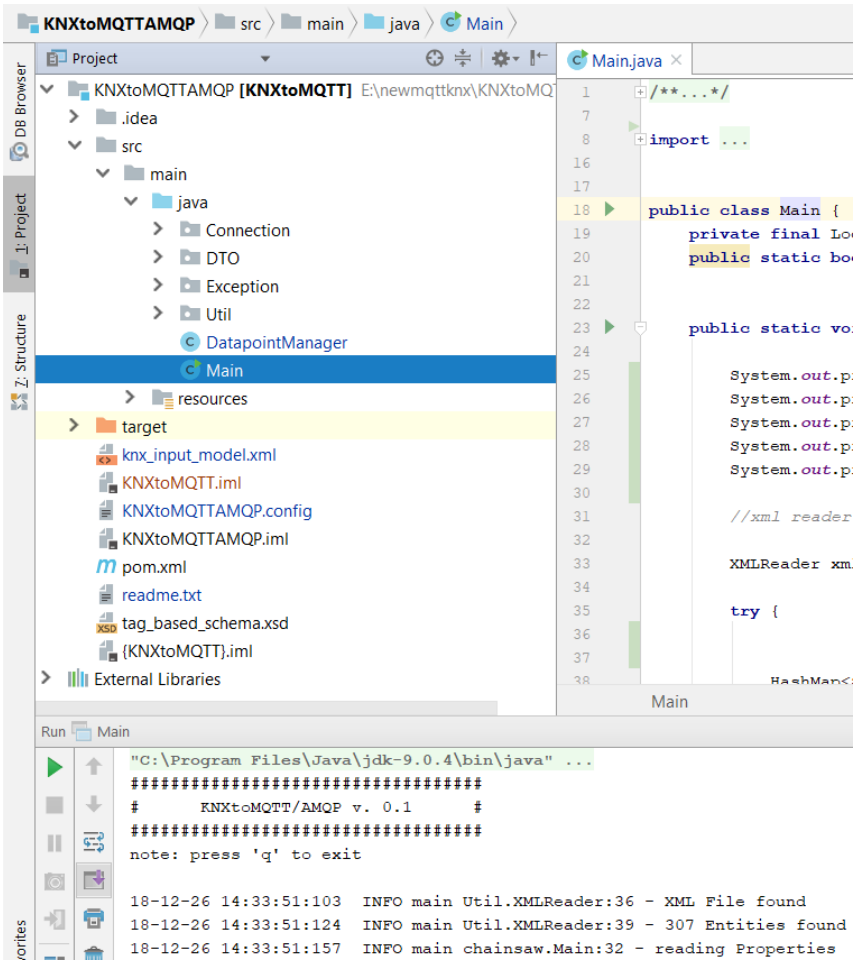

Figure 5.5: KNXtoMQTTAMQP Java Application

Calimero is a Java API that allows access to KNX systems developed by the Automation Sytems Group of TU Wien. The API acts like a bridge between a Java application and KNX over an IP network interface [20].

5.2.2 Gamma Training Kit

For the test case the Gamma Training Kit by Siemens acts as a BAS (Figure 5.6) [21]. The Gamma Training Kit simulates a building containing 6 dimmable LED lamps, 2 blinds simulations, 2 window simulations and a cooling/heating simulation connected via KNX. The IP-Router N 146 enables the connection between the the Java application and the Training Kit.

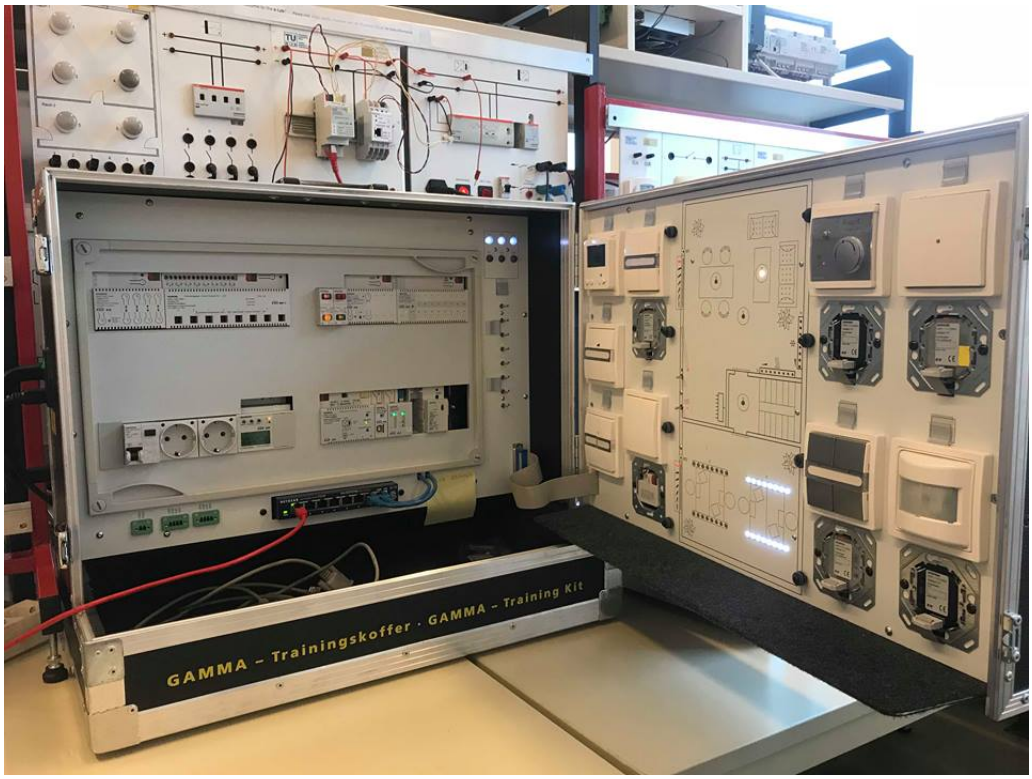

Figure 5.6: Gamma Training Kit

5.2.3 ETS5

ETS 5 (Engineering Tool Software) is a Windows software by KNX that is a configuration software tool to design and configure building control installations with the KNX system [22]. With this software the Gamma Training Kit has been configured to simulate a building and is also used to export the KNX WS Information model of a Gamma Training Kit with the help Web Service Exporter ETS 5 APP (Figure 5.7) [23].

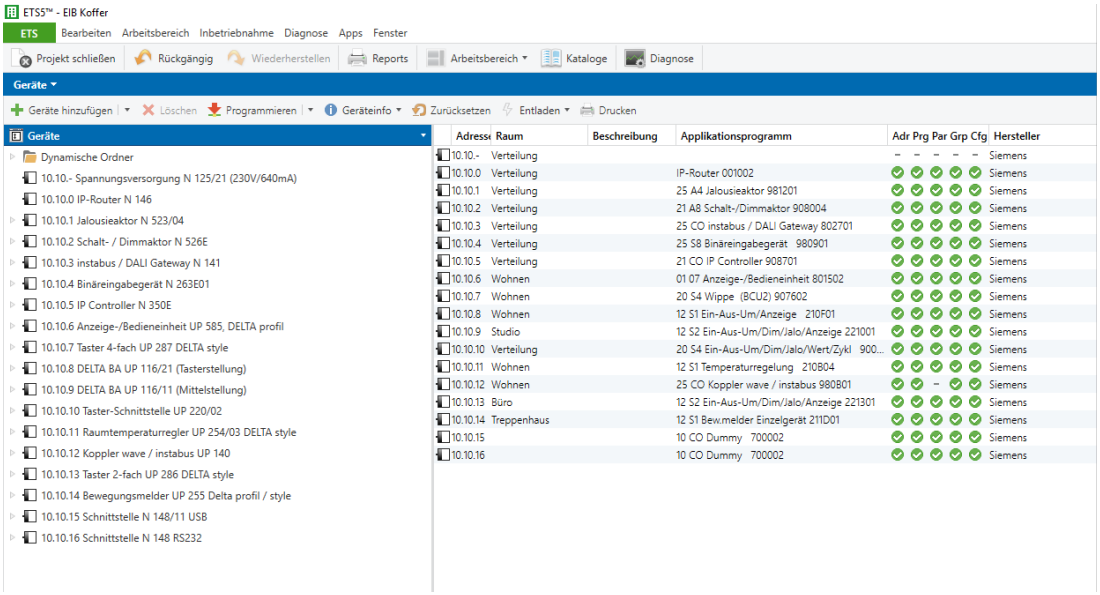

Figure 5.7: ET5 Project of the Gamma Training Kit

5.2.4 InfluxDB

InfluxDB is a time series database to handle high loads of data. It is used to store large amounts of timestamped data like IoT sensor data, application metrics, and real-time analytics [24]. An InfluxDB database is similar to an SQL table, the time is always the primary index. Unlike in SQL tables, no table-schema has to be defined and a database can contain millions of different measurements [25].
InfluxDB 1.6 stores the sensor data received from the IoT broker in a database by using Telegraf 1.7.3. Telegraf is a plugin-driven server agent for collecting and reporting metrics [26]. With the help of the Telegraf MQTT Input Plugin, it is possible to add messages from specified MQTT topics to an InfluxDB database. On the other hand, the Telegraf Input Plugin for AMQP 0.9.1 saves messages from specified AMQP message queues.

5.2.5 Mosquitto

Mosquitto is an open source message broker by the Eclipse Foundation and implements the MQTT protocol version 3.1. Because of its lightweight fashion, the broker is suitable for low power devices [27].

5.2.6 RabbitMQ

RabbitMQ is an AMQP 0.9.1 broker. It was originally developed for AMQP, but RabbitMQ also supports other IoT protocols like STOMP, AMQP 1.0 or HTTP. RabbitMQ offers its own client library for the communication between the AMQP broker and a Java application [28].

5.2.7 KNX Information Model

The KNX Information Model of the KNX WS specification is more simplified than the concept mentioned in chapter 4. For example, there exist no sub classes for special tags and the class feature is called tag/value pair. Only “one” and “many” are supported as cardinality. Furthermore, the class composition is called relation. At the moment, the application supports as input only the KNX WS Information model of a BAS, that has been exported with Web Service Exporter [23]. The application logic creates for each readable datapoint-entity of the exported XML-file a new topic or queue.

```
<?xml version="1.0" encoding="utf-8"?>
<project version="0.4">
    <entity id="115" name="thermometer" orderNumber="5WG1 254-2AB_3" serialNumber="" datapointRef="115_0" device="true" />
    <entity id="115_1" name="room temperature" description="temperature in degrees" datapoint="true" />
    <entity id="A-68-1" datapointRef="115_1" groupAddress="1/3/5" updatable="false" readable="true" writable="false"  />
</project>
```

Figure 5.8: KNX WS information model example

The graphic above shows a valid example of a thermometer device, that is connected with a KNX network. In this case, the application publishes the temperature value from a KNX datapoint with the address “1/3/5” to an MQTT topic or AMQP queue named “thermometer/roomtemperature” every second.

5.3 Test Cases

5.3.1 Test Case 1: reading current lab temperature

The Gamma Training Kit is equipped with a room thermostat. In this test case, the actual room temperature of the lab has to be transmitted into a InfluxDB database over the AMQP protocol. The display of the thermostat shows in this test case 20.6 degree Celsius (Figure 5.9).

Figure 5.9: Gamma Training Kit thermostat

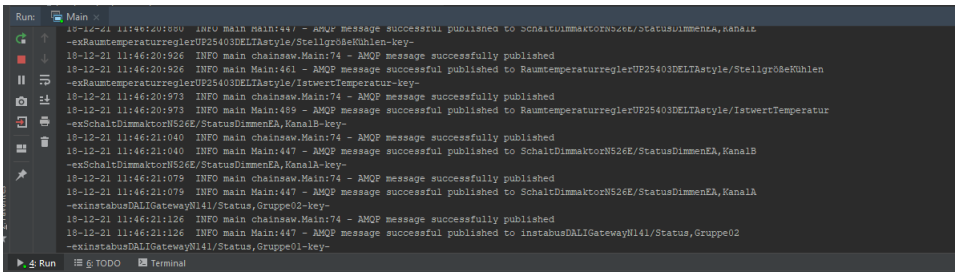

Figure 5.10: KNXtoMQTTAMQP application terminal

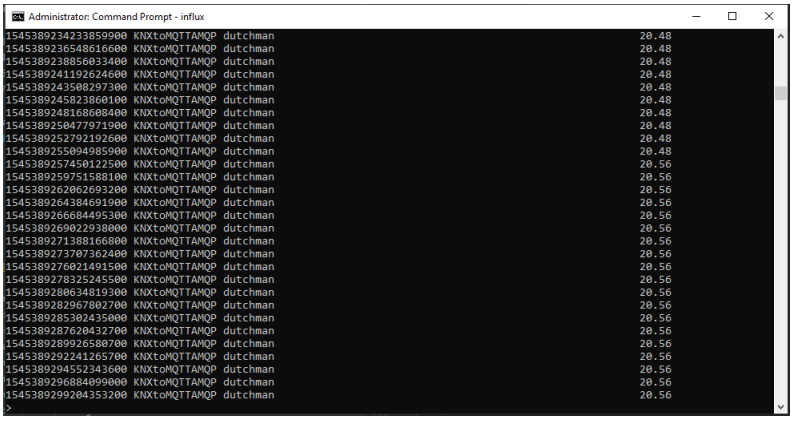

Figure 5.11: InfluxDB temperature database

The Java application reads each second all readable datapoints of the Gamma Trainings Kit and publishes them to AMQP queues (Figure 5.10). In this case, the application reads from the KNX groupAddress “1/3/4” the temperature value and publishes it on the queue “RaumtemperaturreglerUP25403DELTAstyle/IstwertTemperature” of an AMQP broker.

Telegraf is subscribing this queue and sends the temperature value into the temperature InfluxDB database. After this, the database contains the unrounded temperature value including a timestamp (Figure 5.11).

5.3.2 Test Case 2: controlling a light of the Gamma Training Kit

As already mentioned the application can be used to control the Gamma Training Kit. In this test case, the light C with the KNX group address “1/1/2” will be switched on. Before the test, the "C" light is switched off (Figure 5.12).

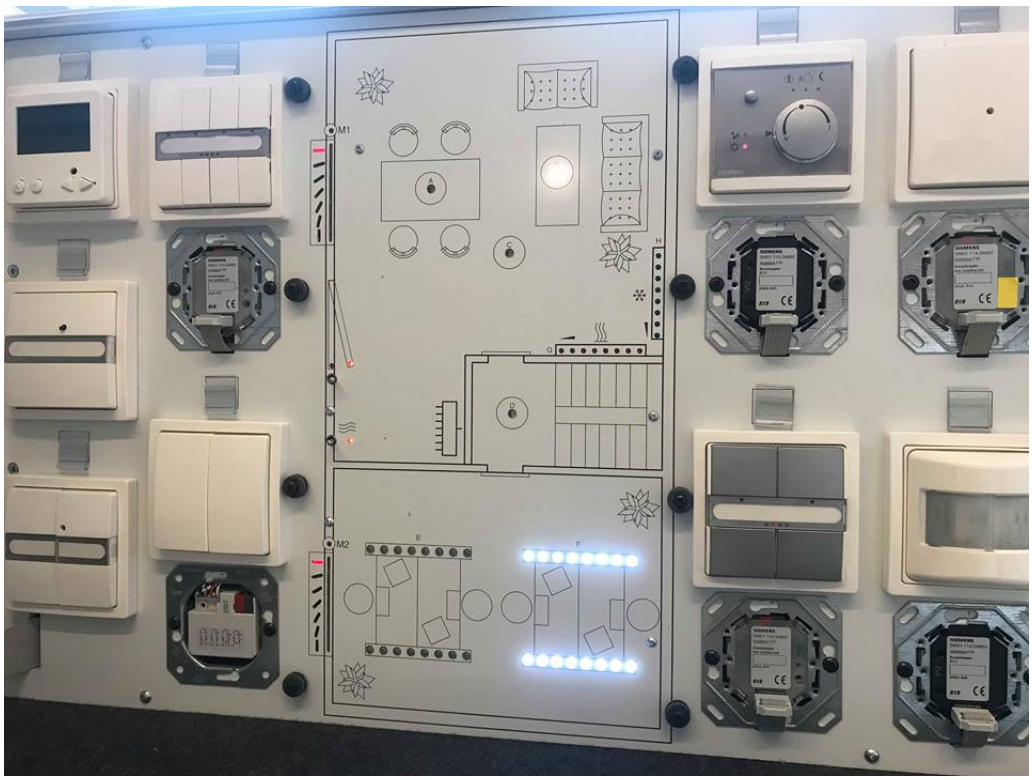

Figure 5.12: Gamma Training Kit

To switch the light on, a command message has to be published to the AMQP_input queue. In this case, the message “1/1/2;boolean;true” has to be sent. This means that the KNX datapoint value has to be set to true (Figure 5.13).

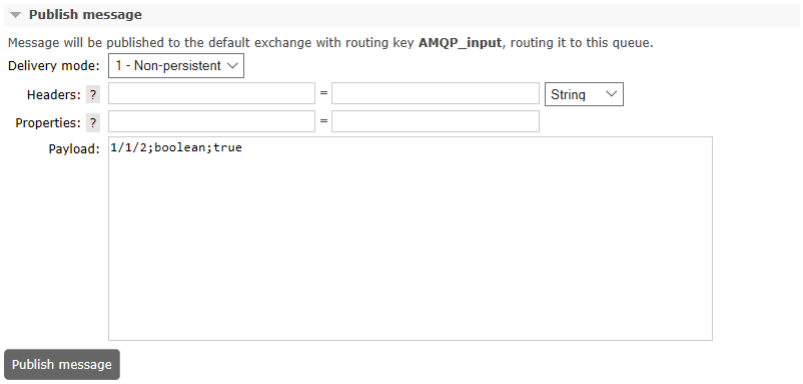

Figure 5.13: Publishing control-message to AMQP broker

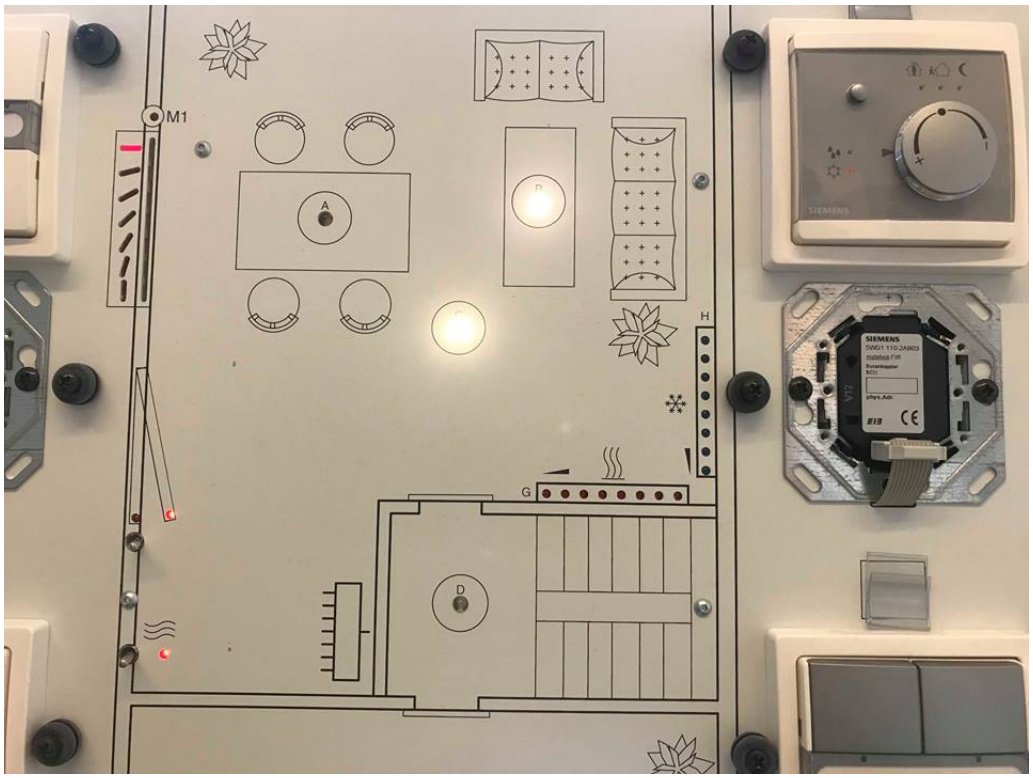

Figure 5.14: Light C switched on

Now the C light of the Gamma Training Kit is on (Figure 5.14)

5.3.3 Application User's Guide

Before the use of the "KNXtoMQTT/AMQP" application, the specified IoT broker needs to run and the KNXnet/IP Gateway has to be reachable. After this the "KNXtoMQTTAMQP.config" file in the root folder of the application has to be defined. In order to use the application properly, all parameters of the "KNXtoMQTTAMQP.config" file have to be filled in. At the moment, the application supports AMQP and MQTT for the IoT communication. Furthermore, the connected KNX system has to be exported from an ETS5 project with the Web Service Exporter and saved as knx_input_model.xml. After this, the "main" class can be executed.

By using this application, IoT devices are able to change KNX datapointvalues over AMQP or MQTT. This feature allows IoT devices to change a status easily, for instance switching a light on/off just with the change of a simple boolean value. In the AMQP mode, the IoT device has to send a message to the "AMQP_input" queue. In the MQTT mode, the IoT device has to send the message to the "MQTT_input" topic. The message consists of 3 necessary parts. It has to contain the KNX datapoint address, the data value type and the data value. These three parameters have to be separated with semicolons. The example "1/1/2;boolean;true" will change the datapoint boolean value with the groupAddress "1/1/2" to true. Currently, only double, string and boolean are supported.

The configuration file with the name "KNXtoMQTTAMQP.config" is located in the root folder of the application (Figure 5.15). The file contains parameters, that have to be set correctly before the use of the application. The first parameter is called "mode". At the moment the application supports only MQTT and AMQP, therefore "AMQP" or "MQTT" are selectable. Depending on the chosen "mode", MQTT or AMQP have to be configured. For MQTT, the URL of the chosen MQTT broker has to be defined with the "MQTT_broker_url". Important to mention is, that the URL has to contain the port. Furthermore the QoS can be defined with 0, 1 or 2. For AMQP, the username, password, virtualhost, hostname and port of the AMQP broker have to be chosen. Furthermore, the IP of the connected KNX IP Gateway has to be defined with the "KNX_host" parameter and the local IP address of the user device.


```
1  ###################################
2  #KNXtoMQTT/AMQP Configuration file#
3  ###################################
4
5  #IoT connection:
6  mode=MQTT
7  # MQTT or  AMQP
8
9  ###################################
10 # MQTT Configuration              #
11 ###################################
12
13 MQTT_broker_url = tcp://localhost:1883
14 # scheme://host:port
15 MQTT_qos = 0
16 #0/1/2
17
18 ###################################
19 # AMQP Configuration              #
20 ###################################
21
22 AMQP_userName = guest
23 AMQP_password = guest
24 AMQP_virtualHost= /
25 AMQP_hostName = localhost
26 AMQP_port = 5672
27
28 ###################################
29 # KNX Configuration               #
30 ###################################
31 KNX_host= 169.254.146.146
32 Local_Ip= 169.254.232.243
33
```

Figure 5.15: KNXtoMQTTAMQP configuration file

CHAPTER 6

Conclusion and future work

In summary, this bachelor thesis shows an overview of the integration of BAS into the IoT with the help of MQTT and AMQP. The practical part gives the implementation of a simple Java application, that acts as a bridge between a KNXnetwork and IoT brokers. The theoretical part discusses the two IoT protocols and compares them. Both protocols, MQTT and AMQP 0.9.1 are suitable for the integration of BAS into the IoT. As mentioned in chapter 3, MQTT is a very lightweight protocol and should be used for large networks of small devices. On the other hand, AMQP 0.9.1 is more secure, but requires more computing power. Therefore, AMQP should be chosen for devices with higher computing power and in cases, where security is more important.

Besides MQTT and AMQP, there are many other IoT protocols, which are also suitable for the integration of BAS into the IoT. As future work, for example, the CoAP (Constrained Application Protocol) protocol could be discussed and added to the application [25]. Furthermore the tag based model support of the application could be improved. At the moment, the applications supports as input only the KNX WS Information model of a BAS, that has been exported with Web Service Exporter [26]. The support of the tag based model mentioned in chapter 3 is left open as future work.

Bibliography

[1] Muhamed Umar Farooq, Muhammad Waseem, Sadia Mazhar, Anjum Khairi, Talha Kamel: A Review on Internet of Things (IoT), International Journal of Computer Applications 113, 2015, p. 2

[2] Daniel Schachinger, Andreas Fernbach,Wolfgang Kastner: Modelingframework for IoT integration of building automation systems, Automatisierungstechnik Methoden und Anwendungen der Steuerungs-, Regelungs- und Informationstechnik 65(9), 2017, pp. 631-635

[3] Wolfgang Kastner, Georg Neugschwandtner, Stefan Soucek, H. Michael Newman: Communication Systems for Building Automation and Control, Proceedings of the IEEE 93, 2005, pp. 1178-1179

[4] Pedro Domingues, Paulo Jorge Carreira, Wolfgang Kastner: Building Automation Systems: Concepts and Technology Review, Computer Standards & Interfaces, 2015, p. 2

[5] Kai Christiani: Get Access to Lean Building Automation, Lemonbeat, lemonbeat website, `https://www.lemonbeat.com/lean-building-automation/`, visited 26.12.2018

[6] Jasenka Dizdarevic, Francisco Carpio, Admela Jukan, Xavi Masip-Bruin: A Suvery of Communication Protocols for Internet-of-Things and Related Challenges of Fog and Cloud Computing Integration, ACM Computing Surveys 51(6), 2018, pp. 12-13

[7] Markus Jung, Christian Reinisch, Wolfgang Kastner: Integrating Building Automation Systems and IPv6 in the Internet of Things, Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, 2012, pp. 684-685

[8] Boris Adryan, Dominik Obermaier, Paul Fremantle: The Technical Foundations of IoT, Artech House, 2017, p. 336

[9] Ravi Kishore Kodali, Sreeramya Soratkal: MQTT based home automation system using ESP8266, IEEE Region 10 Humanitarian Technology Conference (R10-HTC), 2016, pp. 1-3

[10] Ashwin Makwana, Dipa Soni: A suvery on MQTT: a protocol of internet of things (IoT), international conference on telecommunication, power analysis and computing techniques, 2017, pp. 1-4

[11] Abhishek D. Pathak, Jitendra V. Tembhurne: Internet of Things: A Survey on IoT Protocols, 3rd International Conference on Internet of Things and Connected Technologies (ICIoTCT), 2018, pp. 485-486

[12] Sanjay Aiyagari, Alexis Richardson, Matthew Arrott: Advanced Message Queuing Protocol Protocol Specification, Cisco Systems, 2008, pp. 11-12

[13] Unknown author: AMQP v1.0 (revision 0) FINAL, Cisco Systems, 2011, pp. 23-24 http://www.amqp.org/sites/amqp.org/files/amqp.pdf

[14] I. Heđi, I. Špeh, A. Šarabok: IoT network protocols comparison for the purpose of IoT constrained networks, 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2017, pp. 502-504

[15] Nitin Naik: Choice of Effective Messaging Protocols for IoTSystems: MQTT, CoAP, AMQP and HTTP, 2017 IEEE International Systems Engineering Symposium (ISSE), 2017, pp. 2-6

[16] Hermann Merz, Thomas Hansemann, Christof Hübner: Building Automation Communication Systems with EIB/KNX, LON and BACnet, Springer, 2018, pp. 63-66

[17] Unknown author: Grundlagenwissen zum KNX Standard, KNX.ch, pp. 4-5 https://www.knx.ch/knx-chde/wdownload-d/Flyer/Endkunden/Grundlagenwissen_zum_KNX_Standard_German.pdf visited on 06.08.2018

[18] Unknown author: IEEE 802.3 standard,IEEE website, http://www.ieee802.org/3/, visited 02.09.2018

[19] Unknown author: KNX System Specifications , Web services , KNX Association, 2016, pp. 6-7

[20] Boris Malinowsky, Georg Neugschwandter, Wolfgang Kastner, Calimero: Next Generation, Automation System Group ,Institute of Automation, Vienna University of Technology ,2015, pp. 1-3 https://github.com/calimero-project/introduction/blob/master/documentation/calimero-ng.pdf

[21] Unkown author: Gamma Trainingskoffer - fit für KNX/EIB, Siemens Aktiengesellschaft, pp. 1-2 http://www.eib-home.de/software/siemens_schulungskoffer_E20001-A5620-P430.pdf, visited 26.12.2018

[22] Unkown author: ETS5, KNX website https://www2.knx.org/lu-de/software/ets/ueber/index.php?navid=848611848611http://www.eib-home.de/software/siemens_schulungskoffer_E20001-A5620-P430.pdf, visited 26.12.2018

[23] Unknown author: Web Service Exporter von KNX Association, KNX website https://my.knx.org/shop/product?product_type_category=etsapps&product_type=web-service-exporter, visited 27.11.2018

[24] Unknown author: InfluxDB v 1.6 docu, Influxdata website, https://github.com/influxdata/docs.influxdata.com/blob/master/content/influxdb/v1.6/_index.md, visited 20.08.2018,

[25] Unknown author: InfluxDB v1.8 docu, Influxdata webiste, https://github.com/influxdata/docs.influxdata.com/blob/master/content/influxdb/v1.6/introduction/getting-started.md, visited 20.8.2018

[26] Unknown author: Telegraf is the Agent for Collecting and Reporting Metrics & Data, Influxdata webiste, https://www.influxdata.com/time-series-platform/telegraf/, visited 02.8.2018

[27] Unknown author: Eclipse Mosquitt: An open source MQTT broker, Mosquitto website, https://mosquitto.org/, visited 02.8.2018

[28] Unknown author: RabbitMQ, RabbitMQ webiste, https://www.rabbitmq.com/protocols.html, visited 02.8.2018

[29] Roman Trapickin: Constrained Application Protocol (CoAP): Einführung und Überblick, Seminars FI / IITM / ACN SS2013, Network Architectures and Services, 2013, pp. 121