

Bitte Platzhalter löschen
und durch eigenes Bild
ersetzen



MQTT Device Description

Hier steht ein Untertitel

Bachelorthesis

Studiengang: Informatik
Autor: Adrian Bärtschi
Betreuer: Prof. Dr. Ing. Reto E. Koenig
Experte: Dr. Federico Flueckiger
Datum: 21.01.2016

Versionen

Version	Datum	Status	Bemerkungen
0.1	23.08.2015	Entwurf	Dokument erstellt
0.2	05.10.2015	Entwurf	Einleitung, Bestehenden Ansätze
0.3	27.10.2015	Entwurf	Kozept
0.4	15.11.2015	Entwurf	Projektmanagement
0.5	10.01.2016	Entwurf	Anforderungen, Verifikation
0.6	11.01.2016	Entwurf	Umsetzung
0.7	18.01.2016	Entwurf	Fazit, Review, Abrunden des Dokuments
1.0	21.01.2016	Final	Abgabe

Management Summary

Das Netzwerkprotokoll MQTT wird verwendet, um beliebige Geräte mit eingeschränkten Ressourcen zu vernetzen (Internet of Things). Der einfache Aufbau des Protokolls macht es jedoch schwierig, die gewünschten Daten eines Gerätes zu erhalten und richtig zu interpretieren. Im Rahmen dieser Arbeit wurde ein Konzept für ein einheitliches Format für die Beschreibung von vernetzten Geräten sowie deren Eigenschaften und Fähigkeiten erarbeitet.

Die Vision des Internet of Things beschreibt, dass in Zukunft viel mehr Alltagsgeräte vernetzt sein werden. Damit diese zahlreichen und kompakt gebauten Geräte trotz eingeschränkter Ressourcen (Speicher, Energie, Netzwerkkapazität) mit dem Internet verbunden werden können, sind neue Netzwerkprotokolle mit einfachem und leichtgewichtigem Aufbau nötig. MQTT (Message Queuing Telemetry Transport Protocol) ist ein Protokoll, welches für diesen Anwendungszweck entwickelt wurde.

MQTT funktioniert nach dem Publish/Subscribe Prinzip. Der Nachrichtenaustausch erfolgt über einen zentralen MQTT Broker, welcher die Nachrichten des Senders entgegennimmt und an die registrierten Empfänger weiterleitet.

Die Spezifikation des Protokolls macht keine Vorgaben zur Codierung oder Struktur der Nutzdaten einer MQTT Nachricht. Dies und die Entkopplung von Sender und Empfänger durch den Broker führen dazu, dass es für den Empfänger schwer ist, an die gewünschten Nachrichten zu kommen und diese richtig zu interpretieren. Anwendungen, welche bereits auf dem Markt sind, haben jeweils ihre eigenen Datenstrukturen für die Nachrichten definiert, was die verschiedenen Systeme inkompatibel zueinander macht.

Das Ziel der Thesis bestand darin, ein allgemeines Konzept für die Beschreibung von vernetzten Geräten mit MQTT zu entwickeln. Pro Gerät sollte ersichtlich sein, wie man damit interagieren kann und welche Nachrichten es erzeugen und versenden wird. Die Beschreibungssprache sollte für den Menschen gut lesbar sein und gleichzeitig von einem Programm interpretiert werden können. Ausserdem musste das System mit den vorhandenen Funktionen des MQTT Protokolls umgesetzt werden.

Es wurde eine einheitliche Beschreibung von vernetzten Geräten entworfen und anhand eines Prototypen mit Sensoren und Aktoren umgesetzt. Dabei werden für jedes Gerät Statusinformationen, zu erwartende Events und die Möglichkeiten zur Steuerung (Commands) ausgewiesen.

Zudem wurde eine Webapplikation entwickelt, welche die Beschreibungen interpretiert und dadurch eine leicht zugängliche Interaktion mit den Geräten ermöglicht.

Inhaltsverzeichnis

Management Summary	i
1. Einleitung	1
2. Projektmanagment	3
2.1. Organisation	3
2.2. Ziele	3
2.3. Meilensteine	3
2.4. Ressourcen	4
2.5. Aufwände	4
2.6. Termine / Abgabefristen	5
2.7. Ablage	5
2.8. Reflektion	5
3. Technologie	7
3.1. MQTT	7
3.2. Bestehende Umsetzungen	10
3.3. Datenformate	12
3.4. Bestehende Beschreibungssprachen	15
4. Konzept	19
4.1. Allgemein	19
4.2. Hierarchie Topics	21
4.3. Abgrenzung	22
5. Anforderungen	23
5.1. Use Cases	23
5.2. Nichtfunktionale Anforderungen	26
6. Spezifikation Device Description	29
6.1. Format	29
6.2. Primitive Datentypen	29
6.3. Schema	29
7. Umsetzung	33
7.1. MQTT Device Description Library	33
7.2. Device Browser	37
7.3. Anwendung Tinkerforge	41
7.4. Beurteilung Umsetzung	44

8. Verifikation	45
8.1. Funktionale Anforderungen	45
8.2. Nichtfunktionale Anforderungen	45
9. Schlussfolgerungen/Fazit	47
Selbständigkeitserklärung	49
Glossar	51
Literaturverzeichnis	53
Abbildungsverzeichnis	55
Tabellenverzeichnis	57
Stichwortverzeichnis	59
A. Arbeitsjournal	59

1. Einleitung

TODO: überarbeiten

Bei Systemen im Internet of Things (IoT) Umfeld sind sehr viele und auch unterschiedliche Geräte in einen Netzwerk miteinander verbunden. Für diese IoT - Machine-To-Machine Kommunikation werden andere Netzwerkprotokolle eingesetzt als im 'klassischen' Internet. Dies ist nötig, weil die Geräte stark eingeschränkte Ressourcen haben und die Netzwerke geringe Bandbreiten aufweisen.

MQTT ist ein Protokoll, welches die Anforderungen für IoT Systeme erfüllen soll. Beim Entwurf des Protokolls wurde auf Einfachheit und Leichtgewichtigkeit grossen Wert gelegt. Mit MQTT ist es möglich, fast beliebige Daten in beliebiger Codierung zu versenden. Dies bietet den Entwicklern der Systeme grosse Freiheiten.

Es ist aber ersichtlich, dass die fehlende Struktur und Beschreibung der Daten gewisse Schwierigkeiten mit sich bringen kann. Eine Anwendung, welche Daten per MQTT erhält, muss wissen wie diese vom Absender codiert wurden und was sie bedeuten.

Beispielsweise wird eine Messung eines Temperatursensors via MQTT versendet werden. Der Sensor liefert den Wert 22° Celsius. Der ganzzahlige Wert 22 wird als binärer Wert 10110 versendet. Der Empfänger erhält nun die MQTT Nachricht 10110. Er weiss aber nicht, was mit diesem Wert anzufangen ist. Wird dieser Wert nicht nach dem gleichen Schema interpretiert, wie der codiert wurde, werden Daten erzeugt, welche nicht der Ursprungsinformation entsprechen. Der Empfänger müsste wissen, dass es sich um einen Integer Wert (hier 32 bit signed) handelt, damit die Daten in das richtige Format gebracht werden können. Ausserdem muss der Empfänger jetzt noch wissen, in welcher Einheit (Celsius, Fahrenheit, etc.) die Temperatur übermittelt wird und es wäre praktisch zu wissen, wo zum Beispiel die Messung stattfand.

Die Erzeuger der Daten stellen diese Information den Entwicklern von Anwendungen zurzeit typischerweise in einem Dokument zur Verfügung. Diese sogenannte out-of-band Dokumentation ist aber aufwändig in der Nachführung und Änderungen. Auch ist es schwierig die Struktur der Daten einheitlich und klar zu erklären.

2. Projektmanagement

2.1. Organisation

Name	Rolle	Aufgaben
Prof. Dr. Ing. Reto E. König <i>Berner Fachhochschule</i>	Betreuer	Hauptansprechperson für Studierenden, verantwortlich für den Ablauf der Thesis. Beurteilung aufgrund von Aufgabenstellung und abgegebenen Artefakten.
Dr. Federico Flueckiger <i>Eidg. Finanzdepartement</i>	Experte	Beurteilung aufgrund der Aufgabenstellung und abgelieferten Artefakten sowie mindestens ein bis zwei Sitzungen mit dem Studierenden.
Adrian Bärtschi	Studierender	Selbständiges Projektmanagement während der Thesis. Setzt die Aufgaben gemäss Aufgabenstellung und Vorgaben des Betreuers um. Organisiert Kommunikation mit dem Betreuer und Experten.

Tabelle 2.1.: Involvierte Personen und deren Aufgaben

2.2. Ziele

Die Arbeit umfasst gemäss Aufgabenstellung folgende Ziele:

- Definition einer Beschreibungssprache für vernetzte Geräte.
- Die Beschreibung muss von Menschen und Computern interpretiert werden können.
- Die Beschreibung muss die Eingabe- und Ausgabeparameter der Geräte beeinhalt.
- Die Beschreibung muss soll möglichst die aktuelle MQTT Spezifikation (3.1.1) nicht verletzen.

2.3. Meilensteine

Die Aufgabenstellung wurde in Meilensteine aufgeteilt, um eine grobe Planung zu erhalten.

Datum	Meilenstein
25.09.2015	Initialisierung, Vorgehen geklärt
01.10.2015	Ziele definiert
15.10.2015	Einfacher Prototyp erstellt, Konzept der Lösung skizziert
30.11.2015	Spezifikation Device Description festgelegt
15.12.2015	Implementation Demo Applikation abgeschlossen
18.01.2016	Dokumentation inhaltlich abgeschlossen
21.01.2016	Dokumentation fertiggestellt, Präsentation vorbereitet

Tabelle 2.2.: Meilensteine der Thesis

2.4. Ressourcen

Die gesamte Thesis umfasst gemäss Modulplan der BFH einen Arbeitsaufwand von 360 Stunden. Dies beinhaltet die Konzeption und Umsetzung der Lösung, Abprachen mit Betreuer und Experte, das Erstellen der Dokumentation und die Vorbereitung der Präsentationen für den Finaltag und die Verteidigung.

Es sind keine Kosten für Softwarelizenzen oder andere Ressourcen angefallen.

2.5. Aufwände

Nachfolgend ist der Gesamtaufwand von 360 Stunden der Arbeit auf einzelnen Tasks aufgeteilt. Ausserdem werden bei Differenzen zwischen geplanten und geleisteten Stunden die Gründe dafür angegeben.

Task	Geplant [h]	Ist [h]	Bemerkungen
Projektmanagement			
Kickoff, Initialisierung	8	8	
Ziele definieren	6	8	
Planung Tasks	8	10	
Meetings Betreuer	10	12	
Meetings Experte	4	2	
Umsetzung			
MQTT Device Description Library	32		
Tinkerforge Prototyp Java	50		
Webapplikation Device Browser	40		
Dokumentation			
Setup	4		
Projektmanagement	16		
Einleitung, bestehende Umsetzungen	32		
Konzept und Anforderungen	16		
Spezifikation Device Description	16		
Umsetzung	32		
Fazit	8		
Finalisierung	24		
Präsentation			
Poster	8		
Book Seite	8		
Präsentation Finaltag	24		
Präsentation Verteidigung	14		
Total	360		

Tabelle 2.3.: Planung der Tasks und Auswertung der geleisteten Stunden

2.6. Termine / Abgabefristen

Datum	Beschreibung
04.01.2016	Abgabe elektronische Form des Posters
11.01.2016	Erfassung und Freigabe der Book Seite
21.01.2016	Abgabe Dokumentation an Betreuer, Experte und BFH
22.01.2016	Finaltag Bern, Präsentation und Ausstellung, Abgabe Dokumentation gedruckt und Source Code an Sekretariat BFH
01.02.2016	Verteidigung

Tabelle 2.4.: Termine und Fristen

2.7. Ablage

Diese Dokumentation sowie sämtlicher Source Code sind auf dem Github Repository <https://github.com/barta3/ch.bfh.bti7321thesis.mqttSemantics> verfügbar.

2.8. Reflektion

nicht klare ziele, ungewissheit Freiheit

+ Prototyping + Frühe Entwicklung + Latex dok

- Deutsche Doku

3. Technologie

3.1. MQTT

Die folgende Einführung des MQTT Protokolls wurde aus der dem Ergebnis des BFH Modul BTI7302 (Projekt 2) von Adrian Bärtschi übernommen.

MQTT (Message Queue Telemetry Transport) ist ein Netzwerkprotokoll, das sich dank einfachem und leichtgewichtigen Design sehr gut für Geräte mit stark eingeschränkten Ressourcen und Netzwerke mit geringer Bandbreite eignet.

Die erste Version von MQTT wurde 1999 von Dr. Andy Stanford-Clark (IBM) und Arlen Nipper (Arcom) beschrieben und entwickelt. Inzwischen ist MQTT in der Version 3.1.1 verfügbar [6] und wird von OASIS Konsortium (<https://www.oasis-open.org>) standardisiert.

Im OSI Modell ist MQTT auf dem Application Layer eingeordnet, basierend auf dem TCP Stack.

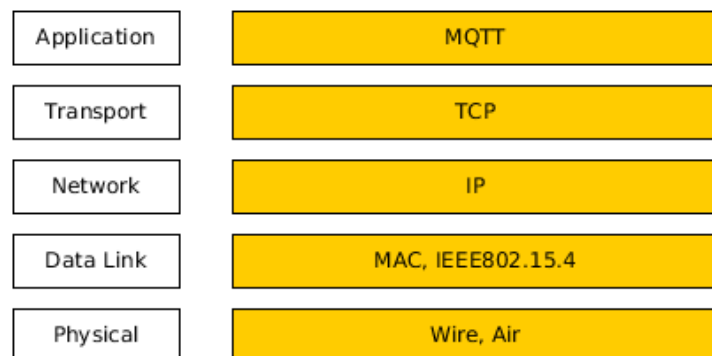


Abbildung 3.1.: MQTT im (vereinfachten) OSI Stack

3.1.1. Publish/Subscribe

MQTT funktioniert nach dem Publish/Subscribe Pattern. Im Gegensatz zum klassischen Client/Server Prinzip registrieren sich die Clients (Subscriber) bei einem Broker für bestimmte Bereiche (Topics), zu denen sie Nachrichten erhalten möchten.

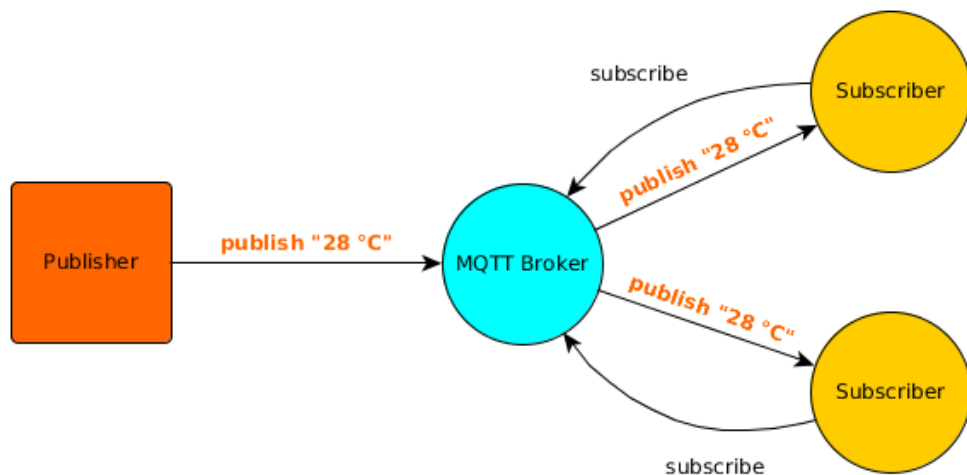


Abbildung 3.2.: Publish/Subscribe Prinzip

Ein Publisher, (z. Bsp. ein Sensor) sendet seine Nachrichten an den Broker. Alle Subscriber, die sich für das entsprechende Topic eingeschrieben haben, erhalten die Nachricht vom Broker.

Diese Entkopplung der Teilnehmer bringt diverse Vorteile mit sich:

- Publisher und Subscriber müssen sich gegenseitig nicht kennen
- Clients können sich beliebig an- und abmelden
- Beim Ausfall eines Teilnehmers sind die anderen nicht blockiert

3.1.2. Topics

Jede Nachricht wird an ein bestimmtes Topic gesendet. Grundsätzlich kann jeder Client zu jedem Topic des Brokers Nachrichten veröffentlichen. Die Topics sind hierarchisch aufgebaut, die Ebenen werden durch einen Slash (/) getrennt.

Zum Beispiel könnte ein Thermometer seine Temperatur im Topic `house/livingroom/temperature` veröffentlichen.

Um die Nachrichten zu erhalten, muss sich nun ein anderer Client für dieses Topic einschreiben. Dabei können die Wildcards `+` und `#` verwendet werden.

Das `+` Symbol steht für eine Ebene in der Topic Hierarchie.

Beispiel: `house+/temperature` steht für die Temperaturmeldungen aller Räume des Hauses .

- `house/livingroom/temperature`
- `house/kitchen/temperature`
- `house/bedroom/temperature`
- ...

Mit # werden alle Topics des Unterbaums abonniert. Beispiel: Wenn ein Client das Topic house/# abonniert, erhält dieser alle Meldungen des Hauses.

- house/bedroom/temperature
- house/door/status
- ...

3.1.3. Messages

Messages in MQTT sind sehr einfach aufgebaut. Eine Message hat folgende Attribute:

Attribut	Beschreibung
Payload	Beliebige Daten im Binärformat. Maximal 256 MB
QoS	Quality of Service 0, 1 oder 2. Details in Kapitel 3.1.4
Retained	Flag, true oder false, Details in Kapitel 3.1.5

Tabelle 3.1.: Aufbau einer MQTT Message

3.1.4. Quality of Service

MQTT bietet drei verschiedene QoS Einstellungen für das versenden von Nachrichten.

- 0: Die Nachricht wird einmal versendet, es gibt keine Bestätigung des Empfängers.
- 1: (Standard) Die Nachricht wird mindestens einmal beim Empfänger ankommen. Es wird so lange versucht zu senden, bis eine Bestätigung erhalten wurde.
- 2: Mit ein Handshake Mechanismus wird sichergestellt, dass die Nachricht genau einmal beim Empfänger angekommen ist.

Je höher die QoS Einstellung, desto mehr Ressourcen werden beim versenden benötigt und es desto mehr Daten werden über das Netzwerk gesendet.

3.1.5. Retained Messages

Bei einer MQTT Message kann das Retained Flag gesetzt werden. Damit wird pro Topic die letzte Nachricht auf dem Broker gespeichert. Verbindet sich ein neuer Client und abonniert das Topic, erhält er die retained Message sofort. Dies kann nützlich sein bei Anwendungen mit Topics, die sehr lange Pausen zwischen den Meldungen haben.

3.1.6. Last Will

Jeder Client, der sich zum Broker verbindet, kann eine "Last Will" Message angeben. Diese Meldung wird an das gewünschte Topic gesendet, sobald die Verbindung beendet wird.

3.1.7. Implementationen

Inzwischen sind zahlreiche Implementationen von sowohl MQTT Brokern, als auch Client Libraries verfügbar.

Broker

Name	Beschreibung	URL
Mosquitto	Leichtgewichtiger Open Source Broker geschrieben in C. Zurzeit de-facto standard MQTT Broker.	http://mosquitto.org/
ActiveMQ Apollo	OSS Message Broker, unterstützt neben MQTT noch andere Protokolle. Konfiguration und Administration über Web GUI.	http://activemq.apache.org/apollo/
Moquette	OSS Broker Implementation in Java. Kann auch als Library in eigene Projekte eingebunden werden.	https://github.com/andsel/moquette
HiveMQ	Proprietärer MQTT Broker, ausgerichtet für Enterprise Anwendungen.	http://www.hivemq.com/
Mosca	MQTT Broker für die NodeJS Plattform. Kann als Standalone oder Embedded Broker verwendet werden.	http://www.mosca.io/

Client Libraries Das Eclipse Projekt Paho [2] stellt Client Libraries für C, Java, Android, Python, Javascript, C/C++ embedded und .Net / WinRT zur Verfügung. Die Java Library sich als stabil und gut dokumentiert erwiesen.

Eine Liste mit weiteren Client Libraries ist im MQTT Wiki [5] zu finden.

3.2. Bestehende Umsetzungen

Die verschiedenen Hersteller von MQTT Anwendungen entwickeln jeweils ihre eigenen Ansätze, um die Daten zu strukturieren.

3.2.1. IBM Internet of Things Foundation

IBM hat unter dem Brand 'IBM IoT Foundation' [3] einen Dienst entwickelt, mit dem vernetzte Geräte verwaltet werden können. Als Kommunikationsprotokoll wird MQTT eingesetzt. Die Plattform verwendet folgende konzeptionelle Ideen:

- Organizations: Eindeutige Identifikation der Kunden der Plattform
- Devices: Beliebiges vernetztes Gerät. Versendet Events und reagiert auf Commands.
- Applications: Anwendung, welche mit den Daten der Devices interagiert.
- Events: Daten, welche von den Devices an die Plattform gesendet werden
- Commands: Applications können mittels Commands mit den Devices kommunizieren.

Events

Events müssen an ein definiertes Topic nach folgendem Schema gesendet werden:

`iot-2/evt/<event_id>/fmt/<format_string>`

Beispiel: `iot-2/evt/temperature_outdoor/fmt/json`

Eine Anwendung, welche Events empfangen möchte, muss sich auf ein Topic in der Form

`iot-2/type/<device_type>/id/<device_id>/evt/<event_id>/fmt/<format_string>` registrieren. Die Teile `device_type`, `device_id`, `event_id` und `format_string` des Topics können auch mit dem Wildcard Charakter '+' ersetzt werden, um jeweils alle Events der Komponenten zu erhalten.

Beispiel: `iot-2/type/temp/id/+/evt/temperature_outdoor/fmt/+`

Commands

Um einen Command zu erzeugen, sendet eine Anwendung eine MQTT Message mit Topic gemäss folgenden Schema: `iot-2/type/<device_type>/id/<device_id>/cmd/<command_id>/fmt/<format_string>`

Beispiel: `iot-2/type/temp/id/sensor1/cmd/setInterval/fmt/json`

Das Device `sensor1` würde damit eine Message auf Topic `iot-2/cmd/setInterval/fmt/json` erhalten.

Payload Format

Grundsätzlich unterstützt IBM IoT Foundation ein beliebiges Payload Format. Es wird jedoch empfohlen, JSON zu verwenden. Um alle Funktionen der Plattform nutzen zu können, müssen die JSON Dokumente zusätzlich nach den Vorgaben [4] von IBM strukturiert sein.

```
{
  "d": {
    "host": "IBM700-R9E683D",
    "mem": 54.9,
    "network": {
      "up": 1.22,
      "down": 0.55
    },
    "cpu": 1.3,
  }
}
```

Listing 1: JSON Beispiel im IBM IoT Foundation Payload Format

3.2.2. Tinkerforge MQTT Proxy

Tinkerforge hat ein modulares System von Sensoren und Aktoren (so genannte Bricklets) entwickelt, die u. A. für Prototyping und in der Ausbildung (auch an der BFH) eingesetzt werden. Um die Module zu steuern, wird klassischerweise das bereitgestellte SDK in der gewünschten Programmiersprache verwendet. Tinkerforge ausserdem eine Anwendung entwickelt und die Bausteine per MQTT ansprechen zu können [1].

Topics

Die Tinkerforge Devices senden ihre Daten an ein MQTT Topic nach Schema `tinkerforge/<prefix>/<uid>/<suffix>`.

Ein Temperatur Bricklet mit Unique Identifier (UID) xf2 würde also den gemessenen Wert an das Topic `tinkerforge/bricklet/temperature/xf2/temperature` senden.

Die Bricklets reagieren auf Messages die an ein passendes Topic mit Suffix `/set` gesendet werden. Sollen beispielsweise die LEDs des Dualbutton Bricklets mit UID mxg eingeschaltet werden, muss eine Message an das Topic `tinkerforge/bricklet/dual_button/mxg/led_state/set` gesendet werden mit folgendem Payload:

```
{
  "led_l": 2,
  "led_r": 2
}
```

Listing 2: JSON Beispiel Tinkerforge Format

Payload Format

Die Tinkerforge MQTT Komponente verwendet JSON als Datenformat für die Messages. Jede Message, die von einem Bricklet gesendet wird, enthält unter dem Key `_timestamp` den Zeitpunkt der Erzeugung als UNIX Timestamp.

```
{
  "_timestamp": 1440083842.785104,
  "temperature": 2343
}
```

Listing 3: JSON Beispiel Tinkerforge Format

Die Beschreibung, unter welchen Topics Daten publiziert werden und wie die Bricklets angesprochen werden können, ist in der Dokumentation von Tinkerforge [1] beschrieben.

3.3. Datenformate

In diesem Kapitel werden verschiedene Datenformate kurz beschrieben, welche für die Beschreibung der Devices in Frage kommen könnten.

Grundsätzlich muss zwischen textbasierten und binären Formaten unterschieden werden.

Textbasierte Formate bieten den Vorteil, dass sie von Menschen ohne weitere Hilfsmittel gelesen und meistens auch interpretiert werden können. Die textuellen Formate haben jedoch Nachteile, wenn binären Daten (Bilder, Video, Audio etc.) codiert werden sollen. Zwar ist es möglich, die binären Daten in ein textbasiertes Format zu integrieren (beispielsweise mittels Kovertierung zu Base64), dies führt aber zur Aufblähung der Datenmenge.

Binäre Formate haben den Vorteil, dass die Daten bei der Serialisierung dichter codiert werden können, somit werden die Dateien meistens kleiner als bei textbasierten Formaten. Der grosse Nachteil ist, dass man für die Interpretation von binären Daten immer auf eine Beschreibung (in Form eines Schemas) angewiesen ist.

JSON

JSON ist ein textbasiertes Datenformat, welches Key-Value Paare für die Strukturierung nutzt. Dank dem einfachen Aufbau ist es sehr weit verbreitet und Libraries sind für alle populären Plattformen verfügbar.

```
{
  "firstName": "Max",
  "lastName": "Muster",
  "age": 25,
  "address": {
    "street": "Bundeshaus",
    "plz": "3000",
    "city": "Bern"
  }
}
```

Listing 4: JSON Beispiel, (148 byte)

YAML

YAML ist dem JSON Format sehr ähnlich, ist aber stärker auf Lesbarkeit ausgerichtet. Für die Strukturierung der Daten kann die kompakte Schreibweise mit Leerzeichen verwendet werden. YAML ist ein Superset von JSON, das bedeutet jedes gültige JSON Dokument ist auch ein gültiges YAML Dokument.

```
---
firstName: Max
lastName: Muster
age: 25
address:
  street: Bundeshaus
  postalCode: 3000
  city: Bern
```

Listing 5: YAML Beispiel

TODO: Glossar Einträge ab hier

XML

Mit XML werden die Informationen in hierarchische Elemente (Tags) verschachtelt, welche mit Attributen versehen werden können. XML Dokumente werden dadurch rasch sehr gross und haben viel redundante Daten. Häufig werden XML Schemas eingesetzt, um die gewünschte Struktur eines Dokuments und dessen Datentypen zu beschreiben.

```

<person>
  <firstName>Max</firstName>
  <lastName>Muster</lastName>
  <age>25</age>
  <address>
    <street>Bundeshaus</streetAddress>
    <plz>3000</plz>
    <city>Bern</city>
  </address>
</person>

```

Listing 6: XML Beispiel

Protocol Buffers

Protocol Buffers ist ein Mechanismus zur Serialisierung von Daten. Der Entwickler muss ein Schema seiner Datenstrukturen erstellen (.proto Datei).

```

message Person {
  required string firstName = 1;
  required string lastName = 2;
  required int32 age = 3;

  message Address {
    required string street = 1;
    required string plz = 2;
    required string city = 3;
  }

  required Address address = 4;
}

```

Listing 7: Protocol Buffer (v2) Schema Beispiel

```

00000000  0a 03 4d 61 78 12 06 4d 75 73 74 65 72 18 19 22 |..Max..Muster.."|
00000010  18 0a 0a 42 75 6e 64 65 73 68 61 75 73 12 04 33 |...Bundeshaus..3|
00000020  30 30 30 1a 04 42 65 72 6e                                |000..Bern|
00000029

```

Listing 8: Beispiel Protocol Buffer Daten (Output hexdump)

Anschliessend werden mit den bereitgestellten Tools (Protocol Buffer Compiler) Klassen für den Datenzugriff in der gewünschten Sprache generiert. Mit diesen Klassen können nun Daten in ein sehr kompaktes binäres Format codiert und wieder decodiert werden.

3.3.1. Vergleich

Bei der Wahl eines geeigneten Datenformats gibt es viele Faktoren, die eine Rolle spielen.

Falls die Dateigrösse der encodierten Daten möglichst gering sein soll, empfiehlt sich ein binäres Format. Bei den oben gezeigten Beispielen beinhalten alle Dateien die gleichen Informationen, bei der Dateigrösse gibt es aber gewichtige Unterschiede:

Protocol Buffer:	41 Byte
YAML:	107 Byte
JSON:	148 Byte
XML:	200 Byte

Für Dateien, welche für Menschen gut lesbar sein sollen, eignet sich aufgrund der Formatierung YAML am besten.

Ein weiterer wichtiger Faktor ist, dass auf den eingesetzten Plattformen Libraries für das Format zur Verfügung stehen. Da JSON und XML am weitesten verbreitet sind, ist für diese beiden Formaten auch die grösste Auswahl an Libraries zu finden. Für alle vier der hier behandelten Formaten sind inzwischen Libraries für alle gängigen Plattformen verfügbar.

3.4. Bestehende Beschreibungssprachen

Es gibt diverse bestehende Sprachen zur Beschreibung von Schnittstellen. Vor allen bei Webservices werden diese häufig eingesetzt.

3.4.1. WSDL

Für die Beschreibung von SOAP Webservices wird eine WSDL Datei verwendet. WSDL ist eine auf XML basierende Sprache und beschreibt die Methoden und Datentypen des Webservice. Das Schema wird verwendet, um daraus Zugriffsklassen zu generieren und für die Validierung der Requests.

```

<definitions name="Demo"
  targetNamespace="http://example.com/demo.wsdl"
  xmlns:tns="http://example.com/demo.wsdl"
  xmlns:xsd1="http://example.com/demo.xsd"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
    <schema targetNamespace="http://example.com/demo.xsd"
      xmlns="http://www.w3.org/2001/XMLSchema">
      <element name="HelloRequest">
        <complexType>
          <all><element name="user" type="string"/></all>
        </complexType>
      </element>
      <element name="HelloResponse">
        <complexType>
          <all><element name="msg" type="String"/></all>
        </complexType>
      </element>
    </schema>
  </types>

  <message name="SayHelloInput">
    <part name="body" element="xsd1:HelloRequest"/>
  </message>

  <message name="SayHelloOutput">
    <part name="body" element="xsd1:HelloResponse"/>
  </message>

  <portType name="DemoPortType">
    <operation name="SayHello">
      <input message="tns:SayHelloInput"/>
      <output message="tns:SayHelloOutput"/>
    </operation>
  </portType>

  <binding name="DemoSoapBinding" type="tns:DemoPortType">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="SayHello">
      <soap:operation soapAction="http://example.com/SayHello"/>
      <input><soap:body use="literal"/></input>
      <output><soap:body use="literal"/></output>
    </operation>
  </binding>

  <service name="DemoService">
    <documentation>My first service</documentation>
    <port name="DemoPort" binding="tns:DemoSoapBinding">
      <soap:address location="http://example.com/demo"/>
    </port>
  </service>

</definitions>

```

Listing 9: Beispiel WSDL eines SOAP Webservice

3.4.2. Swagger Spec

Webservices, welche nach dem RESTful Architekturstil umgesetzt sind, können u.A. mithilfe eines Schemas von Swagger beschrieben werden. Ein Schema in JSON oder YAML gibt an, welche Funktionalität der Webservice zur Verfügung stellt. Mithilfe eines Code Generators können aus dem Schema Zugriffsklassen für den Webservice generiert werden.

```
swagger: "2.0"
info:
  version: "1.0"
  title: "Hello World API"
paths:
  /hello/{user}:
    get:
      description: My first service
      parameters:
        - name: user
          in: path
          type: string
          required: true
          description: The name of the user to greet.
      responses:
        200:
          description: Returns the greeting.
          schema:
            type: string
        400:
          description: Invalid characters in "user" were provided.
```

Listing 10: Beispiel Swagger Beschreibung eines Webservices

4. Konzept

TODO: Einleitung

4.1. Allgemein

Um ein Internet of Things (IoT) Systems zu realisieren, sind viele Komponenten nötig, welche miteinander interagieren müssen. In diesem wird Abschnitt beschrieben, aus welchen Teilen ein solches System besteht und welche Teile die Beschreibung eines Devices enthalten soll.

Device

Sensoren und Aktoren werden unter dem Begriff Device zusammengefasst.

IoT Gateway

Die einzelnen Devices eines IoT Systems sind an einen Gateway angeschlossen. Dies kann fix per Kabel oder über eine Drahtlosschnittstelle wie z. Bsp. Bluetooth umgesetzt sein. Typischerweise werden Kleincomputer (Single Board Computer) wie ein Raspberry Pi oder Intel Galileo eingesetzt. Diese Geräte zeichnen sich aus durch sehr kompakte Bauform (Kreditkartenformat) und bieten trotzdem genügend Ressourcen um darauf hardwareunabhängige Anwendungen (z. Bsp. Java, Python, Javascript) laufen zu lassen. Ein Gateway führt die Daten der einzelnen Devices zusammen und vereinheitlicht die Formate der Daten. Ausserdem übernimmt er die Kommunikation mit der Aussenwelt, indem er eine Verbindung zum MQTT Broker herstellt.

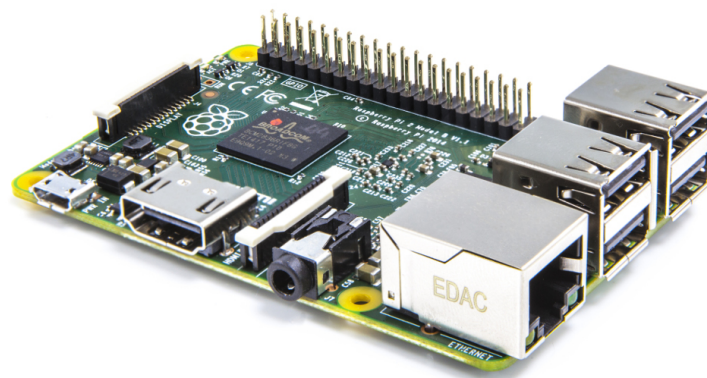


Abbildung 4.1.: Beispiel eines Gateways: Raspberry Pi 2

MQTT-Broker

Der Broker ist bei MQTT Anwendungen von zentraler Bedeutung, da alle Daten via Broker zwischen den Teilnehmern ausgetauscht werden. Pro System gibt es typischerweise einen Broker. Dieser muss für alle Teilnehmer (Gateways und Applikationen) einfach zu erreichen sein und muss eine hohe Verfügbarkeit aufweisen.

Applikation

Es können grundsätzlich beliebige Applikationen auf allen gängigen Plattformen und Technologien mit den System kommunizieren. Die einzige Grundbedingung ist, dass die Applikation MQTT fähig ist, was sich durch das Einbinden der entsprechenden Libraries einfach realisieren lässt.

Device Description

Ein Device wird als ganzes mit einem Schema beschrieben. Die Device Description wird als retained MQTT Message beim Start des Devices resp. Gateways an den Broker geschickt.

Bei der Beschreibung eines einzelnen Devices werden folgende drei Bereiche unterschieden:

State

Die Menge aller Eigenschaften des Devices und deren Werte wird als State bezeichnet. Beispielsweise Name, Firmwareversion, etc. Der State eines Devices ist beständig, d.h. solange keine Interaktion geschieht, verändert sich der State nicht.

Events

Tritt auf dem Device ein Ereignis ein, wird dadurch ein Event erzeugt. Ein Event wird grundsätzlich vom Device selbst ausgelöst. Beispielsweise erzeugt ein Temperatursensor alle 5 Sekunden ein Event, welches den Messwert enthält. Applikationen registrieren sich, um bestimmte Events von den Devices zu erhalten.

Commands

Eine Applikation interagiert mit einem Device, indem Commands an eines oder mehrere Devices gesendet werden. Die Devices empfangen die Commands und reagieren entsprechend. Ein Command ist bestimmt durch einen Namen und Parameter mit dazugehörigen Werten. Beispielsweise kann bei einem Temperatursensor über einen Command `SetInterval: 10s` der Abstand der Messungen eingestellt werden. Ein Device muss bekanntgeben, auf welche Commands mit welchen Parametern es reagiert.

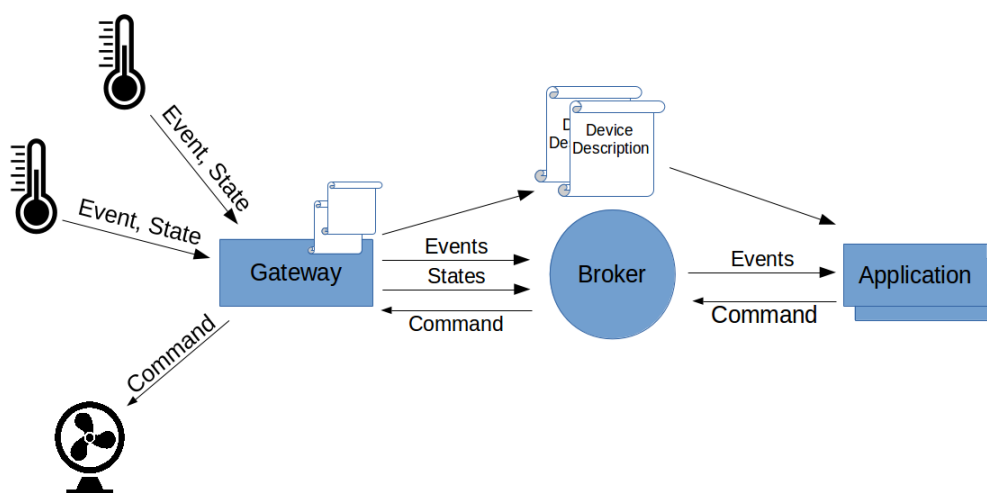


Abbildung 4.2.: Systemübersicht

4.2. Hierarchie Topics

Die Topic Hierarchie wird nach folgenden Muster aufgebaut:

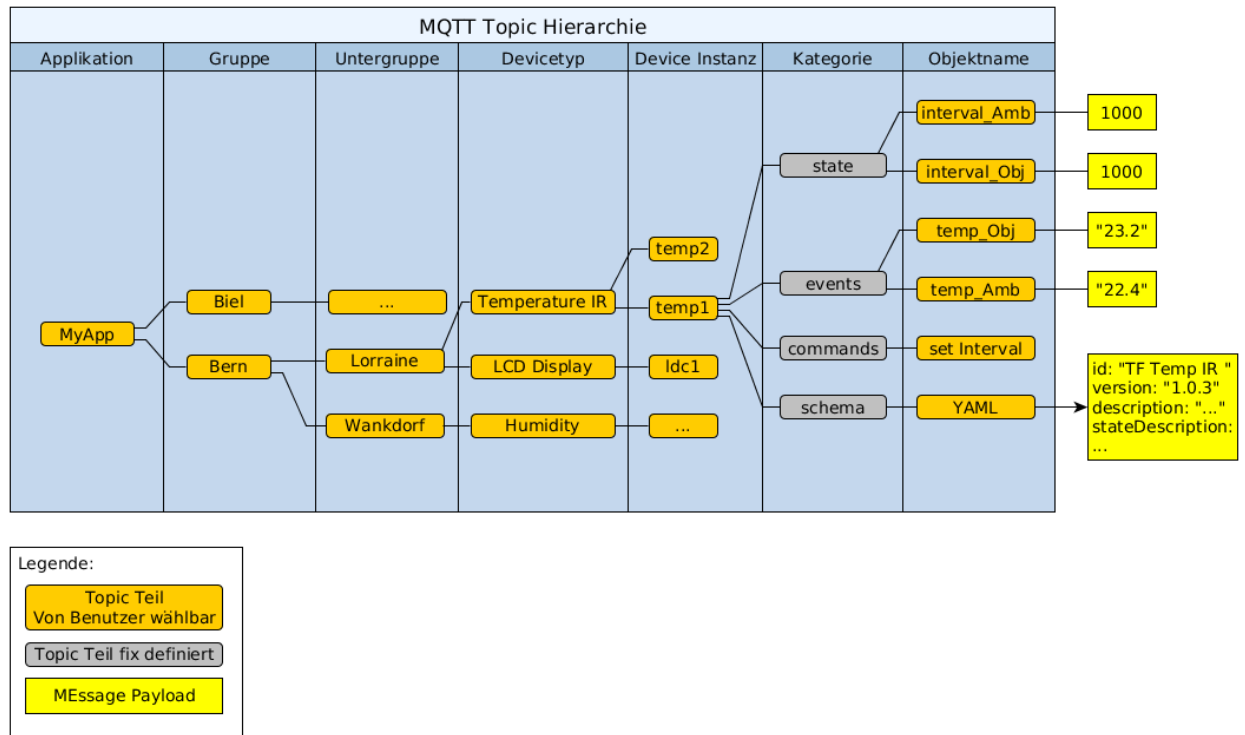


Abbildung 4.3.: Aufbau MQTT Topics mit Beispieldaten

Level	Beschreibung	Beispiel
0	Applikation Identifikation der Anwendung.	thesisDemo
1	Gruppe Gruppierung der Devices auf Stufe 1	Bern
2	Untergruppe Gruppierung der Devices auf Stufe 2	Lorraine
3	Devicetyp Bezeichnung, um was für einen Typ von Sensor oder Aktor es sich handelt.	Temperatursensor
4	Device Instanz Da mehrere Devices vom selben Typ im Einsatz sein können, wird auf dieser Stufe mit einer eindeutigen ID der konkrete Sensor resp. Aktor angegeben.	temp1
5	Kategorisierung Devicedaten Die Eigenschaften und Daten werden in die nachfolgenden Teile aufgegliedert.	events, state, commands, schema
6	Objektname Name es State, Events oder Commands	temp_Obj
6	Schema Format Als Unterobjekt von 'schema', wird hier das Format des Schemas angegeben	YAML, JSON

Tabelle 4.1.: Topic Hierarchie

4.3. Abgrenzung

4.3.1. Security

Der Faktor Security wird in diesem Konzept bewusst nicht behandelt. Es wird davon ausgegangen, dass alle Teilnehmer des Systems dazu berechtigt sind und innerhalb des Systems nur Nachrichtenaustausch nach dem Beschriebenen Konzept vorkommt.

4.3.2. Optimierungen Datenmenge

Bei der strukturierung der Device Topic Hierarchie und dem Format der Payload Daten (Schema, State, etc.) ist im Rahmen dieser Arbeit Verständlichkeit und Übersichtlichkeit höher priorisiert als die Optimierung auf möglichst geringe Datenmengen.

5. Anforderungen

5.1. Use Cases

Die funktionalen Anforderungen werden in diesem Kapitel mit Use Cases beschrieben.

TODO: UC Diagram

Aktoren

In den Use Case Beschreibungen werden die folgenden Aktoren als Benutzer des Systems verwendet:

Device Integrator: Erstellt die Device Description und integriert die Device Description Library

Entwickler Fremdsystem: Ist als interessiert an den Daten und Funktionen der Devices.

5.1.1. UC01 - Beschreibung Device

Use Case ID	UC01: Beschreibung Device
Beschreibung	In einem Schema müssen die Metadaten der Status- Event- und Commandinformationen aufgeführt werden können.
Aktor	Device Integrator
Auslöser	Device Description wird erstellt
Vorbedingungen	Funktionalität des Devices ist bekannt
Ablauf	1. Benutzer erstellt Beschreibung des Device mit der Java MQTT Device Library oder manuell 2. Die Anwendung published das Schema des Devices als retained MQTT Message.
Nachbedingungen	De Device Description ist auf dem MQTT Broker hinterlegt und kann abgefragt werden.

Tabelle 5.1.: UC01: Beschreibung Device

5.1.2. UC02 - Angabe Datentypen

Use Case ID	UC02: Angabe Datentypen
Beschreibung	Für die State- Event- und Commandangaben muss definiert werden können, welche Datentypen die Werte aufweisen.
Aktor	Device Integrator
Auslöser	Device Description wird erstellt.
Vorbedingungen	API für das Ansprechen des Devices ist bekannt.
Ablauf	Benutzer definiert in der Device Description für die State, Event und Commandobjekte die Datentypen.
Nachbedingungen	Die Datentypen der Description Objekte sind im Schema integriert.

Tabelle 5.2.: UC02: Angabe Datentypen

5.1.3. UC03 - Definition Range

Use Case ID	UC03: Definition Range
Beschreibung	Bei der Beschreibung eines Datentypen muss angegeben werden können, in welchem Bereich (Minimum, Maximum) der Wert sein kann.
Aktor	Device Integrator
Auslöser	Device Description wird erstellt.
Vorbedingungen	Datentyp und Range des Wertes ist bekannt.
Ablauf	Benutzer definiert zu einem Datentypen den Range, in dem der Wert liegen kann.
Nachbedingungen	Datentyp ist mit der Range Information ergänzt.

Tabelle 5.3.: UC03: Definition Range

5.1.4. UC04 - Definition Enum

Use Case ID	UC04: Definition Enum
Beschreibung	Es muss möglich sein, einen Datentypen als Auswahl aus einer fixen Liste (Enum) zu definieren.
Aktor	Device Integrator
Auslöser	Device Description wird erstellt.
Vorbedingungen	Benutzer möchte Datentyp als Auswahl einer festen Liste abbilden.
Ablauf	Benutzer definiert für einen Datentypen die Liste der möglichen Werte.
Nachbedingungen	In der Device Description sind alle möglichen Werte als Auswahl angegeben, welche der Datentyp annehmen kann.

Tabelle 5.4.: UC04: Definition Enum

5.1.5. UC05 - Gruppierung Devices

Use Case ID	UC05: Gruppierung Devices
Beschreibung	Die Devices müssen so in die Topic Hierarchie eingegliedert werden, dass sie benutzerdefiniert gruppiert werden können.
Aktor	Device Integrator
Auslöser	Descriptions der Devices werden published.
Vorbedingungen	Devices sind bereit und Descriptions wurden erstellt.
Ablauf	1. Benutzer definiert die Attribute für die Gruppierung der Devices (Gruppe, Untergruppe, Device Typ) mit er Java MQTT Device Library. 2. MQTT Device Description Library erzeugt aus den Gruppierungsattributen der Devices die Topics für das publishing der Daten und Descriptions.
Nachbedingungen	Die Devices sind nach den Anforderungen des Benutzers gruppiert.

Tabelle 5.5.: UC05: Gruppierung Devices

5.1.6. UC06 - Konfiguration Device Description Library

Use Case ID	UC06: Konfiguration
Beschreibung	Die Device Description Library muss so aufgebaut sein, dass Optionen wie das Format des Schemas oder die Broker URL konfigurierbar sind.
Aktor	Device Integrator
Auslöser	Library wird initialisiert.
Vorbedingungen	Angaben für die Konfiguration sind bekannt.
Ablauf	Beim der Initialisierung der Library werden die Angaben für Applikations Id, Broker URL, Schema Format, etc. gesetzt.
Nachbedingungen	Library ist initialisiert und bereit für die Verwendung.

Tabelle 5.6.: UC06: Konfiguration Device Description Library

5.1.7. UC07 - Anzeige Devices

Use Case ID	UC07: Anzeige Devices
Beschreibung	Es muss möglich sein, in einer Weboberfläche eine Liste mit den registrierten Devices anzuzeigen.
Aktor	Entwickler Fremdsystem
Auslöser	Webapplikation zur Anzeige der Devices wird geöffnet.
Vorbedingungen	Device Descriptions wurden published.
Ablauf	1. Benutzer öffnet Webapplikation zur Anzeige der Devices. 2. Webapplikation verbindet sich auf MQTT Broker und empfängt die hinterlegten Device Descriptions. 3. Webapplikation stellt die gefundenen Devices in einer Liste dar.
Nachbedingungen	Der Benutzer hat eine Übersicht über die vorhandenen Devices.

Tabelle 5.7.: UC07: Anzeige Devices

5.1.8. UC08 - Anzeige Device Description

Use Case ID	UC08: Anzeige Device Description
Beschreibung	Es muss möglich sein, in einer Weboberfläche zu einem Device die Description anzuzeigen.
Aktor	Entwickler Fremdsystem
Auslöser	Der Benutzer will Description eines Devices einsehen.
Vorbedingungen	Mindestens eine Devicedescription ist vorhanden und wird in der Weboberfläche angezeigt (UC07)
Ablauf	1. Benutzer wählt gewünschtes Device aus der Liste. 2. Webapplikation zeigt die Description des Devices im Klartext an. 3. Webapplikation interpretiert die Description und erstellt eine Übersichtliche Darstellung für den Benutzer
Nachbedingungen	Device Description wird als Klartext und in interpretierter Form angezeigt.

Tabelle 5.8.: UC08: Anzeige Device Description

5.1.9. UC09 - Anzeige Eventdaten eines Devices

Use Case ID	UC09: Anzeige Eventdaten
Beschreibung	Es muss möglich sein, in der Weboberfläche die Daten der Events anzuzeigen, welche das Device erzeugt.
Aktor	Entwickler Fremdsystem
Auslöser	Der Benutzer möchte die Daten eines Events anzeigen.
Vorbedingungen	Der Benutzer hat ein Device gewählt (UC08). Devicedescription hat mindestens ein Event. Device ist online und versendet Events.
Ablauf	1. Der Benutzer sucht im Bereich 'Events' den gewünschten Eintrag und registriert sich für den Empfang der Eventdaten. 2. Die Webapplikation stellt Verbindung zum Broker her und empfängt die gewünschten Events. 3. Die Webapplikation zeigt die empfangenen Daten dem Benutzer an.
Nachbedingungen	Der Benutzer erhält die Eventdaten des Devices.

Tabelle 5.9.: UC09: Anzeige Eventdaten eines Devices

5.1.10. UC10 - Versenden eines Commands

Use Case ID	UC10: Anzeige Eventdaten
Beschreibung	Es muss möglich sein, in der Weboberfläche einen Command an ein Device zu senden.
Aktor	Entwickler Fremdsystem
Auslöser	Der Benutzer möchte einen Command an ein Device senden.
Vorbedingungen	Der Benutzer hat ein Device gewählt (UC08). Devicedescription stellt mindestens einen Command zur Verfügung. Device ist online und kann den Command empfangen.
Ablauf	1. Der Benutzer sucht im Bereich 'Commands' den gewünschten Eintrag und gibt den Inhalt des Commands an. 2. Der Benutzer löst das Versenden des Commands aus. 3. Die Webapplikation leitet den Command an den MQTT Broker weiter. 4. Das Device empfängt den Command und reagiert entsprechend. 5. Falls durch den Command der State des Devices geändert hat, wird der State neu published.
Nachbedingungen	Das Device hat den Command empfangen und darauf reagiert.

Tabelle 5.10.: UC10: Versenden eines Commands

5.2. Nichtfunktionale Anforderungen

5.2.1. Erweiterbarkeit Devices

Die Lösung soll so gestaltet sein, dass es möglich ist, Devices von unterschiedlichen Herstellern einzubinden.

5.2.2. Erweiterbarkeit Datenformate

Die Lösung soll so gestaltet sein, dass es möglich ist, verschiedene Datenformate für die Beschreibungen und Nutzdaten der Devices zu verwenden.

5.2.3. Einfache Installation

Das System soll für den Anwender einfach zu installieren und zu konfigurieren sein.

5.2.4. Kompatibilität

Die Lösung soll so gestaltet sein, dass es von allen MQTT fähigen Applikationen genutzt werden kann, unabhängig davon ob die bereitgestellte Device Description Library genutzt wird.

6. Spezifikation Device Description

Dieses Kapitel beschreibt, wie eine Device Description aufgebaut ist.

6.1. Format

Die Description wird als UTF-8 String im Payload der MQTT Message versendet. Als Datenformat kann JSON oder YAML verwendet werden. Die Beschreibungen sind Case sensitiv.

6.2. Primitive Datentypen

Die Device Description ist als verschachteltes Objekt aufgebaut. Die Werte der Attribute können entweder ein weiteres Objekt oder einen der folgenden primitiven Datentypen enthalten.

Datentyp	Beschreibung	Minimum	Maximum
Integer	32 Bit ganzzahlig, signed	-2^{31}	$2^{31}-1$
Long	64 Bit ganzzahlig, signed	-2^{63}	$2^{63}-1$
Float	32-bit IEEE 754 floating point, single-precision	1.4×10^{-45}	3.4028235×10^{38}
Double	64-bit IEEE 754 floating point, double-precision	4.9×10^{-324}	$7976931348623157 \times 10^{308}$
String	UTF-8 String		

Tabelle 6.1.: Primitive Datentypen

Die Definition der primitiven Datentypen orientiert sich an den Datentypen von Java.

6.3. Schema

6.3.1. DeviceDescription Objekt

Hintweis optionale Felder

Feld	Datentyp	Beschreibung
id	String	Identifikation des Devices
version	String	API Version des Devices
description	String	Allgemeine Beschreibung des Devices
stateDescription	StateDescription	
eventDescription	EventDescription	
commandDescription	CommandDescription	
complexTypes	ComplexTypes	

Tabelle 6.2.: DeviceDescription Objekt Schema

6.3.2. StateDescription

Feld	Datentyp	Beschreibung
states	Liste State	Auflistung von State Objekten

Tabelle 6.3.: StateDescription Objekt Schema

State

Feld	Datentyp	Beschreibung
name	String	Pflichtfeld. Bezeichnung des State-Eintrages. Wird gleichzeitig als Subtopic für den eigentlichen Wert genutzt.
range TODO:naming	Range	Pflichtfeld. Information zum Wert des State.
options	Enum	Wird verwendet, falls der State Wert eine Auswahl aus einer fixen Menge ist.
complexTypeRef	String	Falls der Wert des State mit einem komplexen Typen abgebildet wird, wird mit diesem Feld der Name des Typs angegeben. siehe TODO
description	String	Allgemeine Beschreibung des State.

Tabelle 6.4.: State Objekt Schema

6.3.3. EventDescription

Feld	Datentyp	Beschreibung
events	Liste Event	Auflistung von Event Objekten

Tabelle 6.5.: EventDescription Objekt Schema

Event

Feld	Datentyp	Beschreibung
name	String	Pflichtfeld. Name des Events. Wird als Subtopic verwendet, auf dem das Event verschickt wird.
range	Range	Typinformationen und ev. Einschränkungen, welche den Wert des Events beschreiben
options	Enum	Wird verwendet, falls der Event Wert eine Auswahl aus einer fixen Menge ist.
description	String	Beschreibung des Events
complexTypeRef	String	Falls der Wert des Events mit einem komplexen Typen abgebildet wird, wird mit diesem Feld der Name des Typs angegeben. siehe TODO

Tabelle 6.6.: Event Objekt Schema

6.3.4. CommandDescription

Feld	Datentyp	Beschreibung
commands	Liste Command	Auflistung von Command Objekten

Tabelle 6.7.: CommandDescription Objekt Schema

Command

Feld	Datentyp	Beschreibung
name	String	Pflichtfeld. Name des Commands. Subtopic, an welches der Command gesendet werden muss.
linkedState	String	Gibt an, welche States durch das Senden des Commands beeinflusst werden können.
options	Enum	Wird verwendet, falls der Command eine fix definierte Menge an Werten beinhalten kann.
Fehlt: description	String	Beschreibung, was mit dem Command ausgelöst werden kann.
complexTypeRef	String	Falls der Wert des Events mit einem komplexen Typen abgebildet wird, wird mit diesem Feld der Name des Typs angegeben. siehe TODO

Tabelle 6.8.: Command Objekt Schema

6.3.5. Range

asdf

Feld	Datentyp	Beschreibung
topic	String	

Tabelle 6.9.: State Description Schema

6.3.6. Enum

asdf

Feld	Datentyp	Beschreibung
topic	String	

Tabelle 6.10.: State Description Schema

Tabelle mit Felder und Beschreibung

7. Umsetzung

Die Umsetzung hatte drei Module als Ergebnis.

7.1. MQTT Device Description Library

Um die Integration von Devices und das Erstellen von Device Description zu vereinfachen, wird eine Java Library entwickelt. Damit ist es möglich, Anwendungen für IoT Gateways in Java zu entwickeln. Zusammen mit der

Die Library übernimmt folgende Aufgaben:

- Handling der MQTT Verbindung zum Broker
- Versenden von Event- Command-, und State Informationen mittels MQTT Messages
- Einheitliche Abbildung der Devices und Topics
- Hilfsklassen zur Erstellung der Device Descriptions

Die Parameter der Library (MQTT Broker, Application ID, etc.) müssen vom Aufrufer der Library konfiguriert werden können.

7.1.1. Klassendiagramm

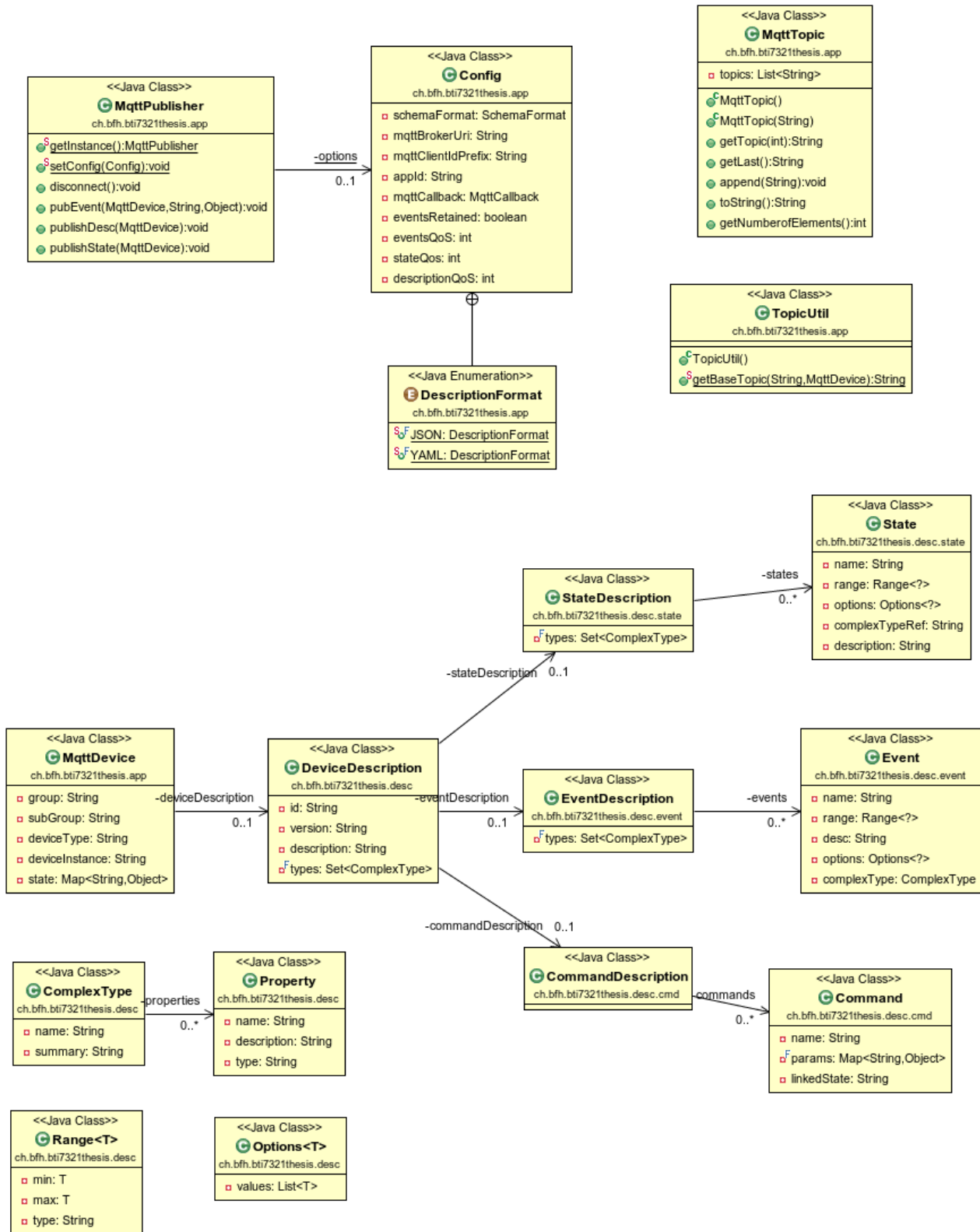


Abbildung 7.1.: Klassendiagramm MQTT Device Description Library

Klasse	Zweck
MqttPublisher	Hauptklasse der Library. wird verwendet für das versenden der MQTT Nachrichten.
Config	Wird verwendet, um bei der Initialisierung von MqttPublisher die Konfiguration abzugeben
DescriptionFormat	Enum mit der Angaben zu den möglichen Formaten der Device Description.
MqttTopic	Hilfsklasse die Abbildung von MQTT Topics.
TopicUtil	Hilfsklasse für die Erstellung von MQTT Topics.
MqttDevice	Klasse für die Abbildung von Devices.
DeviceDescription	Enthält alle Attribute der DeviceDescription.

Tabelle 7.1.: Beschreibung der Klassen

7.1.2. Verwendung

Die Device Description Library muss als erstes im eigenen Java Projekt eingebunden werden. Dies kann manuell mit der generierten .jar Datei (ch.bfh.barta3.mqttdevicedescription-1.0.0.jar) oder als Dependency Deklaration eines Maven Projektes getan werden.

```
<dependency>
  <groupId>ch.bfh.barta3.mqttdevicedescription</groupId>
  <artifactId>ch.bfh.barta3.mqttdevicedescription</artifactId>
  <version>1.0.0</version>
</dependency>
```

Listing 11: Maven Dependency MQTT Device Description Library

Als erstes muss die Konfiguration erstellt und an die Klasse MqttPublisher übergeben werden. Danach muss das Device inklusive Description erstellt werden. Sobald dies erfolgt ist, ist kann die Library beginnen, Daten zu versenden und reagiert auf empfangene Commands.

```

...

// Define Config
Config options = new Config();
options.setMqttBrokerUri("tcp://iot.eclipse.org:1883");
options.setAppId("myApp");
options.setMqttCallback(new MyCallbackHandler());
MqttPublisher.setConfig(options);

// Create Device and Description
MqttDevice device = new MqttDevice("Bern", "Wankdorf", "Temp", "t1");
device.setDeviceDescription(new DeviceDescription("test", "1.0"));
...

// Start publishing
MqttPublisher.getInstance().publishDesc(device);
MqttPublisher.getInstance().pubEvent(device, "event", "22");
...

```

Listing 12: Einfaches Beispiel für die Verwendung der Library

Konfiguration

Über das Config Objekt wird die Library konfiguriert. Folgende Parameter können gesetzt werden:

Name	Beschreibung
mqttBrokerUri	Pflichtfeld. URI des MQTT Brokers (inkl. Protokoll und Port) Beispiel: tcp://iot.eclipse.org:1883
mqttClientIdPrefix	Prefix für die ClientIds der MQTT Connections. Es werden zwei Connections aufgebaut. Für das Publishing eine mit ClientId [mqttClientIdPrefix]_pub und eine Connection für das Empfangen von Messages mit ClientId [mqttClientIdPrefix]_sub Falls kein Wert angegeben wird, wird eine zufälliges Prefix generiert.
appId	Pflichtfeld. Identifikation der Applikation.
mqttCallback	Pflichtfeld. Implementation des Interfaces org.eclipse.paho.client.mqttv3.MqttCallback. Wird aufgerufen, wenn die Devices einen Command erhalten.
eventsRetained	true / false. Gibt an, ob die Event Messages mit Retained Flag versendet werden sollen. Standard: true
descriptionFormat	Format der Device Description. Mögliche Werte: YAML, JSON. Standard: YAML
eventsQoS	Gibt an, mit welcher QoS die Event Messages versendet werden sollen (0, 1, 2). Standard: 1
stateQoS	Gibt an, mit welcher QoS die State Messages versendet werden sollen (0, 1, 2). Standard: 1
descriptionQoS	Gibt an, mit welcher QoS die Description Messages versendet werden sollen (0, 1, 2). Standard: 1

Tabelle 7.2.: Konfigurationsoptionen der Library

7.1.3. Verwendete Komponenten

Für die Umsetzung der MQTT Device Description Library wurden folgende Komponenten integriert:

Komponente	Beschreibung
Eclipse Paho	MQTT Client Implementation. http://www.eclipse.org/paho
Jackson JSON Processor	JSON Serialisierung und Parsing http://wiki.fasterxml.com/JacksonHome
Jackson Modul YAML	Erweiterung der Jackson Library für das YAML Datenformat https://github.com/FasterXML/jackson-dataformat-yaml

Tabelle 7.3.: Externe Komponenten

7.2. Device Browser

Mit der entwickelten Webapplikation lassen sich die Device Descriptions übersichtlich darstellen. Die Applikation ist als reine Clientanwendung in Javascript umgesetzt. Die Anwendung zeigt die auf einem Broker publizierten Devices in einer Liste an. Zu einem Device kann der Benutzer sich die Device Description im Klartext anzeigen lassen. Die Device Description wird interpretiert und unterteilt in die verschiedenen Bereiche werden die Attribute angezeigt.

The screenshot displays the MQTT Device Browser interface. At the top, there are tabs for 'MQTT Device Description', 'Device Browser', and 'Topic Tree'. The 'Device Browser' tab is active, showing a table of devices:

Application	Group	Subgroup	Device Type	Device ID
thesis	X1-Carbon	tfstack2	Temperature IR Bricklet	qC1
thesis	X1-Carbon	tfstack2	Humidity Bricklet	qSG
thesis	X1-Carbon	localhost	Joystick Bricklet	hEc
thesis	X1-Carbon	localhost	Dual Button Bricklet	mxg

Below the table, the selected device's description is shown in a text area on the left. On the right, there is a detailed view of the device's state information, including sections for 'Info' and 'State'.

Info:

- Id:** IoT - Temperature IR Bricklet
- Version:** 0.0.1
- Description:** The Temperature IR Bricklet is equipped with an infrared thermometer. It can extend the features of a Brick with the capability of contactless temperature measurement. You can read out object temperature and ambient temperature in °C. It is possible to define the emissivity of the object you want to measure (most infrared thermometers can't do this). With configurable events it is possible to react on changing temperatures without polling.

State:

- AmbientTemperatureInterval:** The period in ms with which the ambient temperature is measured. A value of 0 means no measurement. Value: Type: Integer, Min: 0, Max: 9223372036854775807
- ObjectTemperatureInterval:** The period in ms with which the object temperature is measured. A value of 0 means no measurement. Value: Type: Integer, Min: 0, Max: 9223372036854775807

The 'Topic' field shows the path: `thesis/X1-Carbon/tfstack2/Temperature IR Bricklet/qC1/state/AmbientT`.

Abbildung 7.2.: Screenshot Device Browser

Bereich State

Die State Objekte werden mit Beschreibung und den Typangaben angezeigt. Zudem wird das Topic ausgegeben, auf welchem die entsprechenden State Informationen publiziert wurden.

Bereich Events

Events

ObjectTemp
Measured with IR sensor in Celsius
Value:

- Type: Double
- Min: -70
- Max: 380

Topicthesis/X1-Carbon/tfstack2/Temperature IR Bricklet/qC1/events/ObjectT

Unsubscribe20.7

AmbientTemp
Ambient temperature in Celsius
Value:

- Type: Double
- Min: -40
- Max: 125

Topicthesis/X1-Carbon/tfstack2/Temperature IR Bricklet/qC1/events/Ambien

Subscribe

Abbildung 7.3.: Device Browser Event Darstellung

Die Event Informationen werden mit Name, Beschreibung und Angaben zum Typ angezeigt. Pro Events wird das Topic ausgegeben. Mit dem betätigen der Schaltfläche 'Subscribe' verbindet sich die Webaplikation auf das Topic des Events und zeigt die empfangenen Daten im Textfeld an. Mit einem Erneuten Klick auf die Schaltfläche wird die Verbindung beendet.

Bereich Commands

The screenshot shows a 'Commands' section with a dropdown arrow. It contains two command entries:

setAmbientTemperatureCallbackPeriod
Linked state: AmbientTemperatureCallbackPeriod
Parameter: CallbackPeriod
Expects:
• Type: Long
• Min: 0
• Max: 9223372036854776000

Topic: thesis/X1-Carbon/tfstack2/Temperature IR Bricklet/qC1/commands/set

[Input field] [Send]

setObjectTemperatureCallbackPeriod
Linked state: ObjectTemperatureCallbackPeriod
Parameter: CallbackPeriod
Expects:
• Type: Long
• Min: 0
• Max: 9223372036854776000

Topic: thesis/X1-Carbon/tfstack2/Temperature IR Bricklet/qC1/commands/set

[Input field] [Send]

Abbildung 7.4.: Device Browser Command Darstellung

Für die Command Objekte der Device Description werden nebst Name, Beschreibung und den erwarteten Typinformationen auch das Topic angezeigt, auf welches der Command gesendet werden muss. Es steht ein Eingabefeld zur Verfügung, in welchem der Payload der MQTT Message eingegeben werden kann. Mit dem Betätigen der Schaltfläche 'Send' wird der Command an das selektierte Device gesendet.

Bereich Complex Types

The screenshot shows a 'Complex_Types' section with a dropdown arrow. It contains one complex type entry:

TemperatureCallbackThreshold
Properties:
• option: String
• min: Short
• max: Short

Abbildung 7.5.: Device Browser Complex Type Darstellung

Falls das Schema Complex Types beinhaltet, werden diese hier aufgelistet. Für jeden Complex Type wird der Name

und eine Liste der Properties (Attribute) aufgeführt.

Topic Tree

Als zweiter Teil der Webapplikation ist die Darstellung aller MQTT Topics des konfigurierten Brokers eingebunden. Dafür wurde die Applikation 'd3-MQTT-Topic-Tree' von Ben Hardill (<https://github.com/hardillb/d3-MQTT-Topic-Tree>) verwendet.

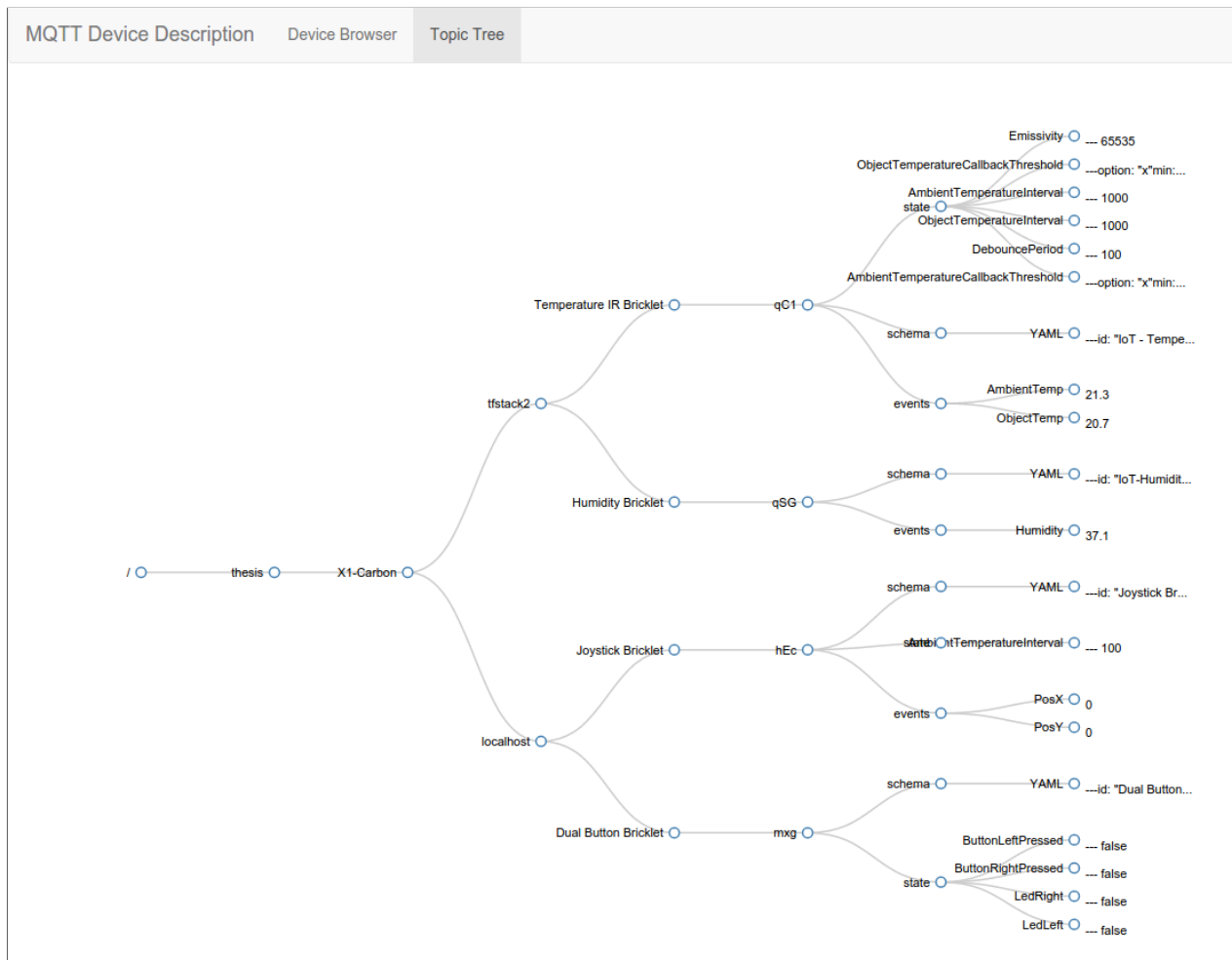


Abbildung 7.6.: MQTT Topic Tree

Vor allen während der Entwicklung und dem Einbinden von neuen Devices hat sich diese Darstellung als sehr hilfreich erwiesen.

7.2.1. Installation und Konfiguration

Der MQTT Broker, auf den die Webapplikation zugreift, muss per Websockets erreichbar sein. Dafür eignet sich zum Beispiel der Mosquitto Broker. Während der Entwicklung der Applikationen wurde eine Mosquitto Installation mithilfe eines vorbereiteten Docker Images (<https://hub.docker.com/r/toke/mosquitto/>) durchgeführt und für sämtliche Tests verwendet.

Um die Webapplikation verwenden zu können, muss lediglich der Inhalt des Verzeichnisses `ch.bfh.bti7321thesis.schemabrowser` auf einen Webserver kopiert und die Datei `js/app/config.js` angepasst werden.

```
host = '46.101.165.125'; // hostname or IP address of broker
port = 9001;             // Websocket Port of broker
descFormat = 'YAML';     // YAML or JSON
```

Listing 13: Beispiel Konfiguration Device Browser

7.3. Anwendung Tinkerforge

Um die MQTT Device Description Library an einem konkreten Beispiel zu testen, wurden verschiedene Bausteine des Tinkerforge Systems verwendet. Die Tinkerforge Bausteine (auch Bricklets genannt) haben den Vorteil, dass sie einfach zu verwenden sind und es bereits Libraries für die Kommunikation mit der Hardware gibt. Das Ziel der Anwendung war es, zu demonstrieren, dass es möglich ist konkrete Devices mit dem entwickelten Konzept resp. der Device Description Library zu beschreiben und eine Interaktion von ausserhalb per MQTT zu ermöglichen.

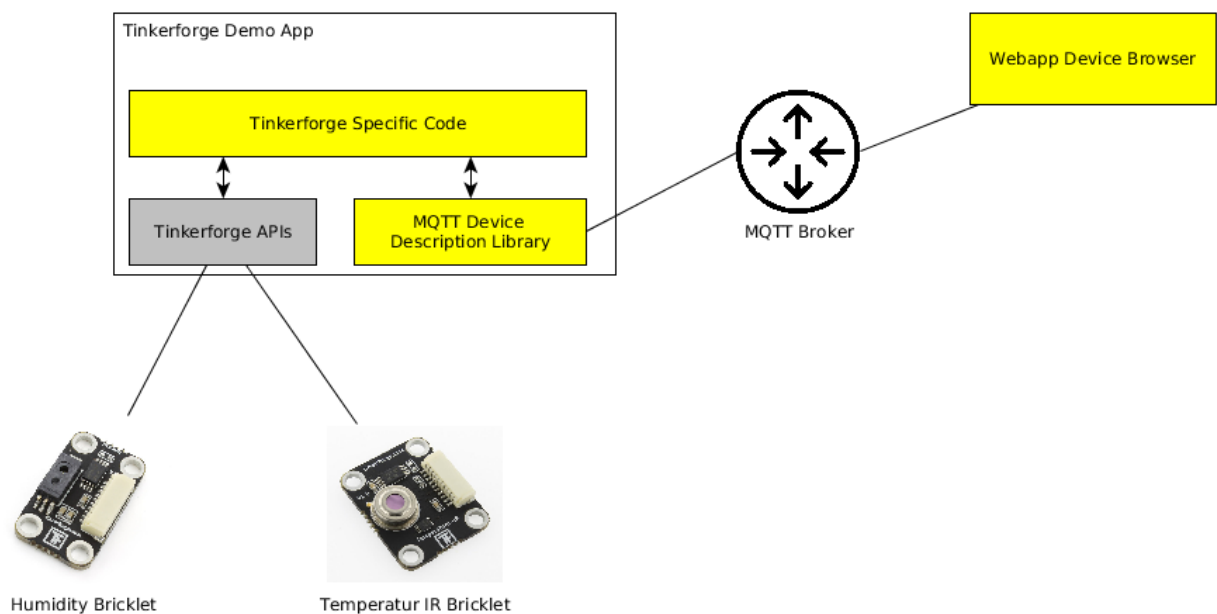


Abbildung 7.7.: Aufbau der Tinkerforge Anwendung

Folgende Sensoren resp. Aktoren wurden ausgewählt:

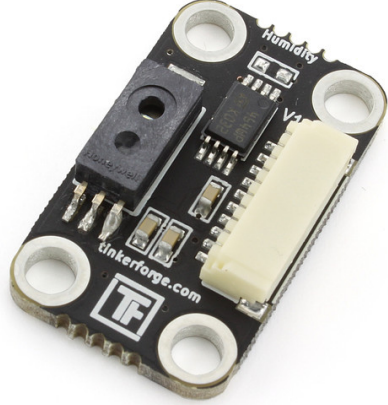

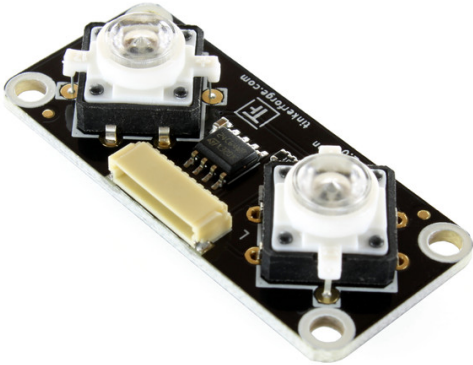
Komponente	Beschreibung	Bild
Sensor Luftfeuchtigkeit	<p>Misst die relative Luftfeuchtigkeit.</p> <p>Mehr Informationen: http://www.tinkerforge.com/en/doc/Hardware/Bricklets/Humidity.html </p>	 <p>A black rectangular circuit board with four mounting holes. It features a white 3-pin header on the right side and a small black sensor component on the left. The text 'Humidity' and 'tinkerforge.com' are visible on the board.</p>
Sensor Temperatur Infrarot	<p>Besteht aus einem Temperatursensor für die Umgebung und einem Infrarotsensor für die Messung der Objekttemperatur</p> <p>Mehr Informationen: http://www.tinkerforge.com/en/doc/Hardware/Bricklets/Temperature_IR.html </p>	 <p>A black rectangular circuit board with four mounting holes. It features a white 3-pin header on the right side and a prominent circular infrared sensor lens in the center. The text 'Temperature-IR' and 'tinkerforge.com' are visible on the board.</p>
Dual Button	<p>Besteht aus zwei Buttons, welche je ein LED integriert haben.</p> <p>Mehr Informationen: http://www.tinkerforge.com/en/doc/Hardware/Bricklets/Dual_Button.html </p>	 <p>A black rectangular circuit board with four mounting holes. It features two white push buttons, each with a clear LED lens, mounted on the board. A white 3-pin header is located on the left side. The text 'Dual Button' and 'tinkerforge.com' are visible on the board.</p>

Tabelle 7.4.: Verwendeten Tinkerforge Komponenten

Quelle Bilder: <http://www.tinkerforge.com/en/doc/index.html#hardware>

7.3.1. Klassendiagramm

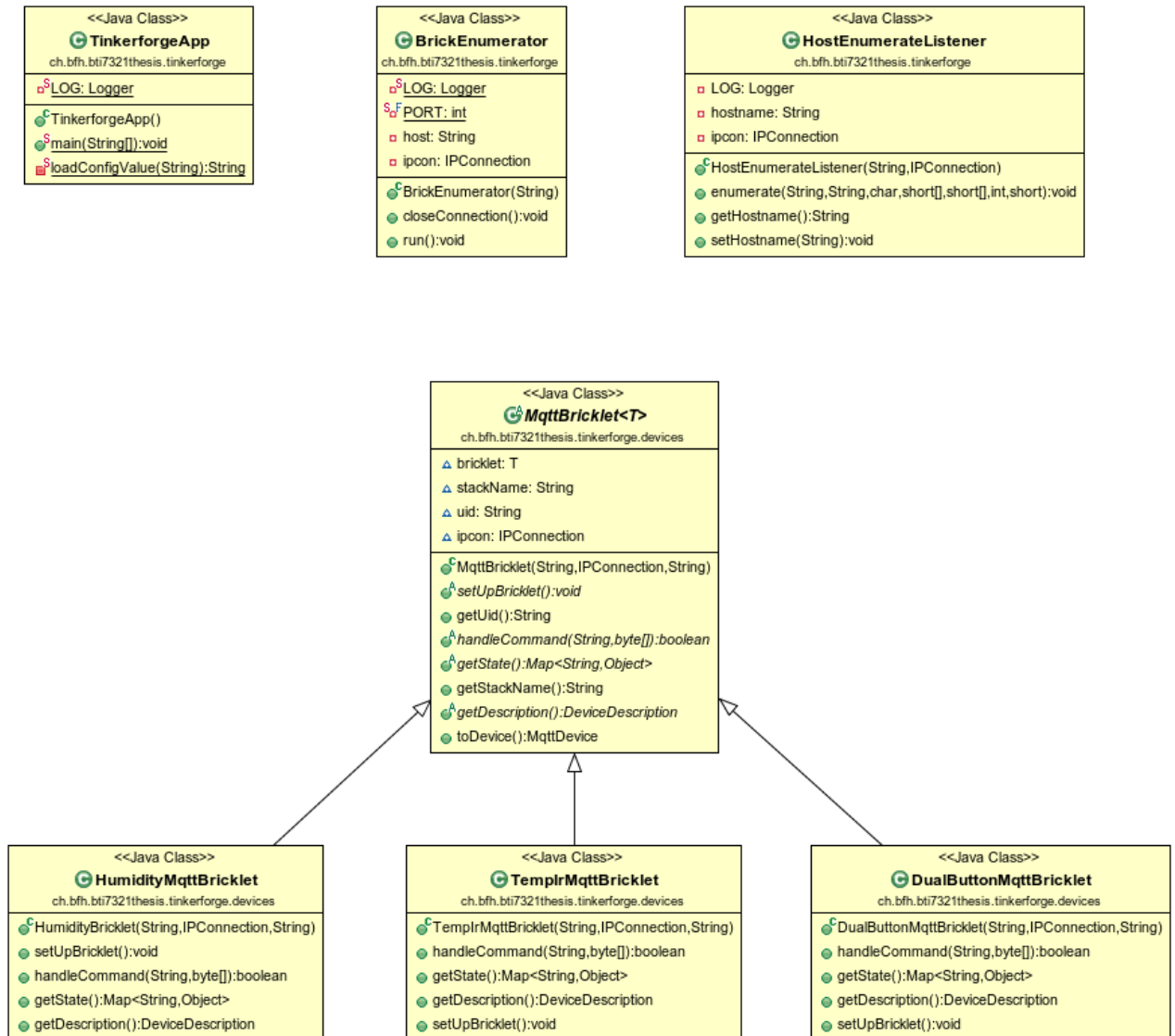


Abbildung 7.8.: Klassendiagramm Tinkerforge Anwendung

Klasse	Zweck
TinkerforgeApp	Start der Applikation, Einlesen der Konfiguration, Setup der MQTT Device Description Library.
BrickEnumerator	Thread, welcher für jeden angegebenen Tinkerforge Stack die Devices (Bricklets) sucht.
HostEnumerateListener	Epfängt die gefundenen Tinkerforge Devices und erstellt die entsprechenden Mqtt-Bricklets
MqttBricklet	Abstrakte Oberklasse für die konkreten Bricklet Implementationen. Definiert die abstrakten Methoden <code>setUpBricklet()</code> , <code>getState()</code> , <code>handleCommand(...)</code> , und <code>getDescription()</code> , welche von den Subklassen implementiert werden. Mit der Methode <code>toDevice()</code> wird eine Instant der Klasse <code>MqttDevice</code> erzeugt wird, welche die Subklassen verwenden um z. Bsp. Events zu versenden.
HumidityMqttBricklet	Implementation für das Humidity Bricklet.
TemplrMqttBricklet	Implementation für das Temperatur IR Bricklet.
DualButtonMqttBricklet	Implementation für das Dual Button Bricklet.

Tabelle 7.5.: Beschreibung der wichtigsten Klassen

7.3.2. Device Desvriptions

Die Device Descriptions der drei behandelten Devices sind im Anhang TODO zu finden.

7.4. Beurteilung Umsetzung

Darstellung Typen

Verlagerung Deserialisierung

Datenformate

8. Verifikation

8.1. Funktionale Anforderungen

Anhand der definierten Use Cases wird getestet, ob die funktionalen Anforderungen erfüllt sind.

Use Case	Bewertung	Ergebnis
UC01: Beschreibung Device	Entwickeltes Schema enthält die geforderten Angaben.	OK
UC02: Angabe Datentypen	Die Beschreibungsobjekte können mit Datentypen versehen werden.	OK
UC03: Definition Range	Die Datentypen können mit Ranges eingeschränkt werden.	OK
UC04: Definition Enum	Die Beschreibung unterstützt Datentypen mit fixen Auswahllisten	OK
UC05: Gruppierung Devices	Die Devices können auf verschiedenen Stufen gruppiert werden.	OK
UC06: Konfiguration Device Description Library	Die Library ist so aufgebaut, dass Konfigurationsparameter gesetzt werden können.	OK
UC07: Anzeige Devices	Devices werden in der Webapplikation angezeigt.	OK
UC08: Anzeige Device Description	Description eines Devices wird in Klartext und in interpretierter Form in der Webapplikation angezeigt.	OK
UC09: Anzeige Eventdaten eines Devices	Der Benutzer kann sich über die Webapplikation Eventdaten anzeigen lassen.	OK
UC10: Versenden eines Commands	Der Benutzer kann über die Webapplikation Commands erfassen und an die Devices senden.	OK

Tabelle 8.1.: Verifikation funktionale Anforderungen

8.2. Nichtfunktionale Anforderungen

Anforderung	Bewertung	Ergebnis
Erweiterbarkeit Devices	Mit der Modularisierung der Device Description Library ist die Abgrenzung zu den konkreten Umsetzungen sichergestellt.	OK
Erweiterbarkeit Datenformate	Durch den gewählten Aufbau der Topic Hierarchie ist es möglich, verschiedene Formate für die Device Description zu verwenden.	OK
Einfache Installation	Die Device Description Library kann als Maven Modul in bestehende Anwendungen integriert werden. Die Webapplikation kann ohne Abhängigkeiten installiert werden, die Konfiguration ist zentral definiert.	OK
Kompatibilität	Die entwickelte Lösung basiert auf dem bestehenden MQTT Protokoll und ist somit kompatibel für die Anbindung an beliebigen Applikationen auf Basis von MQTT.	OK

Tabelle 8.2.: Verifikation nichtfunktionale Anforderungen

9. Schlussfolgerungen/Fazit

Selbständigkeitserklärung

Ich bestätige mit meiner Unterschrift, dass ich meine vorliegende Bachelor-Thesis selbständig durchgeführt habe. Alle Informationsquellen (Fachliteratur, Besprechungen mit Fachleuten, usw.) und anderen Hilfsmittel, die wesentlich zu meiner Arbeit beigetragen haben, sind in meinem Arbeitsbericht im Anhang vollständig aufgeführt. Sämtliche Inhalte, die nicht von mir stammen, sind mit dem genauen Hinweis auf ihre Quelle gekennzeichnet .

Ort, Datum: Bern, 21.01.2016

Namen Vornamen: Bärtschi Adrian

Unterschriften:

Glossar

Base64 Verfahren zur Codierung von binäredaten in eine Zeichenfolge <https://tools.ietf.org/html/rfc3548>.

IEEE 754 IEEE Standard for Floating-Point Arithmetic <http://ieeexplore.ieee.org/servlet/opac?punumber=4610933>.

IoT Internet of Things.

JSON JavaScript Object Notation; Kompaktes, textbasiertes Datenformat <https://tools.ietf.org/html/rfc7159>.

MQTT Masdf.

QoS Quality of Service.

Serialisierung Strukturierte Daten (Objekte) in sequenzielle Form bringen zur Persistierung oder Netzwerkübertragung.

UID Unique Identifier.

UTF-8 Character Encoding für Unicode <https://tools.ietf.org/html/rfc3629>.

YAML YAML Ain't Markup Language; Gut lesbares, textbasiertes Datenformat <http://www.yaml.org/spec/1.2/spec.html>.

Literaturverzeichnis

- [1] Brick MQTT Proxy – Tinkerforge. Accessed: 2015-11-02. [Online]. Available: http://www.tinkerforge.com/en/doc/Software/Brick_MQTT_Proxy.html
- [2] Eclipse Paho Projekt. Accessed: 2015-11-15. [Online]. Available: <http://www.eclipse.org/paho/>
- [3] IBM Internet of Things Foundation. Accessed: 2015-11-02. [Online]. Available: <https://internetofthings.ibmcloud.com>
- [4] Message Payload — IBM IOT Foundation 1.0 documentation. Accessed: 2015-11-02. [Online]. Available: <https://docs.internetofthings.ibmcloud.com/messaging/payload.html>
- [5] MQTT Client Libraries. Accessed: 2015-11-01. [Online]. Available: <https://github.com/mqtt/mqtt.github.io/wiki/libraries>
- [6] MQTT Specification Version 3.1.1. Accessed: 2015-11-02. [Online]. Available: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>

Abbildungsverzeichnis

3.1. MQTT im (vereinfachten) OSI Stack	7
3.2. Publish/Subscribe Prinzip	8
4.1. Beispiel eines Gateways: Raspberry Pi 2	19
4.2. Systemübersicht	20
4.3. Aufbau MQTT Topics mit Beispieldaten	21
7.1. Klassendiagramm MQTT Device Description Library	34
7.2. Screenshot Device Browser	37
7.3. Device Browser Event Darstellung	38
7.4. Device Browser Command Darstellung	39
7.5. Device Browser Complex Type Darstellung	39
7.6. MQTT Topic Tree	40
7.7. Aufbau der Tinkerforge Anwendung	41
7.8. Klassendiagramm Tinkerforge Anwendung	43

Tabellenverzeichnis

2.1. Involvierte Personen und deren Aufgaben	3
2.2. Meilensteine der Thesis	3
2.3. Planung der Tasks und Auswertung der geleisteten Stunden	4
2.4. Termine und Fristen	5
3.1. Aufbau einer MQTT Message	9
4.1. Topic Hierarchie	21
5.1. UC01: Beschreibung Device	23
5.2. UC02: Angabe Datentypen	23
5.3. UC03: Definition Range	24
5.4. UC04: Definition Enum	24
5.5. UC05: Gruppierung Devices	24
5.6. UC06: Konfiguration Device Description Library	25
5.7. UC07: Anzeige Devices	25
5.8. UC08: Anzeige Device Description	25
5.9. UC09: Anzeige Eventdaten eines Devices	26
5.10. UC10: Versenden eines Commands	26
6.1. Primitive Datentypen	29
6.2. DeviceDescription Objekt Schema	30
6.3. StateDescription Objekt Schema	30
6.4. State Objekt Schema	30
6.5. EventDescription Objekt Schema	30
6.6. Event Objekt Schema	31
6.7. CommandDescription Objekt Schema	31
6.8. Command Objekt Schema	31
6.9. State Description Schema	31
6.10. State Description Schema	32
7.1. Beschreibung der Klassen	35
7.2. Konfigurationsoptionen der Library	36
7.3. Externe Komponenten	37
7.4. Verwendeten Tinkerforge Komponenten	42
7.5. Beschreibung der wichtigsten Klassen	44
8.1. Verifikation funktionale Anforderungen	45
8.2. Verifikation nichtfunktionale Anforderungen	45
A.1. Arbeitsjournal	59

A. Arbeitsjournal

Woche	Datum	Arbeiten
1	14.09.	Kickoff-Vorlesung BFH Bernhard Anrig Projektinitialisierung, Aufsetzen Dokumentation auf sharelatex.com Besprechung mit Reto König betreffend Ablauf, Organisatorisches, Ziele
2	21.09.	Projektplanung Dokumentation Projektmanagement, Dokumentation Einleitung
3	28.09.	Recherche bestehende Konzepte Einarbeitung Tinkerforge Bausteine Aufsetzen Prototyp Tinkerforge
4	05.10.	Einlesen bestehende Technologien (Coap, LWM2M, RESTful API Dokumentation) Besprechung mit Reto König, skizzieren des Prototyps
5	12.10.	Prototyp Tinkerforge Temperatursensor mit automatischer Enumeration/Erkennung, Versenden der Werte per MQTT Einbindung MQTT Topic Tree Webapp für bessere Übersicht
6	19.10.	Übernahme der Prototyp Ergebnisse in neues Projekt, Refactoring Server Setup DigitalOcean. Installation Docker, Mosquitto Broker und Apache Webserver Analyse IBM IoT Foundation
7	26.10.	Einbinden von Tinkerforge DualButton und Joystick Probleme mit Online Dokumentation auf sharelatex.com, Aufsetzen und Einrichtung lokale Latex Umgebung Umstrukturierung der Applikation Erzeugung Device Description in JSON Einführung einheitliches Logging
8	02.11.	Besprechung mit Federico Flueckiger (Einführung in Thema, Termin Verteidigung) Ansatz für generische Einbindung der Tinkerforge Bricklets mittels Reflection des Java APIs. Besprechung mit Reto König betreffend Termin Verteidigung, Poster, Book
9	09.11.	Generischer Reflection Ansatz verworfen, nicht praxistauglich Organisation Termin für Verteidigung Verfeinerung Device Description
10	16.11.	Dokumentation Konzept, Architektur Besprechung mit Reto König betreffend TODO
11	23.11.	Initialisierung Webapplikation für Device Description Anzeige
12	30.11.	Besprechung mit Reto König betreffend TODO
13	07.12.	Commands
14	14.12.	Poster für Finalday BFH Book Page
15	11.01.	Trennung Library und Tinkerforge-Demo Git Repo Reorg
16	18.01.	Commands

Tabelle A.1.: Arbeitsjournal