TU WIEN Informatics

Enhancing Contextual Understanding through Semantic Extraction of Information from Topic Names and Endpoints

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Software Engineering and Internet Computing

eingereicht von

Marija Mihajlova
Matrikelnummer 01528582

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Prof. Dr. Henderik Proper
Mitwirkung:

Wien, 31. September 2024

Marija Mihajlova	Henderik Proper

TU WIEN Informatics

Enhancing Contextual Data Through Extraction of Metadata from MQTT Topic Names

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Software Engineering and Internet Computing

by

Marija Mihajlova
Registration Number 01528582

to the Faculty of Informatics

at the TU Wien

Advisor: Prof. Dr. Henderik Proper
Assistance:

Vienna, September 31, 2024

Marija Mihajlova	Henderik Proper

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Marija Mihajlova

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, 31. September 2024

Marija Mihajlova

Danksagung

Ich möchte meinem Betreuer, Prof. Dr. Henderik Proper, für seine Betreuung, Unterstützung und sein Feedback in jeder Phase dieser Masterarbeit danken. Sein intensives Engagement für akademische Leistungen und seine sorgfältige Aufmerksamkeit für Details haben meine Arbeit stark beeinflusst.

Mein Dank geht auch an meine Eltern, meine Geschwister und meinen Partner, die mich während meines Masterstudiums in allen Belangen unterstützt und inspiriert haben.

Acknowledgements

I would like to thank my supervisor, Prof. Dr. Henderik Proper, for his guidance, encouragement, and feedback throughout each stage of this master's thesis. His intense dedication to academic achievement and meticulous attention to detail have greatly influenced my work.

Thank you to my parents, siblings, and partner, who have been a great source of support and inspiration, helping me in all aspects throughout my master's studies.

Kurzfassung

In den letzten Jahren hat das Internet of Things (IoT) stark gewachsen, was zu immer heterogeneren Umgebungen geführt hat, in denen Geräte über unterschiedliche Protokolle und Standards miteinander kommunizieren. Bei der Kommunikation zwischen diesen Standards können Informationen in unterschiedlichen Komponenten vorhanden sein. Einige Protokolle, wie das weit verbreitete MQTT, verwenden identifizierende Informationen wie Standort, Messtyp oder Einheit im Ressourcennamen. Diese Arbeit analysiert, inwieweit sich die semantische Interoperabilität und das kontextuelle Verständnis durch die systematische Extraktion von Metadaten aus MQTT-Topic-Namen und deren Einbindung in die Nachrichten-Payloads verbessern lassen.

Eine gründliche, multivokale Literaturrecherche wurde durchgeführt, um aktuelle Praktiken der Metadatenverarbeitung, der MQTT-Topic-Strukturierung und des Context-Aware Computing zu bewerten. Auf der Grundlage der gewonnenen Erkenntnisse wurde ein MDE-Ansatz angewandt, um ein Artefakt zu gestalten und umzusetzen, das Metadaten auf Topic-Ebene parsen und harmonisieren kann. Dieses Artefakt wurde als Proof of Concept in die Home Assistant Plattform integriert, um seine Verwendung in einer Smart-Home-Umgebung zu zeigen.

Diese Arbeit präsentiert ein strukturiertes Metadatenmodell zur Interpretation von Topic-Namen und bietet ein wiederverwendbares Framework zur Implementierung, das in verschiedenen IoT-Plattformen genutzt werden kann. Die Auswertung zeigt, dass die resultierende Nachricht mehr gebrauchsfertige Informationen für weitere Verarbeitung wie semantische Schlussfolgerungen enthält.

Abstract

Due to the Internet of Things' (IoT) rapid growth in recent years, the IoT environment has grown increasingly diverse, with devices utilizing a wide range of protocols and standards to interact. Information can be found in various components of communication between these standards, and certain protocols, like the popular MQTT, include identifying information in the resource name, like location, measurement type, or unit. This thesis examines how methodically extracting metadata from MQTT topic names and incorporating it into message payloads may improve semantic interoperability and contextual comprehension.

In order to evaluate current methods in context-aware computing, MQTT topic structure, and metadata processing, a thorough multivocal literature analysis was carried out. An artifact that can parse and harmonize topic-level metadata was designed and implemented using a Model-Driven Engineering (MDE) technique based on the information gathered. This artifact was incorporated into the Home Assistant platform as a proof of concept to show how applicable it is in a smart home setting.

This thesis offers a reusable implementation framework that may be used on a variety of IoT platforms, together with a structured metadata model for subject name interpretation. According to the evaluation, the final message has more information that is ready for use in subsequent processing steps, like semantic reasoning.

Contents

Kurzfassung **xi**

Abstract **xiii**

Contents **xv**

1 Introduction **1**
1.1 Status Quo . . . 2
1.2 Research Aim and Objectives . . . 9
1.3 Motivating Example . . . 11
1.4 Thesis Structure overview . . . 12

2 Theoretical Foundations **13**
2.1 Internet of Things (IoT) . . . 13
2.2 Message Queuing Telemetry Transport (MQTT) . . . 14
2.3 Metadata . . . 16
2.4 Modeling . . . 16
2.5 Context-aware Computing . . . 19

3 Research design and methodology **21**
3.1 Literature Review . . . 21
3.2 Model-Driven Engineering (MDE) . . . 24

4 Related Work **27**
4.1 White Literature Review Summary . . . 27
4.2 Grey Literature Review Summary . . . 30
4.3 Conclusions Towards Research Questions . . . 39

5 Artifacts **41**
5.1 Metadata Model . . . 41
5.2 Metadata extraction Algorithm . . . 44
5.3 Proof of concept . . . 48
5.4 Implementation . . . 56

6 Evaluation **59**
6.1 Data for Evaluation 59
6.2 Evaluation Results 60
6.3 Proof of concept 64

7 Conclusion **69**
7.1 Summary of the thesis 69
7.2 Future Work 70

List of Figures **73**

List of Algorithms **75**

Acronyms **77**

Bibliography **79**

A Metadata Collection Home Assistant **87**

B Metadata Home Assistant Test Model **89**
B.1 Initial basic metadata 89
B.2 Extension to the metadata 91
B.3 Home Assistant metadata collection code 93

CHAPTER 1

Introduction

The Internet of Things (IoT) concept emerged in 1999, introduced by computer scientist Kevin Ashton. He had an idea to add RFID chips to products and follow them through the production line [HKS17]. Since then, the field of IoT has taken off, so much so that in 2025 it is estimated to have between 30 and 75 billion connected IoT devices [CuRK23]. IoT devices support every technical revolution, including smart cities, smart factories, and smart homes. However, the rapid growth of IoT implementations has left little time for standards to be developed properly and implemented uniformly. Furthermore, this heterogeneity is a natural feature of the Internet, which enables anyone to join and participate, according to Montori et al. [MLJ+18]. Because anyone can connect and participate, data is gathered in a variety of methods, resulting in heterogeneous data. One of the biggest obstacles in the IoT space is merging devices that use disparate protocols and standards due to the intrinsic diversity of IoT devices [SSdCG24].

The interoperability problem can be addressed on several levels. Those are data, applications/services, middleware, networking, and device levels [APS19]. First, the goal of the data (semantic) level is to have a common understanding of the shared data. Then, the application/service level has the task of providing heterogeneous services that can be reused, regardless of the technology used by each service. Next, the networking level deals with object mobility and information routing. Next, the device layer facilitates the seamless integration of additional devices into the existing network of IoT devices. Finally, the middleware layer is responsible for providing seamless connections and data exchange. It should be developed to offer easy interconnection, the possibility for data analytics, context awareness, and resource management [GPP+18]. Finally, according to Noura et al. [NAG18], improving interoperability at the middleware level can prevent users from being locked into a specific vendor.

Shiina et al. [STC+23] state that as the number of devices increases, the sensor management problem becomes more challenging. According to them, the solution to this problem lies in utilizing the metadata of sensor devices (device-specific information). The

collected metadata facilitates more detailed sensor and data management, which enhances the reliability of sensor data [STC+23]. Hönle et al. define metadata as data about data. Montori et al. [MLJ+18] looked into publicly available data (like data collected by a smart city) and discovered that the ability to automatically process the measurement's metadata, including the measurement type (e.g., temperature), the unit of observation (e.g., Celsius), location, etc., is a basic prerequisite for re-purposing open IoT data.

As the authors of [SS24] argue, a reliable means of communication is a must to fulfill IoT's full potential. However, the authors also observe that IoT systems and devices were developed using different pre-existing protocols, resulting in numerous interoperability challenges. According to them, some of the protocols involved in causing this heterogeneity are MQTT, Constrained Application Protocol (CoAP), Transmission Control Protocol (TCP), and Modbus, which coexist in different IoT systems. Furthermore, Tantitharanukul et al. found in [TOH+16] that the MQTT protocol is one of the most widely used ones and is often used in various systems. Still, communication between these systems is also challenging. According to them, the difficulty arises from the fact that MQTT communication involves two components that must be aligned for successful communication: the topic and the message. The topic filters messages for each connected client, and the Message contains the data generated by the sender. They specify a topic that consists of one or more topic levels, each separated by a forward slash, e.g., `myhome/livingroom/temperature` or `austria/vienna/traffic`. Tantitharanuku et al. say that not having an MQTT topic naming standard makes it difficult to understand, subscribe to, and utilize data from different sources. In addition to this, multiple surveys [AZ19, AHDM22] say that some of the biggest challenges in the semantic IoT (one of the key requirements) are Data Quality, Standardization, and Reusability. However, some systems do not utilize the full amount of available data. For example, some messages hide part of their contextual data in the topic name, such as location (an example is the SparkplugB standard [Lin19]), to make it easier to filter. Leaving that information out of the message processing means that the system can lose vital contextual information regarding the message.

1.1 Status Quo

To provide a detailed picture of the field and to position the contribution of this thesis, a scoping review was carried out. The aim of the review was to identify and present existing research related to enhancing context-awareness in IoT environments using semantic structures in topic names, to explore the current state of research and uncover potential gaps. The literature review was executed using the TU CatalogPlus (that uses DBIS System [1]) and supported by the AI research tool Scite.ai [2]. Because the research topic intersects several sub-domains, research was done through 3 complementary perspectives. (i) Context-aware systems. As the goal of this thesis is to increase context awareness of IoT solutions - studies were examined to distil architectural patterns and data

[1] `https://dbis.ur.de/ALL/`
[2] `https://scite.ai`

representations that current IoT platforms employ to increase their contextual knowledge. (ii) Metadata processing. Work that leverages device -level metadata—such as room identifiers, geospatial coordinates, or functional roles—was reviewed to understand how such descriptors can be automatically extracted and reasoned with. (iii) Modelling in the field of IoT. Research on on systematic topic-naming schemes in publish/subscribe protocols as well as challenges in defining these schemes were assessed for its ability to formalize semantic annotations and promote interoperability across heterogeneous IoT systems.

1.1.1 Perspective 1: Context-aware Systems

The first research perspective is context-aware systems, which stems from the goal of making IoT systems more context-aware. This section is on existing literature in this field; the detailed definition of Context-aware computing is presented in Section 2.5. Due to the large number of existing sensors that produce raw data, it needs to be analyzed and interpreted to extract meaningful information, by utilizing its contextual information (for example, time and location). Context-aware computing helps in this regard by allowing the use of contextual information linked to the raw data collected from sensors, making interpretation easier. Among other responsibilities, applications must be able to react dynamically and respond to changing contexts. There are two significant problems regarding processing context in applications. Firstly, it's complicated to capture the context within an evolving environment. In addition to that, even if one succeeds at the first point, the second hurdle comes, which is to make sense of that context to adapt the behavior [DRD+14]. Therefore, it is crucial to focus on acquiring the context and providing a context interpretation service for ubiquitous environments.

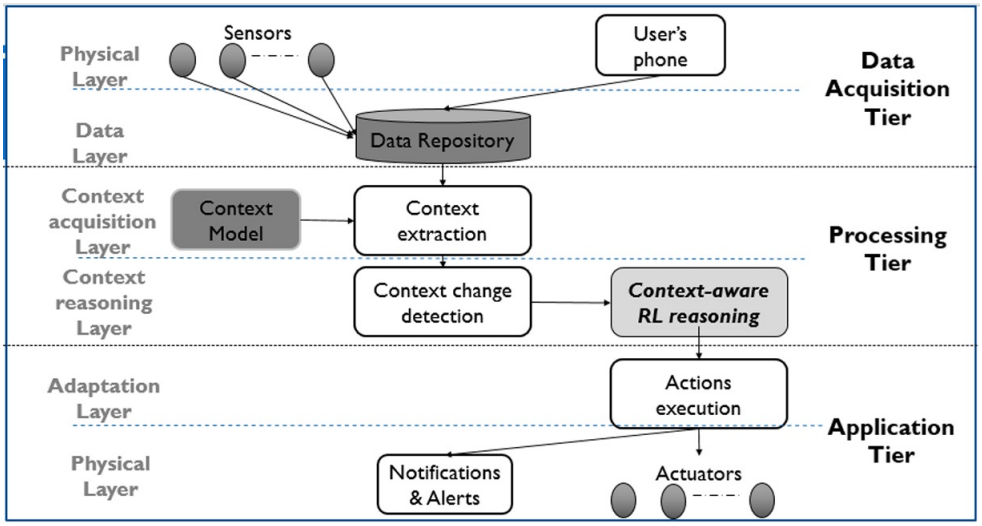

Figure 1.1: A multi-level approach to context extraction and reasoning, adopted from [HFHN24]

According to Almusaylim and Zaman [AZ19], the literature reviews on context-aware middleware have shown that the characteristics of devices and contexts, such as limited resources and dynamic contexts, are the primary causes of challenges in the development and implementation of context-aware middleware. They conclude that context-aware middleware solutions help reduce the time required for context-aware applications and decrease their complexity. They further state that context-aware middleware enables developers to focus on application development and automatically respond to various contexts without explicitly gathering contextual information from multiple sources. The literature has extensively examined context-aware systems, and researchers have attempted to address this issue through various approaches, middleware, and frameworks. However, most research focuses on reasoning with contextual information instead of collecting it. An example is [HFHN24].

The authors of [HFHN24] have investigated an approach to context-aware system development that utilizes metamodeling and reinforcement learning. In their research, they solve the problem in 3 stages. In the first stage, they focus on developing an architecture, which is the foundation of the system. In the second stage, they define a meta-model that conceptualizes the vision for the system. And finally, in the last stage, they focus on the reasoning process. The architecture they designed for their system consists of three layers: the Physical, Processing, and Application Layers. In the physical layer, the system connects to and gathers information from sensors in the physical world. The processing layer acquires and reasons with context, and finally, the application tier deals with adaptation, executing actions that result from the decision made during the reasoning process. A proposal of the architecture can be seen in Figure 1.1. In the second layer, they collect and process the message's context. To achieve this, they have proposed a metamodel for the context of the IoT system, as shown in Figure 1.2. The model contains the following elements:

- Thing – can be a physical thing (from the real world) which can have multiple devices attached to it, a virtual thing – a device which directly communicates with the system, or a device which can be a sensor, an actuator, or a combination of the two.
- Service – services are used to communicate or interact with the things of the system.
- Command – calls a service of a thing.
- Measure – All measures of the things can be used as contextual information of the user environment.

Finally, the context-aware reasoning component utilizes a reinforcement learning technique, specifically a Q-learning algorithm, which is an adaptive control algorithm essential for the dynamic environment of IoT.

Many researchers focus on rule-based reasoning techniques such as [WDD+22, PKPV21, DMM19]. However, these focus only on the reasoning and not on the context collection.

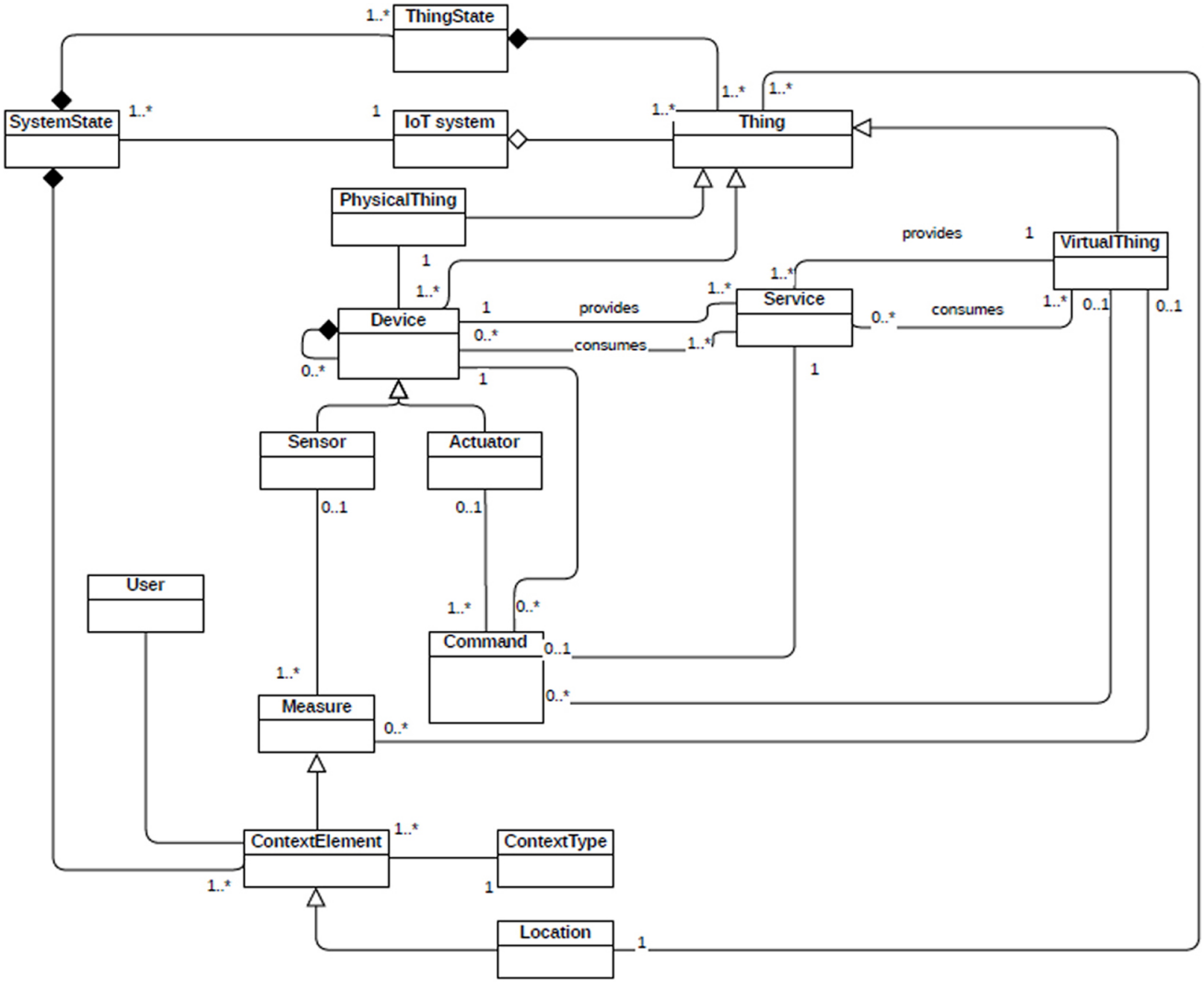

Figure 1.2: Meta model of the MDE Methodology, adopted from [HFHN24]

Anagnostopoulos focused his research in [Ana16] on the automatic collection of contextual information. However, he focuses more on the contextual data collected by sensors, at which point it is already processed, and the possible contextual information written in the topic name is lost.

1.1.2 Perspective 2: Metadata processing

The second research perspective is metadata, to gain insights into the possible data that can be contextualised. In the existing research, processing of metadata has been done in 2 ways. Some researchers have focused on the metadata of the data, such as age, author, and technical restrictions. Other researchers have focused on the data that is sensor-related, such as location, measurement, and unit.

Hönle et al. [HKN+05] have investigated the benefits of integrating the metadata into the context model. They found that incorporating metadata into the context model helps with finding resources, improves data selection, increases trust and data quality, and helps with sensor fusion (combining sensor values to derive higher context) [HKN+05]. They achieved this by using different types of metadata - metadata belonging to data providers, metadata belonging to data objects, and metadata belonging to attribute values. However, they only focus on the integration of the metadata within the contextual data and not on the collection of such metadata.

Montori et al. [MLJ+18] have focused on the open-access data that can be accessed from public platforms. They state that: "A fundamental requirement in successfully re-purposing such open IoT data, to enable interoperability as envisioned by Semantic Web 3.0, is to be able to automatically characterize its metadata i.e. information such as observation type (e.g. temperature, humidity), unit of observation (e.g. Celsius, Fahrenheit), location etc. [MLJ+18]". However, they also cite a study that says that the majority of publicly available IoT data lacks precise metadata, and even the type of observation is not certain. The authors then develop two methods to classify open IoT data streams and deal with the missing metadata. One of the approaches focuses on classification based on the data of the sensor readings, and the second one focuses on classifying the data stream based on the textual metadata describing the data stream [MLJ+18]. However, in this paper, the researchers focus on categorizing the data and not on incorporating this information into the message.

Shiina et al. recently investigated methods for metadata collection to enhance metadata enrichment, which, according to the authors, improves sensor management in platforms with a large number of devices [STC+23]. Their study specifically explored the utilization of layer-2 protocols for metadata collection to optimize resource efficiency in metadata transmission. While their findings indicate that the incorporation of metadata remains an open question, they do not account for the metadata that is already transmitted with the message.

Waterworth et al. investigated approaches for converting informal data into formal metadata in [WSS21], emphasizing the lack of formal metadata as a critical barrier to the

implementation of digital solutions in buildings. Their research focuses on the extraction of metadata from pre-existing labels stored as unstructured text fields. Additionally, Wang et al. investigated sensor metadata tagging in [WBK+18], they assessed the state of the field in 2018 and identified research gaps. In their research, Wang et al. highlight that manual metadata tagging is both labor-intensive and costly, which ultimately pose as significant challenges to the digitalization of buildings. Jiao et al. expanded this research and proposed a framework in [JLW+20] that tags sensors in new buildings based on insights gained from existing buildings, leveraging sensor names as a source of metadata. However, the authors found that as different buildings utilize sensors from various vendors, the naming conventions of these sensors often differ, complicating standardization. While existing research on metadata tagging primarily focuses on extracting information from stored data, this study aims to address real-time metadata retrieval at the moment a message is received.

1.1.3 Perspective 3: Modeling in IoT

Finally, modeling in the field of IoT is researched to investigate if topic naming/structuring has somehow been formalized. With the increase in the use of IoT, the quantity of data that is produced and transmitted is also increasing. Marina Harlamova et al. found in their survey on Semantic Applications in the IoT domain, that one of the biggest challenges in implementing IoT is Standardisation [HKS17]. The same findings regarding standardization were presented in the survey on semantic Web technologies by Rhayem et al. [RMG20]. The authors of both papers agree that standardization is crucial for the interoperability of devices, but having one standard that is used in every situation is highly unlikely. Therefore, they say that an adapter (or more) is needed to map between the different ontologies, which allows for the devices to be able to interwork, for the data center to be able to process this data at a higher level, and for easier resource management [HKS17]. Unifying implementations by modeling them is necessary to overcome challenges from missing standardization. Modeling in IoT has been done on two levels: modeling the IoT system and modeling the context of an IoT system. Schnellmann [Mar23] researched a modeling method to abstract IoT environments. As part of defining this modeling method, a few metamodels were created. Firstly, through a literature review, the author designed the metamodel, which can be seen in Figure 1.3.

In this metamodel, there are three types of entities: **Physical Entity** (a Physical object that can be found in the real world), Virtual Entity (a virtual representation of a Physical Entity), and **Augmented Entity** (Smart Object, which is the link between the physical and virtual entity). A **Device** is hardware in the real world, and as it is part of the physical world, it is also a Physical Entity. Furthermore, a Device must be connected to a **Network** to be able to communicate with the IoT System, and it can only communicate through specified **Communication Protocols**. In addition to this, the device has at least one endpoint open for communication. Furthermore, attributes of the Devices are modeled through the computational capabilities, physical capabilities, and device resources. Physical entities are connected to sensors that receive the stimuli, convert them

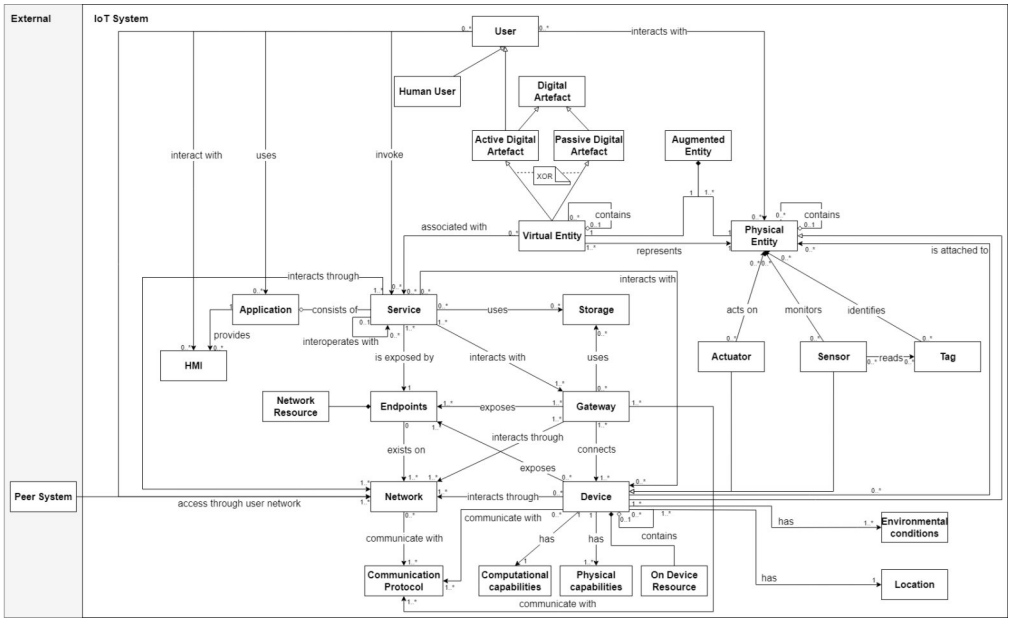

Figure 1.3: A literature-based IoT metamodel, as adopted from [Mar23]

.

into signals, and send them to the physical entity. Moreover, the physical entities can have Actuators, which are actors that can change their physical status, such as turning them off. And finally, the physical entities have tags that identify them and can be read from special devices. The environment of the entities is modeled through **Location**, which can be a building, room, or floor, and **Environmental conditions**, which take into account the conditions of the environment of the device, such as weather. Like any system, an IoT system has **Users** that interact with it. The metamodel differentiates between a **Human User** which needs a **Human Machine Interface (HMI)** to communicate with the system and a **Digital Artifacts** which are pieces of Software that interact with the system, such as Software Agents (Active) or Database entries (Passive).

Another research team [PKPV21] has focused on modeling context in the field of IoT. They developed a metamodel and context ontology that can be used among different implementations of IoT systems. They created a modeling approach that consists of Context Model for IoT (CM-IoT), Configuration of the model, and Context Ontology for IoT (CO-IoT), which is a generic ontology-based context organization. The CM-IoT, as shown in Figure 1.4, is an ontology-based model; it consists of the formal conceptualization of things and their categories, relationships, and taxonomy. Based on the 5 W's principle they have specified six dimensions of context: Entity (identity of the thing), Activity (task of the entity), Event (specification of why a task should be done at a specific location/time), Relationship (relationship between 2 context dimensions. Eg. depending on the location the temperature is different), Location (where is the thing located) and Time (when should the thing execute the task). For the context configuration, they

have designed a context manager that has three tasks – Context Cataloging, Context Fetching, and Context Dissemination. During the context cataloging task, the manager gathers all of the contextual information and pools that information at the associated registries. During an event, the application can fetch the needed context, and it will receive the context specific to that application. The context dissemination task provides the necessary context relevant to the end-users. Finally, to organize the context in an application, they provided a Context Ontology (CO-IoT). The ontology was created using the NeOn methodology, which is specialized to build ontologies based on the re-utilization of existing ontologies as well as the dynamic evolution of ontologies. The hierarchical structure of the ontology is divided into three layers: Upper-level (Core and Task) Ontology, Middle-level (Domain) Ontology, and Lower-level (Application) Ontology. The upper-level Ontology is the most generic one. It has 8 Context Concepts for a Thing - Event, Goal, Entity, State, Location, Time, and Profile, which are part of the Core ontology, and Activity, which is part of the Task Ontology. Entities are individual elements within the IoT system, they are separated into four categories - Living Entity (e.g. people or animals), Computational entities(e.g. Device or Network), Non-living entities (eg. temperature, water, building) and Virtual Entities (e.g. Documents, policies, units, etc.). Events are representations of specific occurrences happening at a particular time and place. A Goal is the state to which a system is supposed to get when an event is executed (the system's state at the end of the action). Location is used to describe the spatial context of a thing. Profile specifies the knowledge about an entity, which is information such as characteristics, factors, abilities, and roles. State ontology uses the state of an entity as part of the context of the application. Time represents the temporal context of an activity. Activity is part of the task ontology, and it represents an action of an entity. The Domain Ontology layer is a modular ontology that provides a lot of flexibility for the upper layer. It gives some generic terms, which are then connected to existing relevant ontologies such as SSN and OWL-Time. Finally, in the lower-level ontology application, specific concepts are provided on a case-by-case basis, depending on the application.

1.2 Research Aim and Objectives

This Thesis aims to enable IoT and data-as-a-service platforms to develop as context-aware systems. The context-awareness is done through the automatic processing of metadata stored in the resource (topic or endpoint) name on which the message is received, and then incorporating it into the message. This processing allows the system to reason semantically with the data.

To conceptualize how context-awareness might be achieved in IoT systems, a three-stage message processing can be proposed. The first stage would allow syntactic interoperability. It would enable technical connectivity and communication with all devices regardless of Application Layer Protocols, message structure, and ontology used by the device. This stage will be exempt from this thesis as there are existing research and tools that allow this. The second stage would facilitate the harmonization of the message with

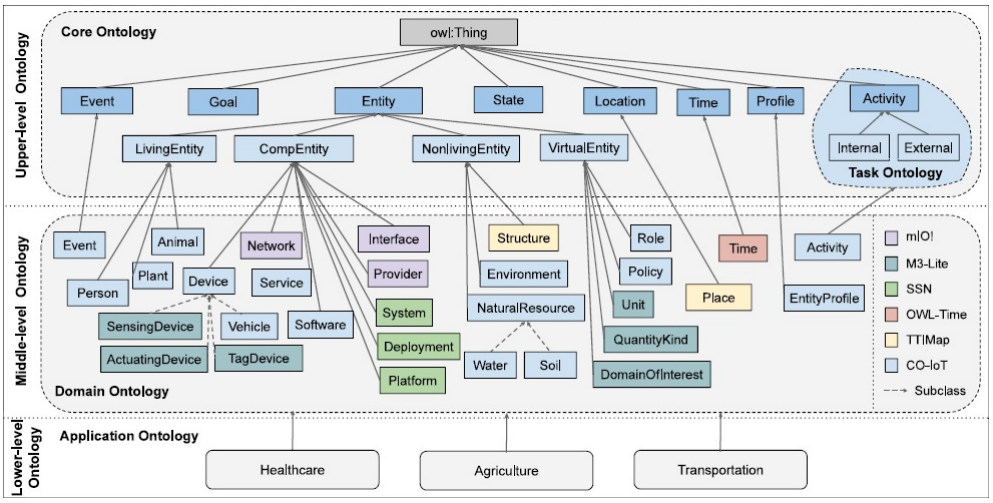

Figure 1.4: CM-IoT Ontology for context modeling as adopted from [PKPV21]

additional metadata into a unified message. At this stage, additional data necessary for the processing would be incorporated into the message and harmonized. Finally, in the last stage, semantic reasoning on the message with the metadata should be executed. As mentioned in 1.1.1, this has also been widely discussed and will not be part of the scope of this thesis.

Therefore, due to the missing research, this thesis focuses on the harvesting of metadata and contextual data present in the resource name and harmonizing it into the message. Due to the extensive implementation possibilities of IoT, the focus of this thesis is on a smart home (building) solution as a first proof of concept. The goal of this thesis is to create an artifact that will harvest metadata from the resource name, incorporate it into the message, and forward it for further processing. As a proof of concept, this artifact will then be incorporated into a smart home platform, however the goal is that it can be incorporated into different platforms. Finally, as communicating with many different ALPs is not part of this research, the proof of concept will be done only using the MQTT Protocol introduced in 2.2, and in the rest of the thesis, the resource name will be referenced as topic name.

Thus, the results of this thesis are twofold. Firstly, a model of the topic name processing will be specified (from existing research). Secondly, based on the initial stage, an artifact will be created to facilitate the processing of topic names. The research questions to support the results of the thesis are:

1. What kind of contextual information can be effectively encoded in MQTT topic names?
2. What are the key challenges in harmonizing extracted metadata across different MQTT topic naming conventions?

3. How can relevant metadata be accurately and efficiently extracted from MQTT topic names and then systematically structured and integrated into the message payload?

After the research questions have been defined, it is crucial to clarify what is meant with some limitations. Firstly, what is meant by effective encoding in the context of the first question needs to be clarified. Effectively encoded means contextual information that can be embedded in MQTT topic names in a semantically meaningful and technically feasible way. This contextual information must be human-understandable, sufficiently structured for machine parsing, and yet compact to conform to the topic name guidelines of MQTT. Second, the third research question requires further details regarding "accurate" and "efficient" extraction. To be accurate and ensure its‘ reliability, the artifact should achieve at least 95% accuracy during extraction of relevant metadata from MQTT topic name. To be efficient the processing time per topic name should not exceed 10 milliseconds, to enable integration in real-time environments of diverse IoT systems. Finally, the solution should be deployable as a plugin or module within existing IoT platforms—such as Home Assistant. This would demonstrate the artifacts usability and ease of integration in real-world applications.

1.3 Motivating Example

The motivating example is of Mr. and Mrs. Smith who live in a 2-story house. On the ground floor, they have a hallway, a living room, a kitchen, a bathroom, and a garden with a garage. On the first floor, there are three bedrooms and one bathroom. They installed a variety of sensors and actuators throughout their home, including humidity and temperature sensors. By putting up a context-aware system, they hope to fully utilize these devices and automate the maintenance of their house. Their goal is to have a personalized experience by having the system adapt to their needs and daily routines, as well as the needs of their home. The system should be able to adapt in the following cases:

- There are two temperature sensors in the living room. One near the window and one near the other wall. They both show different temperatures because cold is coming in through the windows. The system should be able to recognize that the temperature is the average.
- There are sensors in the kitchen and the hallway near the door. The system should know that in the hallway it will be colder due to the entrance door and that this space does not have to be warmed, since Mr. and Mrs. Smith use it only when coming in and going out.
- If motion sensors on the ground floor do not register anyone in the evening after 9, all lights on the ground floor should be turned off.

1.4 Thesis Structure overview

The rest of this Thesis is separated in Theoretical Foundations in Chapter 2, then the methodologies used in this thesis are presented in Chapter 3. In Chapter 4, the results of the literature review are presented, as well as conclusions to some of the Research Questions. Based on the knowledge gained from the Literature Review, the Artifact is developed and presented in Chapter 5, and evaluations of the artifact, as well as a proof of concept based on the chosen SmartHome Platform, are presented in Chapter 6. The conclusions based on the literature research, artifact development, and evaluations are presented in Chapter 7. Finally, the data used for the evaluations and data models are presented in the Appendix 7.2.

CHAPTER 2

Theoretical Foundations

In this chapter, the terminology and concepts mentioned throughout the paper are presented.

2.1 Internet of Things (IoT)

According to David Sembroiz et al. the Internet of Things "..is commonly defined as a network of physical and virtual objects, devices, or things that are capable of collecting surrounding data and exchanging it between them or through the Internet. To enable data collection, devices are embedded with sensors, software, and electronics; the exchange capability is achieved by connecting them to local area networks or the Internet. [SRC18]"

Implementations of IoT are possible in many different areas. Some of those are Smart Home, Smart Cities, Industrial IoT, and Industrial IoT. The evaluation method will be done on an example of Smart Home implementation; however, the solution should be implementable in any implementation of IoT

2.1.1 Communication Protocols

Standardized protocols are necessary for communication between devices. Since the beginning of IoT, many communication protocols have been developed. Some of those are presented in this section.

Wireless Infrastructure Protocols

The authors of [SRC18] say that Wireless Infrastructure protocols actuate inside the underlying infrastructure and allow communication between the system layers, for instance, the communication between the perception layer and the network layer. They further say that the IoT devices are often limited by their computing capabilities and/or energy

consumption. Because of that, different protocols were developed, some of which are ZigBee, Bluetooth Low Energy (BLE), 6LoWPAN, etc.. As the connection using different Wireless Infrastructure Protocols is already set up by SmartHome platforms, these protocols will not be considered in detail for this thesis.

Application Layer Protocols (ALP)

An Application Layer Protocols (ALP) facilitates the connection of the infrastructure to the application. The authors of [GIC20] say that in IoT applications ALPs usually refer to the lower part of the TCP/IP stack model's application layer and that those protocols are typically referred to as messaging protocols. They further specify that the protocols usually follow either the request/response model or the publish/subscribe model. As the field of IoT developed, so did different ALPs; some of the protocols which the authors presented in their paper are:

- Message Queuing Telemetry Transport (MQTT) is a client-server publish-subscribe protocol, it's advantages are that it is simple, lightweight, and very easy to implement [GIC20]. More details on this protocol are in 2.2.
- Hypertext Transfer Protocol (HTTP) is a request-response protocol with a client-server model. It is the foundation of the World Wide Web. Its use has been extended to the IoT field, however, it is not always the best choice due to its overhead and requirements.
- Constrained Application Protocol (CoAP) is a specialized protocol explicitly tailored for the constrained nodes and networks of IoT. It was standardized by the IETF Constrained RESTful Environments (CoRe) Working Group. It is an extension of HTTP and is based on the REST model; however, it also supports publish-subscribe communication.

2.1.2 Smart Home

Smart Home is also known as an automated home or an intelligent building. This implementation incorporates IoT devices that control home features. In the beginning, only environmental systems such as lighting and heating could be controlled; however, the technology has evolved such that almost any electrical component can be controlled [RMD+06]. According to Ricquebourg "*Smart homes are now dedicated to simplify the life of its inhabitants, to make energy saving, to provide comfort and security solutions. [RMD+06]*"

2.2 Message Queuing Telemetry Transport (MQTT)

Message Queuing Telemetry Transport (MQTT) is a lightweight client-server publish-subscribe messaging transport protocol [GIC20]. Due to its lightness, it is ideal for

constrained communication environments such as IoT [GIC20]. It can be used on any network protocol that allows ordered, lossless, and bi-directional communication [GIC20].

The three Quality of Service (QoS) levels make sure to keep the reliability of messages [OAS19]:

- At most once: In this scenario, messages are delivered at most once. The protocol tries to send it with a best effort; however, message loss can occur. It is tailored for scenarios where losing messages is not critical.
- At least once: The message is sent at least once. This quality of service means that a message can be sent multiple times if the sender is not sure that the receiver received it.
- Exactly once: This is the strictest QoS. Messages arrive once. This QoS is used only by critical systems that must have reliable operation. An example of such a system is a banking system

Because MQTT uses a publish/subscribe model, it also supports one-to-many message distribution. This distribution allows developers to create application decoupling.

MQTT Topic Name Requirements

MQTT is a lightweight protocol, allowing users as much flexibility as possible. However, even with this goal in mind, some technological requirements necessitate a structure for naming the topics. According to the specifications of MQTT 5 [OAS19], topic names are UTF-8 encoded strings and they must not contain the ‘null‘ character. In addition, the topic's size must be at least one character long and at most 65.535 bytes long. Furthermore, according to the specifications, topic names are case-sensitive, meaning that “home/livingroom” and “home/livingRoom” are two different topics. Another possibility for topic names is that they include a space; due to this “home/living room” is also an option when creating the topic names. Moreover, a character that is crucial for the structure of the topic names is ‘/‘; this character separates the topic into levels, which can be used later for filtering. More special characters which are used by MQTT are ‘$‘, ‘+‘, and ‘#‘, where ‘+‘ and ‘#‘ are wildcard characters and are used only for filtering; they are not allowed to be in the Topic Names. The character ‘$‘ should also not be used or used limited, as topics starting with ‘$‘ are reserved for server-specific topics, some widely accepted prefixes are “#SYS”, for example “#SYS/monitor”.

As the basic definition of the MQTT standard allowed users to be as flexible as they could, the topic structure is just as flexibly defined. However, research has shown that having some structure of the topics (topic tree) is crucial because it provides better communication, organization of the data and messages, can help with enhancing security measures, as well as help with insightful analytics [MV24]. To get the most out of the topic structure, researchers have tried to find the best structure. The structure can

be designed in two dimensions. First, you have to choose the correct name for the topics semantically, and secondly, the proper structure of the tree must be used. For the semantic processing of the data to create correct topics, methods such as latent semantic analysis (LSA), LDA, and hLDA can be used. They are natural language processing techniques and provide a generalized topic and categories into which it can be split to organize the data. As for the hierarchical structure of the data, multiple structures have been researched. For example, for geolocation-based topic structure, some of the topic structures that were researched are:

- Hierarchical structure based on location. It can be within the world, but most often it is used within an implementation. For example: "Austria/Vienna/1010/temperature" or "home/livingRoom/temperature"
- R-Tree Index Structure
- AP Tree

The flexibility of MQTT is what brought so many users to use it over other protocols. However, as different services started to be used together, the users and designers of the systems realized that some structure is still required. Due to that, some further specifications, such as [Lin19], were developed; however, this is presented in Chapter 4.

2.3 Metadata

In short, metadata is the data about data [HKN+05]. It gives us more information about the received data, such as its quality, age, precision, author, etc. According to Hönle et al. [HKN+05], several attributes of the data, such as Reliability, Precision, Consistency, Age, and Access Control, can be derived from correctly processed metadata.

In a newer study [STC+23], Shiina et al. focus on the metadata of the sensors as they process the data received from them. They call the metadata associated with the devices device-specific information. According to them, this metadata can be used to improve sensor data management and, with that, improve the reliability of the received data.

2.4 Modeling

Innovative approaches to software development are required in newly developing high-tech systems, such as Smart-home systems [GT23]. These approaches reduce the time and other resources needed for the development of these projects. To achieve them, one of the methods used is model-driven engineering [GT23]. While the details of this method are defined later in 3.2, in this section only the theoretical concepts needed for the method are presented.

Models

Models are an abstraction of a system. They are a simplified version of a system that points out the outstanding features of the system. After creation, the models are used for the presentation and evaluation of the system. According to Ludewig [Lud03], three main criteria necessary to make a distinction between models and other artifacts are:

- Mapping criterion - the model maps an original object or phenomenon.
- Reduction criterion - the model is a reduced version of the original object. This reduction means the model maps some of the original object's properties, but does not have to be all.
- Pragmatic criterion - Given a purpose, the model can replace the original object. It is helpful in some cases, but it does not have to be all.

Models can be classified in two ways - Abstraction hierarchy or dynamic behavior of the system [TDOY16].
Abstraction hierarchy:

- Highest level: On this level, models are very abstract. They explain the problem domain, however they do not reference the implementation-oriented and application-specific concepts [TDOY16].
- High level: On this level, the application-specific needs are modeled. The needed structure and behavior are specified in terms of coordination models and algorithms. Even at this level, no technical details are specified [TDOY16].
- Concrete-level: On this level, based on the provided technological and platform-specific constraints and aspects, abstract solutions are mapped to concrete solutions [TDOY16].

Dynamic behavior:

- Static models: The system's layout and organizational structure are specified by these models. They also describe the system's interactions, characteristics, and concepts [TDOY16].
- Dynamic models: Focus on the behavioral specifications. Such as a sequence of actions (activity-flow). State transitions and interaction dynamics which depict how the components of the modeled systems cooperate and coordinate with each other [TDOY16].

Meta Model

A Meta Model is a model of models; it is used to describe and specify a model [EGH24]. When defining a meta-model, a problem can arise, such that the language in which the meta-model is written has to be specified as part of the meta-model's specification. An example of such language is Lisp, which has a Lisp written compiler. To deal with this approach, OMG has specified a four-layered architecture – Meta Object Facility (MOF) [EGH24]. According to [EGH24], the top-most layer, M3, is a meta-modeling layer that is responsible for specifying a language which can be used to specify meta-models. The next layer, M2, is also a meta-modeling layer, but it is responsible for the instantiation of the meta-models. Domain-specific layer M1 is the third layer. It builds the model according to a system, its users, and its needs. The instantiation of the original object's components is represented by the last layer, M0.

2.4.1 Modeling Languages

According to Rodrigues da Silva [Rod15], a modeling language is "a set of all possible models that are conformant with the modeling language's abstract syntax, represented by one or more concrete syntaxes, and that satisfy a given semantics". The authors of [EGH24] say that a modeling language is specified through abstract syntax, concrete syntax, semantics, and pragmatics. They state that abstract Syntax is specified by cataloging and listing all concepts, abstractions, and relationships of the goal domain, through using subject-matter expertise. Resulting in a metamodel at the meta-domain level. The authors say that some of the most used mechanisms to define modeling languages are the UML profile mechanism and Meta Object Facility. As part of the Abstract Syntax, Structural Semantics is also defined, which provides how two or more elements relate to each other. [EGH24] specify also the concrete Syntax which allows the user to interact with the modeling language through using graphics, texts, tables, and forms. Finally, they define what they mean by semantics, which is the definition of the meaning of the legitimate expressions in a language. They can be executable and non-executable. Executable semantics defines how the concepts should be executed, and non-executable semantics defines the meaning of the other concepts, which can be related to requirements specifications and other deployment concepts. Correctly interpreting a language according to its context is the main goal of pragmatics [EGH24].

Domain-specific Modeling Languages (DsML) and General-purpose Modeling Languages (GpML) are the two categories of languages utilized in Model Driven Engineering. Domain-specific programming languages and modeling languages are examples of modeling languages designed to build models in a particular application domain. These languages include MATLAB, VDHL, Eclipse Modeling Framework, and others. On the other hand, GpMLs—such as the Unified Modeling Language (UML)—are used to create models that are applicable to a wide range of fields.

Domain Specific Language (DSL)

Fowler and Parsons in “Domain-Specific Languages” say that it is hard to define a DSL clearly, but that they would define it as: "Domain-specific Language(noun): a computer programming language of limited expressiveness focused on a particular domain" [FP11]. Furthermore, they elaborate that just like any other programming language, it should be created so that it is easy for humans to read but also possible for computers to execute. Moreover, they say that ‘limited expressiveness’ means that the DSL should not allow lots of capabilities like a general-purpose programming language. Instead, it should focus on its domain and provide functions only for the necessary actions. And finally, they note the importance of a clear domain focus [FP11].

We can differentiate between the two types of DSLs in any project - external and internal DSL. An external DSL, is different from the primary language of the project, with custom syntax. Some common external DSLs are SQL and XML [FP11]. An internal DSL is based on a general-purpose programming language; however, only a subset of the language’s features are used with the goal to handle a specific part of the system. The result of this implementation leaves the developer with a feeling that they are using a different language. Some general-purpose programming languages that allow the creation of internal DSLs are Lisp and Ruby (Rails Framework) [FP11].

2.5 Context-aware Computing

Context-aware computing is a style of computing in which a system can capture the environment state and environment state changes and adapt its function accordingly. They aim at increasing usability and effectiveness by being able to adapt by taking into account the current context without needing explicit user interactions. This adaptability, especially in IoT, makes the systems more interesting to the users [HFHN24]. According to [dMAT+15], context-aware computing is a key enabler and a vital research prospect in the IoT field.

In September 1991, the term “Ubiquitous Computing” was proposed and defined as “Ubiquitous computing is the method of enhancing computer use by making many computers available throughout the physical environment, but making them effectively invisible to the user [Wei99].”

2.5.1 Context

According to Dey, humans are quite successful at communication, and the reason for that is that they can understand the implicit situational information (context). Understanding context is something that computers cannot do implicitly, and it has to be specified. [Dey01] A widely used definition of context defined by Dey is:
“Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves. [Dey01]”

Another more recent definition is the one by Almazan: "Context consists of one or more relationships an information item has to another information item. An information item can be any entity, either physical (like a person, a computer, an object), virtual (like a computer service, a message), or a concept (location, time, and so on). A relationship describes a predicate connecting two or more information items, which may change at any time for any reason [Alm10]".

Context life cycle

A life cycle of data shows how the data moves in a software system, explaining to us where the data is generated and consumed. In terms of context, it means it follows the movement of context within context-aware systems [PZCG14]. According to Perera et al. [PZCG14], there are four stages to the life cycle of context information. Those are: acquisition, modeling, reasoning, and distribution. The acquisition of the context can happen from different sources (physical or virtual devices). Context modeling is explained in more detail in 2.5.2. During the context reasoning phase, the context-aware system deduces new knowledge based on the context collected in the previous stage. And finally, during distribution, it delivers context to consumers through querying or through a publish/subscribe pattern.

2.5.2 Context Modeling

An important step in designing context-aware systems is Context Modeling. According to [BBH+10], it is often also referred to as Context Representation. They further say that context models can be designed either as static or dynamic models. The static models work with a predefined set of contextual information, while the dynamic ones can work with changing contextual information. Furthermore, the authors of [BBH+10] state that the requirements which should be taken into consideration are heterogeneity and mobility, relationships and dependencies, timeliness (freshness), imperfection, reasoning, the usability of modeling formalisms, and efficient context provisioning.

The context modeling process can be separated into two stages:

- Context modeling process: new context is modeled where information is defined as attributes, characteristics, and relationships with existing context
- Organize context according to the model: Validate the result of the previous step and merge the new information into the existing context

As Perera et al. summarized in their paper [PZCG14], some of the most widely used methods for context-modeling are:key-value, markup schemes, graphical, object-based, logic-based, and ontology-based modeling.

CHAPTER 3

Research design and methodology

During the research and implementation for this thesis, two methodologies were used. In the first stage, a literature review is conducted to find the current status in the existing literature, and to support the research and implementation in the Model-Driven Engineering approach. In the second stage, the Model-Driven Engineering approach is used to develop the artifact.

3.1 Literature Review

A multivocal literature review approach was used to ensure a comprehensive and structured literature review. To improve efficiency and provide a thorough exploration of the literature, the AI-powered research tool Scite.ai[1] was utilized for initial general research of the topic, and for fine-tuning the research queries for the systematic literature review.

3.1.1 Scite.ai

Scite.ai is an AI-based system designed to support researchers by helping them find and understand the research papers they are looking for. In this thesis, Scite.ai, specifically the Assistant, was used to help refine the research queries, as well as find a common topic in a large volume of literature and suggest additional research papers to read. Initially, Scite.ai Assistant was used to grasp a general knowledge and extent of the research on the topic. During this phase, the queries were formed in the following structure "Can you do a literature review on the topic of" and the topic was specified for different viewpoints on this thesis. Some of those are:

- Contextual Data (and later specified "in the field of IoT")

[1] https://scite.ai

- Context Modeling in the field of IoT
- Model Driven Engineering in the field of IoT
- DSL use in IoT, specifically Home Automation solutions
- Semantic Models used in IoT, specifically Home Automation solutions
- Collection of Contextual Data in IoT
- Metadata extraction in IoT

This tool helped specify the research queries as from some of the sources it could be seen that some authors use "modeling" and others use "modelling", and further, it helped in understanding that in research papers, contextual data harvesting is usually done from the stored data and refocused the research towards metadata harvesting.

3.1.2 Multivocal Literature Review

Multivocal Literature Review is an extension of the Systematic Literature Review; it does not include only formally published (peer-reviewed) papers but also gray literature such as blog posts, videos, and white papers [ASH16]. The authors of [ASH16] define MLR as:

"*Multivocal literatures are comprised of all accessible writings on a common, often contemporary topic. The writings embody the views or voices of diverse sets of authors (academics, practitioners, journalists, policy centers, state offices of education, local school districts, independent research and development firms, and others). The writings appear in a variety of forms. They reflect different purposes, perspectives, and information bases. They address different aspects of the topic and incorporate different research or non-research logics*"

The authors consider the Multivocal Literature Review to have 3 phases. In the first phase, "Planning the Review", the researcher should identify the need, then specify the goal and define the research questions of the MLR. Next, during the "Conducting the Review" stage, the researcher should identify the research, choose the primary studies, and evaluate their the quality. To conclude the second stage, the researcher should extract and monitor the data as well as synthesize the data. In the last stage of the review "Reporting the Review" the researcher should specify the dissemination mechanisms, format, and evaluate the main report.

According to them, the literature that can be used is white (formally published) literature as well as gray literature. They define gray literature in 3 tiers:

- 1st tier. These sources have high outlet control and/or high credibility. They can be: books, magazines, government reports, white papers

- 2nd tier. These sources have moderate outlet control and/or credibility. They can be: Annual reports, news articles, presentations, videos, Q/A sites (such as StackOverflow), Wiki articles
- 3rd tier. These sources have low outlet control and/or credibility. They can be: Blogs, emails, tweets

The goal of the Multivocal Literature Review was twofold. Firstly, to answer the first and second Research Question. Secondly, to find out if any similar research on extracting metadata has already been done. As many MQTT standards have been developed by the industry and IoT enthusiasts, a large body of information lies in the gray literature; it is necessary to include the gray literature in this thesis. In addition to this, there is already research done in the white literature on extracting metadata and using contextual data, which leads to the necessity to also look into the white literature. Due to this, it is necessary to implement the Multivocal Literature Review in this thesis.

During the second stage of the research process, data collection was conducted through two distinct sources. For scholarly, peer-reviewed literature (white literature), a systematic query was executed using the TU CatalogPlus, which uses DBIS System [2] searches multiple databases such as Scopus, IEEE/IET Electronic Library (IEL), ACM Digital Library, Springer Nature Link, etc. For non-traditional, non-peer-reviewed sources (gray literature), the query was performed using Google. The query used in both cases is:

Search query:

(“contextual data” OR “context data” OR “metadata” OR “meta-data” OR “structure” OR “standards”) AND ("MQTT” OR “Message-Queuing Telemetry Transport") AND (“topic name” OR “topic tree” OR “topic naming”)

White Literature

The following filters were applied after the initial search to ensure that the reviewed literature is relevant and high-quality:

- Restrict to English-language publications
- Show only peer-reviewed results
- Limit to publication date of last 15 years (2010-2025)

After these constraints were applied, the result of the query contained 151 relevant sources.

[2] https://dbis.ur.de/ALL/

The results were further filtered such that the results did not contain duplicate publications, and the title and abstract were evaluated for relevance. After this filter was used, the result was 39 publications. After reading thoroughly all of the 39 publications, only eight sources remained. The papers and their conclusions are presented in chapter 4

Grey Literature

To maintain the research's feasibility and prevent the dataset from expanding indefinitely, the results retrieved from the Google query were restricted to the first 100 entries. During the initial data filtering process, duplicate links and redundant information—such as multiple blog posts providing a general overview of MQTT topic name structures—were removed. Additionally, the dataset was refined based on the title's relevance and the brief descriptions provided in Google search results. Following this filtering process, 44 links remained. After reading them thoroughly, only 22 sources remained. The sources and their conclusions are presented in chapter 4

3.2 Model-Driven Engineering (MDE)

Building complex applications, such as those in smart environments, especially ones beyond the send-condition-act processes, must be done after identifying the main barriers or requirements. Developing these applications in the IoT field is a difficult task because of the increasing amount, and heterogeneity of IoT devices, platforms, and frameworks [PDTS24]. One of the methods that tries to solve these issues is the Model-Driven Engineering (MDE)) process. In this section, the MDE process and concepts, along with different modeling levels and operations, are presented. According to the authors of [TDOY16] MDE differentiates from other development processes, such as object-oriented or agent-oriented, in the way that it uses models and relations instead of objects or agents as the building blocks of the development process. They further say that within a system, multiple models can co-exist on different levels or different aspects of the system. They are the building blocks of the system. By heavily relying on the use of models and model engineering, the MDE process can systematically produce and transform models. Models and model engineering are used extensively in the process, which enables the principled and methodical creation and modification of models [TDOY16].

3.2.1 Model-Driven Engineering Methodology

"MDE approaches idolise models as the development centre in an application domain for any given software project. The MDD approach focuses on the disciplines of requirements, analysis, design, and implementation. It defines or employs modelling languages for the specification of the System Under Study at various levels of abstraction, resulting in M2M and M2T transformations that contribute to the successful improvement of the software system. MBT is restricted to the automation of the testing of the software system, where some testing models are created that represent some desirable behaviour of the System Under Test (SUT)." [EGH24]

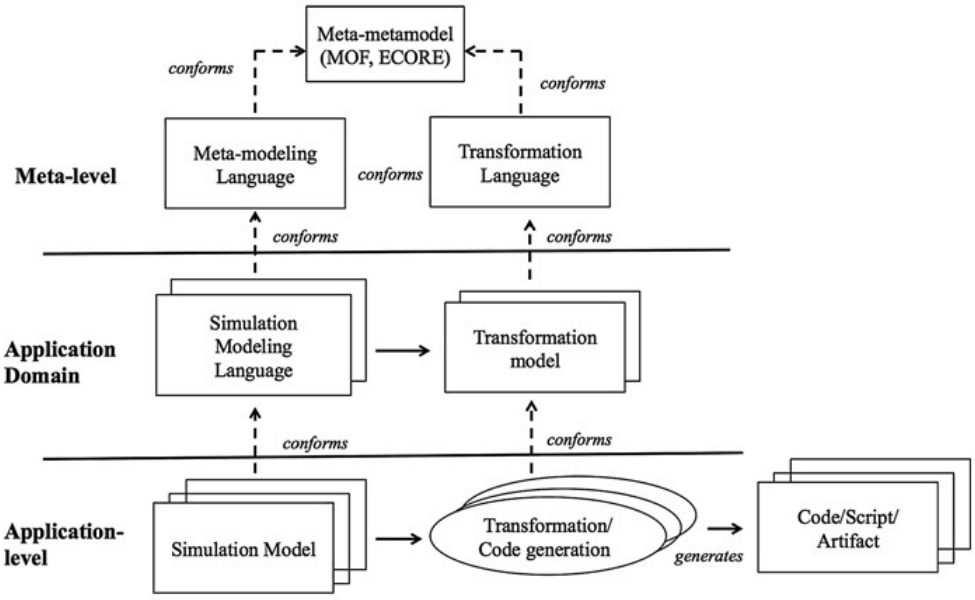

Figure 3.1: A multi-level view of MDE Methodology. Adopted from [TDOY16]

As shown in figure 3.1, according to [TDOY16], the Model Driven Engineering methodology can be separated into three dimensions: meta-level, Application Domain, and Application-level. They further say that at the Meta-level dimension, model-driven engineering provides meta-models and facilities and languages to define the meta-models, such as meta-metamodels. Meta-models derived by the meta-level dimension of the methodology are used in the application domain dimension. A mapping from the conceptualization to the realization space is made at this level, based on the meta-models as the modeling language and the transformation model. Finally, a model of the real scenario is made at the application level using the modeling language derived in the application domain, and then using the transformation mechanisms, a code/script/artifact is created. [TDOY16] Within each dimension, there are two rows. In the first row (modeling level), it defines simulation models, meta-models, and meta-metamodels. In the second row (transformation level), it automates the derivation of the artifacts (such as code, scripts, and XML documents) at the realization level by using transformation specifications, languages, and applications. [TDOY16]

CHAPTER 4

Related Work

In this section, the results from the Multivocal Literature Review are presented. Initially, the related sources are presented and grouped if possible, and in the last section, conclusions towards the research questions are presented.

4.1 White Literature Review Summary

Following the initial filtering process, 39 sources remained for a more in-depth assessment of their relevance. After a thorough examination and the exclusion of non-relevant materials, eight sources remained. As the percentage of related papers shows, topic naming is not a priority in the literature. From the relevant sources, most of them focus on the structure of the topic name and how it relates to performance and security, rather than the semantic meaning of the name.

Multiple sources have focused on basing access control on the topic naming. Authors of [GGB20] focus on Attribute-Based Access Control (ABAC) based on topic naming. They define a security model that allows producers to push messages to specific topics and consumers to subscribe to topics based on the rules of the security model. They specify the rules by creating topic structures using variables and constants, which points to the necessity of having a well-structured topic tree. The Authors of [BGG19] also focus on ABAC; however, they do not focus on the topic name, as they specify an administrator per topic who specifies the access rules per topic. They also control the subscriptions and publish to the topic names.

Hwang et.al. researched in [HDZ+20] a network of brokers and message routing between them based on a topic name. Each broker has its own topic tree where users are subscribing. The leaf broker gets the message and tries to find the topic in its topic tree, and in case it does not find it, the broker subscribes to other brokers and forwards the

message so they can process it if they can. Again, there is logic based on topics. However, the authors do not specify the topic structure.

[MEBAEK24] designs a system for a software system for land sustainability. They specify a topic structure's importance and define it as shown in Fig 4.1. It can be seen that the structure is created such that each topic contains the location (area code) of where the data comes from or where the command should be executed. As they control only the sprinklers, the topics that receive the commands that control the sprinklers also contain the hardcoded value "Sprinklers".

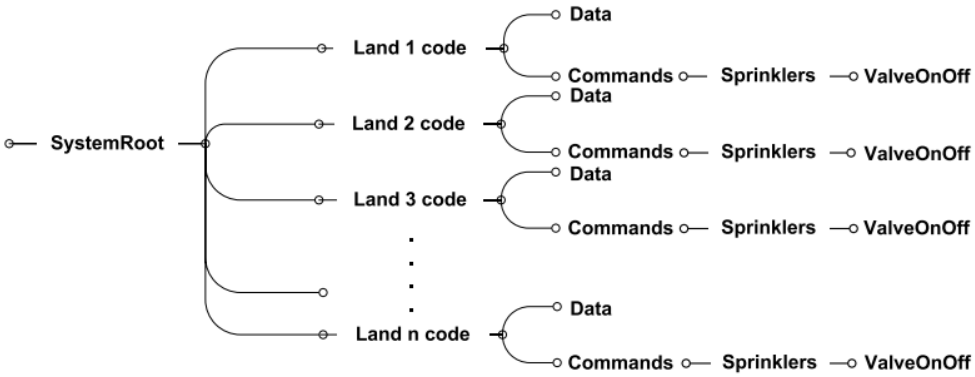

Figure 4.1: Topic Structure for Agriculture Implementation, as adopted from [MEBAEK24]

A design of a multi-microgrid communication platform was developed in [JMHL19]. The authors have specified a multi-protocol communication platform where MQTT is used for inter-microgrid communication. For this purpose, they suggest the topic name structure of "dataTypeName/DeviceName/MicroGridName". An interesting fact about this topic structure is that they define data type so the consumer can understand how to read the message based on the topic name, then the device name helps the receiver identify the device and to which microgrid it belongs. This structure is the opposite of the many structures where the hierarchy was built on location being the more general information, and data type or device identifier being the more specific information. It shows that the structure depends on the implementation, and the hierarchy cannot be fixed to start with Location.

The authors of [vdBGV+18] define an MQTT Tree for a smart home implementation. They define a Topic as the subject of the message that is published, and state that a topic name allows the broker to filter messages. The authors take the requirements for this tree from the basic principles of MQTT. The first of those principles is that the broker floods messages to each subscriber of the topic. They state that if one subscriber subscribes to too many topics or if the messages are not filtered properly, then there is a risk of overloading the subscribers and making the system not scalable. The authors say

1	2	3	4	5	6	7	8	9
api	<id api>	room	<piece>	device	<device type>	id	<id>	<request/response/indication>

Figure 4.2: MQTT Topic Tree adopted from van den Bossche et. al. [vdBGV+18]

that an example that would run into such a risk is a system where the designs remove data from the topic name and put it in the message in a JSON format. According to them, this causes part of the analysis to be taken away from the broker and assigned to the subscribers. They further advise assigning as much of the analysis to the broker (have more specific topic names) if the devices have limited resources (processing capacity and energy capacity). Therefore, they advise creating the most specific topic name by using precise, explicit, and hierarchical topics. The second principle on which they design the MQTT tree is the wildcard subscription to topics. Therefore, they suggest creating a topic structure that has a general root element and leaf elements with exact descriptions. Based on these system requirements, the authors propose a structure of 10 layers, which are to be considered in pairs. Each pair of levels consists of the first layer, which allows us to interpret the meaning of the second layer. They specify the first two levels, where the first one is hard-coded "API" and the second level is the version of the API. The following three pairs are called "free levels", where the developer could choose what they define; these layers are used to specify the place, nature, and equipment of the message. The last level is used to determine the nature of the message (request/response/indication)

Standardization of topic and message structure in publish/subscribe protocols is the goal of [FJSGK22]. According to the authors, the absence of standardization of the topic and payload results in one of the best benefits but also drawbacks – flexibility. The authors say that to facilitate interoperability and reduce the processing effort needed, the IoT application's data must contain meaningful context. Additionally, they specify that the context provides metadata such as device identifications, sensor identification, and deployment location. In their research, the authors found existing research on topic structure. One is [TOH+16], which is described later in Section 4.2.2. The other structure that the authors mention is "networkName/ nodeID/ country/ districtState/ cityTown/areaDescription/ area/ building/ room/ control"; however, they say that this topic is too long for ultra-low-power devices. The resulting structure presented in [TOH+16] allows the authors to specify the requirements of the implementation, and one of them is that the topic should identify the data source that creates the data. They specify further the information that is necessary for the topic:

- Domain: an IoT architecture can be in multiple domains, some possible values are: school, healthcare, transportation. The designer can choose to use a numeric sequence.
- Device identification: deviceID is the unique identification of the device's MAC address.

- Sensor type: The type of sensor is noted by objectID
- Sensor identification: the unique identification of a sensor, also noted as objectID. The length of the ID should be at most 5 bytes, and an additional 1 byte in case a device has multiple sensors of the same type.

Thus, the authors define the structure of the topics as: /<domain>/<deviceID> with additional identification of the objectID and instanceID in the payload.

This paper [GBKN23] transforms IoT events into meaningful business events. The authors concentrate on the process of converting real-world data into information and subsequently knowledge that benefits users and their interactions with the physical world. The designed architecture enables two-way bidirectional communication. They propose that the architecture focuses on two stages: event transformation and complex event processing (CEP). The implementation is done using two protocols: HTTP and Kafka. The authors design the system based on the concept of dumb broker and smart consumer, where most of the logic is on the client (subscriber) instead of the broker. In protocols such as Kafka, where the messages are stored on the broker, this concept allows scalable and fault-tolerant data distribution. In their research, they found that meaningfulness in the context of CEP can mean but is not limited to semantics, schema, correlations, associations, topics, origin, domain knowledge, and data quality. In the second stage, the authors define complex transformations such as time-windowed average value calculation of the enriched and filtered messages or anomaly detection in the dataflow of another component. They finally present the final architecture in real life on a smart farming application domain. This research shows us that metadata can be collected for a message if the topic is message-specific.

4.2 Grey Literature Review Summary

Following the initial filtering process, 45 sources remained for a more in-depth assessment of their relevance. After a thorough examination and the exclusion of non-relevant materials, 22 sources remained, along with an additional seven sources identified through those sources. The topics of the sources could be split into three categories: General MQTT Protocol Specification and best practices, Topic structure based on an= specific implementation, topic naming standardization attempts, then a couple of papers are presented which were found with the gray literature and finally an implementation of a similar tool in the industry is presented.

4.2.1 MQTT Protocol Specification and Best Practices

In 2.2, a general description and requirements of the protocol are presented. In this section, in addition to the official specifications, information found in blogs that supplements the official specifications is presented. [Hiv24] stresses that properly defining and structuring topics ensures effective data exchange and handling. [Hiv24, Bra23, Pet19, Pon, Ama21]

provide best practices on creating topic names. They initially provide the specification of the MQTT Protocol. Firstly, the topic contains at least one character. Secondly, the topics can include spaces, which to make topics more readable. Thirdly, topics are case sensitive, which makes the following topics different “home/livingroom” and “home/livingRoom”. Finally, a topic can start with a forward slash; however, this represents that the topic has a first topic layer as an empty string. In addition to this [Hiv24, Bra23, Pet19, Ama21] specify what they found as the best practices through the experience of building MQTT Systems. The initial best practices are regarding the characters used for naming the topic:

- Topics should not start with a forward slash, as it introduces an unnecessary topic level, and [Bra23] note that this adds an unnecessary load for the broker when checking the subscribers of the topic.
- Topic names should not contain spaces to separate words in a topic layer, because multiple characters that represent an empty space could cause a problem in the processing
- When defining the topic, only ASCII characters should be used. Additionally, non-printable characters should be avoided. Not following this makes it challenging to identify typos or matching issues.

The next category of best practices found in [Hiv24, Bra23, Pet19, Ama21], is the length of the topic name and how specific it should be:

- MQTT Topics should be kept short and concise - this helps to keep the message size smaller and decrease the load on the network.
- Client Id or Unique identifier should be embedded in topic names – it helps improve the message identification and security. Knowing the sender helps to control publishing permissions.
- Although topics should not be large, they should be specific, to promote clarity on what information is sent via the topic, and help with better handling of new features, such as retained messages. In [Ama21], it is specified by saying that any relevant routing information that could identify the entity should be included in the topic name. Examples are location, unique identity of the IoT device, and the thing name. They discourage writing contextual information (such as session identifier, the return topic, requestor identifier, and logging) in the topic name and sending it in the message payload.
- Additionally, only [Pet19] notes that for enhanced routing of messages, once they are received, unnecessary levels of the topic name should be avoided. For example, if all of the sensors are at home and all topics have a level home, then this does not bring any new information but puts a higher load on the broker.

Finally, there is best practice on the structure of the topic levels:

- [Pet19] Notes that topics should categorise information into groups and that the structure of the topics should resemble that of a tree so that wildcards can be used accordingly.
- [Pet19, Ama21] note that it is important to define a naming convention and that it should be followed so any new topic can be incorporated in the existing implementation.
- [Ama21] stresses the importance of following the pattern of general to specific topic levels. This means that the most general topic levels are on the left, e.g., Country (for smart city implementation) or "home" (for smart home implementation), and more specific on the right, such as measurement type (e.g., temperature)

4.2.2 Topic Naming conventions/standards

As professionals and IoT enthusiasts started implementing MQTT in their solutions, the need for standardising and formalising how MQTT is used emerged. For this reason, there were a few tries to start standards and conventions. There is Sparkplug, which was defined for the industrial IoT, Homie, and MQTT Home for some smart home solutions, and a few often cited blogs that try to generalize the structure of the MQTT topic names.

Sparkplug

One of the first additional specifications, at the beginning of 2016, was Sparkplug, which was extended right away by the end of the year by Sparkplug B [Lin19] regarding the payload specification. This specification is widely used in the Industrial IoT. Sparkplug defines topic structure as: "namespace/group_id/message_type/ edge_node_id/[device_id]", where:

- namespace: is a predefined prefix, which further defines the structure of the topic and the possible payload. The two possible options right now are "spAv1.0"and "spBv1.0".
- group_id is the name of a logical grouping of MQTT EoN Nodes. The name should be as short as possible and not contain the special characters "+", "#", and "/".
- message_type is the type of message which is written to the topic. These types are predefined in the Sparkplug specifications
- /edge_node_id - Is the id of the MQTT EoN Node. The combination of group_id and edge_node_id should be unique to 1 node. Two nodes can have the same ID if they are in different groups.
- device_id - This is an optional topic level. It is the ID of the actual physical device. Sometimes the device is the edge node, and because of that, this level is not needed.

Homie

Another convention built on top of the MQTT protocol is the Homie convention [Hom24]. The goal of this convention is to allow devices to share data without knowing anything about the internal implementations, hardware, or software. The Homie convention standardizes how devices and services announce themselves and publish the structure of their data, to allow automatic discovery, configuration, and usage. It is specified in some of the blogs found online, such as [Bat19].

The Homie convention defines the topology of an IoT solution. Based on this topology, there are different topic schemas. The topology is defined as:

- Devices – a physical piece of hardware. Examples of devices are individual sensors (e.g., thermometer in the living room), a car, or a coffee machine
- Nodes – Nodes are parts of a device. For example, a car has doors, wheels, lights, etc..
- Properties – Properties are characteristics of a node. For example, a coffee machine might have properties of water level, water temperature, and coffee bean level.
- Attributes – Specific attributes of devices, nodes, and properties. A precise definition is important to allow the automatic discovery of devices. Examples of attributes are the IP Address of the device, the name of a node or a unit of a property.

The topic names can have the following structure:

- Device Topics: homie/{device_ID}
- Device Attributes: homie/{device_ID}/${device_attribute}
- Device Topics: homie/{device_ID}/{node_ID}
- Device Topics: homie/{device_ID}/{node_ID}/${node_attribute}
- Device Topics: homie/{device_ID}/{node_ID}/{property_ID}
- Device Topics: homie/{device_ID}/{node_ID}/{property_ID}/${property_attribute}

Raspberry Valley

Raspberry Valley is a Swedish group of Raspberry Pi enthusiasts that helps people who want to develop new implementations with Raspberry Pi devices. According to them and their experience, a topic name should not be too long because it is harder to read, but also not too short because it will miss some information. According to them, the structure of an MQTT Topic Name should be:
“device-category/device-id/payload-context/payload-differentiator” [mqt]
Where:

- device-category – logical grouping of devices such as home, office, machinesInTheField, etc [mqt].
- device-id – unique identifier of a device, such as a serial number or unique ID in the IoT platform [mqt].
- payload-context – context of the information in the message. This should be standardized example values are 'temperature', 'pressure', 'location', etc. [mqt].
- payload-differentiator – this field is optional, an example would be Front/Back to differentiate movement in the front or back of the house [mqt].

Tinkerman

This blog [Pé12] is an often-linked naming convention. It is cited by other blogs as well as in forums of IoT platforms such as [Com20] and [Bat19].

The blog initially classifies the topic naming approaches by the type of information conveyed into Physical and Semantic approaches. According to the physical approach, the topic is called by what the producer is (e.g., id of entity) and/or what it is attached to (e.g., type/name/id of device). According to the author, this approach is slightly more machine-friendly. On the other hand, the semantic approach names the topic by describing the thing with information such as location and measurement type. This approach is more human-friendly and preferred as it is easier to maintain. The author states that most of the implementations that use the semantic approach start with the location of the entity, with the order of most general to most specific location left to right. This structure, according to the author, is essential to allow scaling in the future. He further states that after this information is given, then the measurement is the next topic level, and after that, some keywords to define the information/command, for example "status".

The author then specifies what kind of information is usually stored in MQTT topics based on experience and MQTT topics found online, and those can be split into four categories :

- Metadata: information such as location, timestamp, units, and alarm
- Aggregation: time ranges (e.g., last24h, yesterday), operators (e.g., max, min, average)
- Actions: for example, get, query, or set
- Structure-related: for example, raw or JSON

Steve's Internet Guide

Steve's Internet guide is a blog by Steve Cope, a specialist in the field who has written the book "MQTT For Complete Beginners: Learn The Basics of the MQTT Protocol". In one blog post [Ste24], he describes possible topic and payload design schemes. He starts by explaining where the information should go, whether the design can be that each entity has an individual topic, or the system designer designs one topic per room, but then the entity has to be in the message content. The author further specifies that the naming scheme approaches range from putting as much information as possible in the topic name to adding as much information as possible in the message payload. High-level topic grouping of devices, given sensor name, and function(which should be executed on receiving the message. e.g. , status, set, get) could optionally be included in the topic's title. In comparison, the message must contain the payload data and, optionally (if not in the topic), the sensor ID and the function to be executed.

Cope further specifies his best practices, which are:

- One topic name should be used either per device or a small group of devices and sensors.
- The topics where commands and data are sent should be separated.
- Data in the payload, which can contain multiple device attributes, should be JSON encoded.

In the end, he proposes a naming convention for public data. He focuses on status updates, command topics, and response topics. He splits the structures into:

- Command topic with structure: "base topic/cmnd/device_id/command".
- Command response topic: "base topic/response/device_id or base topic/response/device_id/result".
- Status updates on: base "topic/status/device_id/state and base topic/status/device_id/info".

MQTT-Topic Naming Criteria of Open Data for Smart Cities [TOH+16]

A paper [TOH+16] published in the 2016 International Computer Science and Engineering Conference (ICSEC) researched topic naming criteria in the field of Smart Cities. According to the results of the evaluations, having the standardized criteria as presented in the paper significantly decreases the time for subscribers to find the data they are looking for. The authors stress that in open source data, such as smart city data, there are multiple producers, and if they produce data on topics that have different structures, the subscribers would not know where to find the data they are looking for. They define the topics to be in the order of Objective, Location, and finally Owner, with each category having one subcategory. The result is "Objective/Subobjective/Location/Sublocation/Owner/Subowner" where:

1. Objective – It has predefined main objectives (Environment, Energy, Human, and Living). On the next layer, there are the predefined sub-objectives (For example, for environment, there are air, water, soil, waste, and forest), and the producer can create a new one, but it has to be added to the list of sub-objectives
2. Location – The location also has predefined values. The main location is a country, the sub-location is a province within the country, and finally, there is a user-defined location, like a university.
3. Owner – is specified by the publisher and not predefined.

The structure has multiple information in the topic, however, the structure of those categories does not mix, and it is always left to right less to more specific.

MQTT-4EST: RulE-basEd WEb Editor for Semantic-aware Topic Naming in MQTT [PMK22]

The authors of [PMK22] specify a tool to help define a structure for MQTT topic names based on a predefined ontology. They stress the importance of a semantic understanding of the data shared in the topic name and provide an editor for creating topic names based on an ontology. While defining the tool, they found no specification of what information or structure the topic names should have, other than the hierarchical structure. The authors propose a small ontology for testing purposes, such that they have formalized the structure of the location of a sensor, such that location is limited to building → room → sensor, where each child is mapped as a property of the parent. They created a proof of concept that the structure of a topic name can be specified using an ontology, however they did not specify an ontology which would cover all implementations.

4.2.3 Topic Naming implementation specific patterns

Some of the specifications for topic structure were done with specific implementations in mind, such as specific firmware like Tasmota or protocol mappings such as OPC UA to MQTT.

Tasmota

Tasmota is an open-source firmware developed for Espressif chipset-based devices. It specifies how they can communicate using different protocols, one of which is MQTT. The Tasmota MQTT specifications are defined in [Com24]. For MQTT, Tasmota defines topics for data transfer as well as for commands. The topic structure must contain the following tokens:

- %prefix% – it can have one of 3 values “cmnd”, “stat”,"tele" (default)
- %topic% – it can have one of 5 values, which are the five defined topic types: “Topic”, "GroupTopic”, "ButtonTopic”, "SwitchTopic”, and “MqttClient”

- %hostname% – the hostname of the device as it was defined by the user
- %id% – Device's MAC Address

To specify a topic, a mix of these tokens can be used. For example, some topics might be:

- %prefix%/%topic%/ default
- FullTopic tasmota/%topic%/%prefix%/
- FullTopic tasmota/bedroom/%topic%/%prefix%/
- FullTopic penthouse/bedroom1/bathroom2/%topic%/%prefix%/
- FullTopic %prefix%/home/cellar/%topic%/

Some platforms allow limited combinations of topic levels to enable auto-discovery. For example, the Home Assistant platform requires that the order of the tokens is first %prefix% and then %topic%.

PCVue

PCVue is an IIoT solution including a SCADA system as one of the many add-ons this solution has is a MQTT connection [PCV24]. As part of this connection, the system can be a publisher and/or a subscriber. They specify that a topic structure should have a hierarchical structure, and as an example, they provide the structure {BuildingId}/{FloorId}/{ZoneId}/{DataId}. However, the actual structure depends on the implementation.

OPC UA

OPC UA is one of the most used protocols in IIoT. It has a strict structure which makes it easier to process the messages, however, it also makes the transmission resource-heavy. For this reason, even the official documentation of OPC Foundation as well as OPC UA vendors provide documentation for connecting the two protocols [OPC24] [Lab24]. They specify that the message structure should have the following pattern:
prefix/encoding/mqttMessageType/publisherId/[writerGroup[/datasetWriter]]

Where:

- prefix provides a scope for the PublisherIds, it is a system-defined, and it is automatically handled by OPC Studio Components
- encoding specifies the encoding of messages sent to the Topic ('json', 'uadp', etc), and it is automatically handled by OPC Studio Components

- mqttMessageType specifies the content of the messages published to the Topic, and it is automatically handled by OPC Studio Components
- publisherId uniquely identifies a Publisher within the scope of the prefix;
- writerGroup (optional) is the name of a WriterGroup within the Publisher
- dataSetWriter (optional) is the name of a DataSetWriter within the WriterGroup

In this structure, they do not provide metadata of the entity, such as location; however, they specify the content of the message, which is highly necessary for processing the message.

Others

There were also a couple of other definitions for either one device or a small use of devices, which will be summarised here. There is the Vaisala ceilometer [Vai24] which has the following structure:
dt/vaisala/gateway-urn/device-urn/obs/format
What is interesting about the structure is that it does not contain device metadata except the ID. Still, it contains the device identifier and the format of the message for easier processing of the message.

Another implementation is the Aranet Pro implementation [SAF22], which defines the MQTT topics in 2 ways - raw (single values are provided included with detail info such as unit) and json (multiple values are sent in the same payload, eg. temperature and humidity; however, no additional information such as unit):

- raw pattern: `<root topic name>/<PRO base serial number>/sensors/<sensorID>/measurements/<measurement type>`
- `json pattern:` `<root topic name>/<PRO base serial number>/sensors/<sensorID>/json/measurements`

In these patterns as well there is no metadata about the entity, however there is information on how to process the message.

4.2.4 Influx data implementation with Telegraf agent

A blog post by InfluxData [Dil21] highlights the industry's need for a tool capable of processing data from the resource name. Among other tools, InfluxData has developed Telegraf, an open-source server agent designed for collecting and transmitting metrics. To deal with the potentially large volumes of sensor data being transmitted, they have introduced a mechanism to extract relevant information from MQTT topic names as part of the MQTT Consumer Input Plugin. According to InfluxData, practical parsing

facilitates improved data organization, filtering, and processing, specifically when dealing with large amounts of sensor data.

As part of their MQTT Consumer Input Plugin, InfluxData implemented a functionality to extract information from topic names and include it in the message data that is processed. Users are provided with multiple approaches to achieve this. The conventional method involves defining a regular expression (regex) to parse the topic name. However, the blog post [Dil21] introduces a new feature that will allow users to specify the meaning of each level of the topic or use an underscore "_" to note that a level has no semantic significance. Even though this development proves the industry's recognition of the need for topic name parsing, it still requires user input. The process still relies heavily on manual configuration rather than fully automated parsing based on pre-existing data.

4.3 Conclusions Towards Research Questions

4.3.1 RQ1: What kind of contextual information can be effectively encoded in MQTT topic names?

Existing research, implementations, and best practices show that the data incorporated into topic naming serves a critical role. It either aids the consumer to identify the producing device/sensor with the payload containing at least the value, or it embeds metadata about the payload within the topic name, facilitating message interpretation. In contrast, the device or sensor identification is provided within the payload. While all agree that a hierarchical structure should be used to create the structure of the topic name, the information that is in the topic name is differentiated by implementation.

Data that can be stored to identify a device or sensor:

- Device or sensor location - This information is usually divided among different topic levels to help with filtering if one wants details for a more general location, for example, the lights' status on the second floor.
- Device ID - unique ID of the device that produces the data or receives a command.
- SensorID - unique ID of the sensor at the device. It can be used in case there are multiple sensors on a device to identify each device.
- Device or Sensor type - sometimes devices can be identified also by the type, for example, sensor, thermostat, control unit, etc.
- Measurement type - what kind of measurement is measured by the sensor (what kind of sensor it is). This is used for identification in case there are multiple sensors. For example: temperature, humidity, smoke, etc.

Data that helps to process the message is:

- Message format - For example: raw, JSON, binary..
- Measurement type - although it is used for identifying the message, it is also important to read and understand the message
- Data Class - in case data is sent in a specific data structure, which is necessary for the consumer to know. For example, OPC UA communication via MQTT.

4.3.2 RQ2: What are the key challenges in harmonizing extracted metadata across different MQTT topic naming conventions?

In the existing research, all sources seem to agree that the source of the data must be identifiable either in the topic name or message. Many also agree that the information about the data, such as the message format, should be incorporated into the topic. However, due to resource constraints, topic names cannot be of arbitrary length, which prevents providing complete information in the topic name. The consumers have to be manually set up per system to be able to read the message. For example, the consumers in Home Assistant (a smart home platform) can read the value from the message based on the users' input. Otherwise, it reads the whole message as the value. Thus, one of the key challenges in harmonizing extracted data into the message is knowing the message format beforehand and reading the message so the metadata can be correctly incorporated. However, fixing this is not part of this thesis, as the tool expects at the time of the harmonization that the message is in a universal format.

Additionally, some implementations require that the data received by the consumer contains some mandatory fields or is transformed in a specific format, for example, for the OPC UA system [OPC24] to be able to process the data further. However, this information is not available data that the tool can automatically collect; this is often information within the platform's code that checks for this information.

Finally, information might be duplicated in the message and the topic name. An example is when one sensor was changed out for another, and both sensors measure temperature, but the new one measures in Fahrenheit and the old one in Celsius. The already defined topic contains C in one of the topic layers, so the user is aware of what unit the measurement is. However, when he puts in the new sensor, it is a lot of effort to set up and link a new topic name, so the user decides to reuse the topic and add the unit to the message. When the messages from the latest sensor arrive, they have two units defined. It could as easily be the other way around, where the outdated information is in the message and the current information is in the topic name. Deciding where the more correct information is coming from is crucial to avoid losing any data.

CHAPTER 5

Artifacts

This chapter presents the key artifacts developed as part of this master's thesis. These artifacts support the automated extraction and utilization of contextual metadata from MQTT topic names in IoT platforms. The artifacts resulting from this thesis are the metadata model, the metadata matching algorithm which uses the model, as well as the implementation which can be incorporated into any IoT platform. This section is divided as follows: first, the metadata model used to structure and store domain-independent contextual information is introduced. Next, the core artifact—the metadata extraction algorithm—is described in detail, including its design and operational logic. Then, the chapter outlines the integration of the artifact into the chosen smart home platform. And finally, the implementations of both the artifact and the integration into the IoT Platform are presented.

5.1 Metadata Model

Based on the related work discussed in Chapter 4, a conceptual model is developed following principles of Model-Driven Engineering (MDE) to define how relevant metadata should be identified and extracted from the topic structure. Initially, a structural model of the message format is defined, describing how the existing metadata in the system is collected and stored. As the authors of [PMK22] have found, a hierarchical relationship between levels of the correct processing of all levels of the topic tree is necessary. The relationships between different levels of the tree have to be carefully maintained. As the authors suggested, one class of information can have multiple children and multiple parents. As an outcome of this research, a hierarchical metadata model was designed using nested Python dictionaries, offering a structure functionally equivalent to JSON:

```
{
 "className": {
   "hasParent": [],
   "single": Boolean,
   "classInstanceID": {
    "aliases": [],
    "childClassName": ["childClassInstanceID",...]
   }
 }
}
```

Where:

- **className** – Text value. This value is the name of the metadata category and will be used as a field name when the value is harmonised in the message.
- **hasParent** – List of Text values. This property indicates whether the metadata class has a parent or not. If the value is empty, this value is probably found at the beginning of the topic name.
- **single** – Boolean value. This flag identifies if one field can be found multiple times within the topic name. This can happen for location, in case the values are stored as a location that can inherit other locations, instead of a structured location configuration of building, room, and floor.
- **classInstanceID** – this is the instance ID in the system, for example, sensorID. The value of the dictionary is an instance of the class
- **classInstanceID.aliases** – name or alternative names of the instance. For example, sensorID is one, but name could be the type of sensor
- **classInstanceID.childClassName** – This field has a list of string values. There could be multiple such fields depending on the classes that are children of the current class. For each child class, there is the field, className, with a value of a list of strings, which provides the ids of the instances of the child class that are connected to the current instance

5.1.1 Design Considerations for the metadata storage structure

In the design of the metadata structure, two approaches were considered for representing the metadata model: a simple, JSON-like hierarchical structure and a formal ontology based on Web Ontology Language (OWL). Each approach offers distinct advantages and disadvantages which are to be considered.

System integration, efficiency, and simplicity are the benefits of utilizing JSON and JSON-like structures. JSON is natively supported by many different tools and protocols,

is lightweight, and is simple to read and write. Because of this, it is especially well-suited for applications that require real-time processing. Additionally, JSON allows developers to have complete control over the form and interpretation of metadata without the need for external reasoning tools or ontology management. Anyone can use this product because of how simple the JSON tree structure is and how little knowledge is required. JSON-based methods can have certain drawbacks, though. First of all, it is challenging to formally explain relationships between entities since they lack explicit semantics. Second, it doesn't support extensibility outside of the developer's own conventions, validation, or inference. JSON formats cannot take advantage of standardized vocabularies or semantic web capabilities, making them more difficult to communicate or integrate across systems.

Conversely, there are a lot of advantages to employing an OWL ontology in terms of reasoning assistance, interoperability, and semantic richness. Domain knowledge, such as class hierarchies, relationships, and constraints, can be formally modeled using OWL. More complex use cases are made possible by this, including integrating with pre-existing semantic web ontologies, validating topic hierarchies, and inferring implicit information. Additionally, OWL-based models make it easier for systems that follow Linked Data principles or need to integrate with knowledge graphs to share and reuse data.

However, there are costs associated with these benefits.Both the learning curve and the tooling are made more complex by OWL. Development frequently calls for reasoning engines and ontology editors (like Protégé), which might not be easily accessible or appropriate for lightweight IoT setups. Furthermore, unless appropriately tuned or separated from real-time processing, OWL-based reasoning may be computationally demanding, which limits its applicability for high-throughput or latency-sensitive applications.

The preferred trade-off between simplicity and semantic power ultimately determines which of these strategies is best. JSON-like structures might be better suited for systems that prioritize quick integration, lightweight performance, and simple user onboarding. OWL offers a more robust long-term foundation for systems that want to grow toward more intelligent, knowledge-driven automation. The decision to use a JSON structure was therefore simple.

Based on the selection of a JSON-like form for the metadata model, other design choices about the structure of the metadata were made. In order to allow both subject specialists and non-technical users to establish or change metadata without the need for specialized tools, the format had to be kept as straightforward and user-friendly as feasible. Additionally, as the model's size and complexity increase, simplicity promotes scalability in terms of memory utilization and performance.

The initial decision that was taken into account was whether links should be bidirectional, parent-only, or child-only. Only child relationships are explicitly shown in the final structure; parent references are left out, and each class instance has references to its children. Several options were examined before choosing this arrangement. Firstly, parent-only linkages were taken into account. In addition to frequently requiring extra steps to filter or arrange children based on their parent class, this would also require the

matching algorithm to go upwards in order to ascertain context. Secondly, bidirectional connections—parent-child—were taken into account. This, particularly as the number of instances and nested layers rises, would provide substantial data redundancy while facilitating traversal in either direction. This might result in non-trivial memory overhead and make consistency maintenance more difficult. Finally, child-only linkages were the final and selected strategy. They make it simple to filter child instances by class and enable effective top-down traversal with little data duplication. Since most metadata extraction starts from a root-level class and proceeds downward (e.g., temperature → kitchen → celsius), this approach aligns well with common usage patterns. Given that the only disadvantage of the last option is that reverse traversal is more complicated, however it is rare and can be solved, the last option seems to be the practical choice in the trade-off between navigability, performance, and data compactness.

The final design decision which had to be made on the metadata model structure is whether it should be a flat list of instances each containing the link to the associated class, or if the instances should be grouped under the associated class object. The initial idea was to structure the instances as a flat list and only reference the class inside as that would make it much easier to traverse and find the matching instance. However this structure proved to be difficult in other areas. Specifically, as the class-based organization is lost it also makes it more difficult to preserve or enforce semantic hierarchies, as well as it is harder to efficiently filter or iterate through instances of a specific class.By grouping instances under their associated class, the structure allows the algorithm to limit the search space to relevant subsets, improving the performance of matching operations. Moreover, class-based grouping helps maintain a logical separation of domain concepts, improving readability and traceability of the model as it evolves.

5.2 Metadata extraction Algorithm

After the metadata is collected and preprocessed, the system can start processing the data from the topic name. The artifact was designed with some restrictions in mind as defined in Chapter 1. Firstly, it is expected that there is a processor for each message such that it provides the data to the artifact as a pair of the topic name (as a string) and the message (either in JSON or as a raw string). Regarding the message, raw is expected to be something like “27” for the temperature. Another limitation of this artifact is that it does not do any reasoning; instead, it provides enriched data to be used by other tools for reasoning. For this reason, to have as flexible a structure as possible, the result of this tool is also in JSON Format. The visual idea of the process is presented in Figure 5.1. Additionally, the process is described step by step as:

1. On startup, load all of the contextual data, which includes processing of the metadata such that they are written into a list of metadata categories, such that each element in the list represents a tree of metadata. For example, one tree is the location and another tree is the message description; they are not connected, so

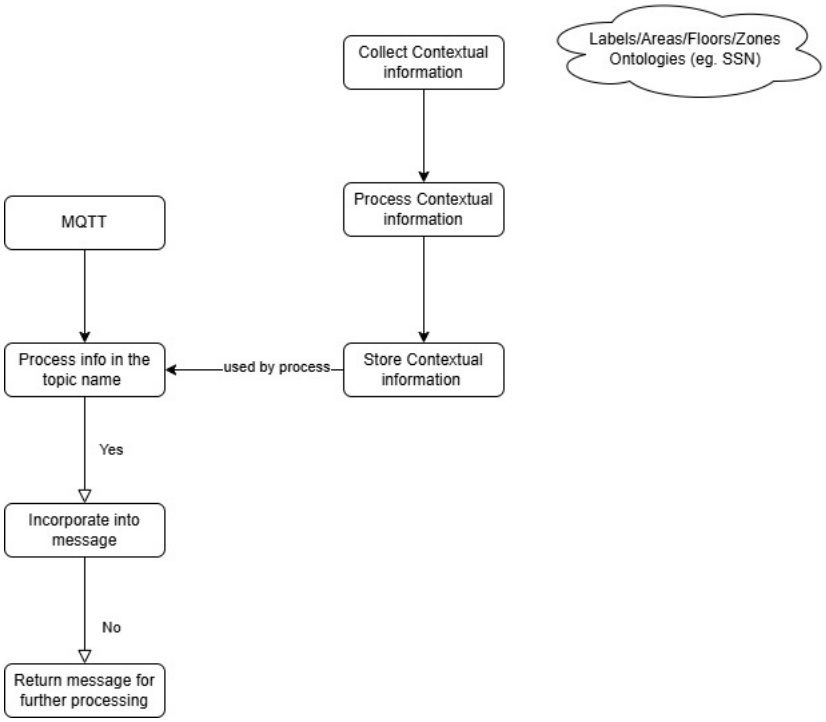

Figure 5.1: First message Example

they don't need to be checked in a specific order. How this is done in terms of the proof of concept can be seen in section 6.3.

2. On message received on a specific topic, the following process is taken:

   a) Is the topic already processed?
      - If yes – fetch from the cache of already processed topic names and go to step 3.
      - else continue to b. This process is defined as well by the Algorithm B.1.

   b) Go through each level of the message (left to right due to the hierarchical structure) and check if it is part of the metadata. The topic name, which is checked, is first cleaned up as:
      i. Split topic by "/"
      ii. Check by exact name

   c) The metadata categories are compared starting from the category that has no parent. Once a match is found, the rest of the topic is processed with the child categories. In case a category is not matched in any of the topic levels, its children are ignored.

   d) If part of the message name is not in the stored and processed context, ignore it.

3. Add {topicName, topicValue} pair in a cache in memory. Where the topic name is the full topic name, and the topic value is the dictionary of matched metadata for that topic name. This storage is reset on system restart.

4. Integrate into message:

   a) Based on the type of metadata, the field name is determined. In the structure above, the name of the field is {className}

   b) In case the message has a field which has the same name as the {className}, the value from the message is taken.

The details of the metadata matching algorithm are shown in Section B.3. The topic matching algorithm B.1 is the top-level algorithm that makes use of B.3 and B.2. The topic matching algorithm B.1 has three functions, the initially called function is **MATCH_TOPIC**, which initiates matching of categories for each topic level. It goes through each topic level, checks which of the following metadata categories can be matched by calling B.3, and then calls **MATCH_CATEGORY** function to match the topic level to the metadata category. This function goes through each of the instances in the category (only the filtered instances as done by B.2) and for each example until a match **MATCH_INSTANCE** function is called. **MATCH_INSTANCE** function checks if the topic level matches the instanceID or its aliases.

To improve the correctness and performance of the matching algorithm, the instances within the categories are filtered in B.2 such that the cases that do not have parents in the metadata matches of the parents are filtered out. This way, if there are topic names such as home/floor1/bathroom and home/floor2/bathroom, the correct bathroom is chosen. In addition to this, there is no need to compare rooms from the second floor if we already know that we need to match only rooms from the first floor.

Finally, as the topic name should always be in a hierarchical structure, this means that the categories should always be matched in this hierarchical order. Algorithm B.3 finds the next categories that should be matched by the algorithms above. It does this by choosing categories that have no parents, or else it returns all categories. This is done to make sure no categories are left over.

5.2.1 Complexity of the algorithm

The complexity per function is:

- MATCH_INSTANCE only checks once if the topic level is the list of aliases. This makes the complexity of $O(1)$
- MATCH_CATEGORY calls MATCH_INSTANCE once per instance of the category. If we say I represents the average number of (cleaned up) instances per category, then the complexity is $I * O(1)$, which is the same as $O(I)$

- CLEANUP_INSTANCE checks for each parent if it has instances, and if so, then includes only those in the calculation. For each matched category, which is at most the number of categories C, the cleaner method puts each child instance (I) in the result. Which makes the resulting complexity $O(C * I)$
- FIND_NEXT_CATEGORIES goes through each category in the list and filters by categories which have no parents, this makes the complexity $O(C)$ where C is the number of categories. As the processed categories are removed one by one, the number of categories decreases with every call.
- MATCH_TOPIC iterates through the topicName t times, where t is the number of topic levels (‘
’ separated strings). And for each topic level first calls the FIND_NEXT_CATEGORIES ($O(C)$) then cleans up the metadata category if there is at least 1 match already by calling CLEANUP_INSTANCE up to the number of Categories which is ($C * O(C * I)$) and then matches the topic to the category by calling MATCH_CATEGORY($O(I)$) and then for updating the result and removing the metadata category from the list which contains the categories to be matched $O(1)$ each. This makes the final complexity $O(t) * (O(C) + C * O(C * I) + O(I) + O(1) + O(1))$ which simplified is $O(t * (C * I(C + 1)))$ which simplified is $O(t * C^2 * I)$

As the top method of the matching algorithm is MATCH_TOPIC, the complexity of the algorithm is also $O(t * C^2 * I)$ where:

- t is the number of topic levels
- C is the number of categories
- I is the average number of instances

Given the complexity of the algorithm, it can be improved by the data that is provided. This can be done by decreasing the number of topic levels, metadata categories, or the average number of instances (which will be compared). Having shorter topic levels is a suggestion not just for this artifact, but a general best practice, as many IoT devices are resource restricted. Another option is to limit the number of metadata categories; however, this depends on the data in the topic names and cannot be controlled much. Finally, the average amount of instances can be easily decreased if the data is correctly structured. For example, if there are 10 rooms in the house, three on the ground floor and seven on the first floor, if the metadata is correctly modeled for each topic on the ground floor, the number of instances decreases by 70%.

5.2.2 Considerations on the performance of the algorithm

As the steps of the algorithm were developed multiple considerations had to be done in order to improve the performance and correctness of the algorithm.

The first consideration which had to be made was regarding the comparison of the topic names with the existing metadata. The topic naming should be done carefully, such that a clean topic tree structure is constructed, as noted by the best practices in the gray literature. It was considered wether a fuzzy matching algorithm or a strict matching algorithm. However the fuzzy matching had 2 disadvantages. Firstly, it allowed the user to have slightly different topic levels which are considered as separate trees by MQTT and would increase the complexity of the topic tree. Secondly, this made the matching slower than with regular matching. The disadvantage of the strict matching algorithm is that it would miss typos, however there is a workaround in this case such that in case of a typo the misspelled instance name can be added as an alias. Thus to improve the performance of the algorithm as well as push the user towards cleaner structure the strict matching was used.

The second consideration was on the order of the comparison of the instances. The initial idea was to have a flat list of instances and always match from the full amount of instances. However even tough it was easier to do so in Python, it decreased the correctness of the matching algorithm in some cases. An example is a house with two floors, each having one bathroom. The house model would contain 2 bathroom instances - one for each floor. The topics would be named house/floor1/bathroom and house/floor2/bathroom. In this case the bathroom part of the topic would match 2 times and might cause confusion in the semantic understanding of the topic name. In addition to that as the decision on the metadata model was to not have a flat map this decision decreases the amount of comparisons need to be made and thus makes the algorithm quicker.

The final consideration which had to be made was the ordering in which the topic levels were supposed to be matched. Initial idea was to do them in parallel and compare each level to all instances. However, although having comparison in parallel would make the algorithm quicker, this could lead to false positives as in the previous example. For this reason the order of the matching is per the recommendation of the existing literature - in hierarchical order.

5.3 Proof of concept

As proof of concept, the developed artifact will be integrated into an existing Smart Home Platform. During the evaluation it will be shown that this artifact can be incorporated into any platform. What needs to be done for this to happen is to initially collect and format the metadata and then use it to match it to the MQTT topics.

5.3.1 Choosing a Smart Home Platform

Many open-source and commercial Platforms allow users to implement IoT in their homes. They enable easy connections to different Protocols and different types of devices to make it easier for users to control their homes. Setz et al. [SGI+21] researched which of the home automation tools are the best based on predefined features. They initially looked

online to find any possible home automation platforms, and based on different search queries, they found 34 distinct tools. Then they limited those to top 4 based on the activity within the project (how many commits in their repository), the popularity on the project (amount of stars in the code repository), how long since the last activity (commit) on the project to penalize inactive projects, whether the project has documentation and finally the amount of contributors because the more contributors within the project the higher the chance that the project will keep evolving. Based on these criteria, they chose the following four systems for further inspection:

1. Home Assistant - an open source Platform with some paid features, such as Home Assistant Cloud, written in Python
2. Demoticz - open source Platform with some paid features, such as Geo-fencing, written in C++
3. OpenHAB - an open-source Platform written in Java
4. ioBroker - open source with paid cloud features written in JavaScript

Setz et. al. further investigated these four solutions. They created use cases based on Features as seen in Figure 5.2 and selected those features based on a survey and the authors' experience [SGI+21]. The five high-level features on which the solutions were compared are:

- Visualisation (F1),
- Localization (F2),
- Notification (F3),
- Data-Handling (F4),
- Interaction (F5)

In addition to these 5 Features, the four solutions are evaluated based on 34 distinct criteria, which could be categorized into eight groups:

- Popularity and Community
- Pricing
- Setup Complexity
- User Interface and User Experience
- Security and Authentication

User Cases \ Features	F3.1: Mobile Notifications	F4.3: Automation Rules	F2.1: GPS-Tracking	F4.1: Sensor – Read	F5.1: Mobile Remote Control	F4.2: Device – Actuation	F4.4: Media Streaming	F5.2: External API Calls	F1.1: Display (Paper-White)	F1.2: Dashboard	F2.2: Presence Detection	F5.3: Scheduler	F5.4: Biometric User Auth.
In case of a fire, I would like to be informed and receive notifications on my mobile phone.	✓	✓		✓									
When it is winter and it is cold, I want my heating to turn on before I get home from work.		✓	✓	✓		✓							
Even when my lamps and light fixtures are from different manufacturers, I want to be able to control all of them from one device.					✓	✓							
While doing the groceries I want to be able to access a video stream inside my fridge to check its contents.							✓						
I would like to know when I should empty my rain barrels in order to benefit the most from an upcoming rain storm.	✓	✓		✓				✓					
I want to see, in real time, much power my solar panels currently generate.		✓		✓					✓	✓			
When I leave the house, and I am the last person to do so, all lights should be turned off in case I forget.		✓	✓			✓					✓		
If the CO detector is low on battery, I would like to receive an email notification that the battery needs to be replaced. This will avoid triggering the alert that goes off when the battery is low, and usually causes panic as people may think CO is leaking into their house [14]	✓	✓		✓									
Every morning at 5:00 AM start brewing my coffee automatically so it will be ready for me when I wake up [14].						✓						✓	
During the day, whenever I walked into the bathroom the light would come on. However, after a certain time, when it is night, when I walk in the bathroom I would want a much softer light to come on [14].		✓				✓					✓		
When someone rings the doorbell it triggers both the television and the lights. If the TV is on then when the doorbell rings the TV should mute itself and/ or pause depending on what the input the TV is. Additionally, the lights should change color to indicate someone is at the door, in case the doorbell was not heard [14].		✓		✓		✓							
I want the sprinkler system to water the lawn only when: it is not raining already, and it is between 4 am and 6 am, and the temperature is above 50 degrees Fahrenheit. [14]		✓		✓		✓		✓					
When I am home and shut my windows, I want all of my doors to lock automatically [14].		✓		✓		✓							
I would like to receive a notification at home when the stock market passes a certain threshold.	✓	✓						✓					
I only want people I know, or people I have given permission, to be able to access my smart home.					✓			✓					✓
My home dashboard should indicate how many unread mails I have.									✓	✓			
My appointments for the day should be visualised on the display of my dynamic calendar, my garbage collection schedule should also be integrated.								✓				✓	

Figure 5.2: Specification on how features of SmartHome Platforms relate to different use cases. Adopted from [SGI+21].

- Extensibility and Support
- System Performance
- Software Quality

While none of the home automation platforms got a perfect score, all of them scored higher than average, which shows that these platforms are quite evolved. Each category could be valued to a max score of 5, and in the end, they were averaged for the final score. The results are that Home Assistant is in first place with 3.9, openHAB in second place with 3.7, ioBroker in 3rd place with 3.4 score, and finally Domoticz with a 3.1 score. The details of each category can be seen in [SGI+21].

After the initial literature review on the different platforms was executed the compatibility with the goals of this thesis was also investigated. First was the flexibility of the configuration, secondly the structure of the home, and thirdly the easiness of testing. Given the above review of [SGI+21]. as well as the considerations in terms of the artifact the chosen platform is Home Assistant.

5.3.2 Limitations of the Integration with Home Assistant

After deciding to use the Home Assistant platform for the proof of concept, the platform was further examined to determine the integration's viability and limitations. Despite Home Assistant's broad support for MQTT-based device identification and setup, a number of issues were found that affected the artifact's evaluation and design. A big part of the implementation is how can the artifact be integrated into the current implementation. A few options which were considered were an integration which would consume all messages from the MQTT integration and then forward them for further processing, extending the MQTT integration with a separate integration for metadata matching and and enabling the processing with a configuration flag only.

During investigation of the first option, an integration which would consume all messages from the MQTT integration and then forward them for further processing, it was concluded that it is not possible to extend the MQTT integration using additional modules and plugins. There were extensions of the MQTT Integration, however in this case it was not possible to configure any custom consumers as it is done using the MQTT Integration, the only configured consumers were the ones set by the integration developers. As an alternative, the second approach was investigated - implementing an interceptor to all of the messages in parallel to the consumers or before the actual consumers. This strategy raises the possibility of data loss since it complicates the system and can result in buffering problems or race conditions, particularly in high-throughput situations.

The last option was to integrate the artifact into the MQTT Integration using configuration so that the users can choose whether they want to incorporate this artifact into their implementation. However, the last option proved to be infeasible as there are limitations with the configurations of the MQTT integration. More specifically, Home Assistant

only allows configuration for the predetermined domains - which are types of devices, for example Button. So the final decision was made to implement the artifact within the MQTT Integration.

Due to these limitations the final proof of concept is an extension of the MQTT Integration such that it always fetches the metadata at the start and matches the metadata on each received message. The System Overview is shown in Section 5.3.4. Furthermore, as the artifact is to be implemented as part of the MQTT Integration, there is a limitation to the programming language which can be used for the implementation of the artifact. As the whole Home Assistant tool is implemented in Python, the implementation of the artifact will as well be written in Python.

5.3.3 Metadata collection in Home Assistant

Context in Home Assistant

The context in Home Assistant is the additional data that can be assigned to entities and devices to identify them or give more information. Data from an MQTT listener is linked to entities; however, not all contextual data can be linked to entities. For example, floors can only be assigned to devices, which adds extra processing power and access to the database to match the entity to a floor on some necessary automation. Based on the metadata defined in section 4.3 and data found in Home Assistant, the following categories can be found in Home Assistant.

- Location: The location is limited in Home Assistant. The location classes in Home Assistant are Floor and then more specific Area. Each label can be linked only once.
- Device identification: The devices to which the entities are linked
- Entity identification: ID of the entity that receives the value of the message
- Measurement Type: What kind of measure does the sensor provide? This is quite strictly defined in Home Assistant through sensor types.

Instance of the metadata model

```
{
 "floors": {
   "hasParent": None,
   "single": True,
   "floor_id": {"aliases”: [name,aliases], "areas": ["areas_id"]}
 },
 "areas": {
   "hasParent": [“floors”],
   "single": True,
```

```
  "id": {"aliases":[name,aliases]}
 },
 "measurements":{
  "hasParent": None,
  "single": False,
  "measurementClass":{"aliases":[],"units":[]}   },
 "units":{
  "hasParent": ["measurements"],
  "single": False,
  "measurementClass":{"aliases":[]}   }
}
```

Integration

The integration of the developed artifact into Home Assistant is performed in two phases. The first phase involves the collection and structuring of contextual metadata, as detailed in Section 5.3.3. The second phase focused on the processing of MQTT topic names during message reception, enabling semantic understanding of the data streams. A representative home configuration was created, as illustrated in Figure 5.3.

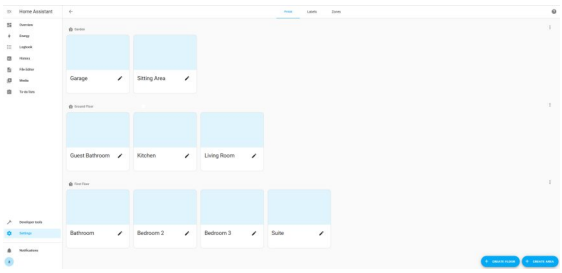

Figure 5.3: Caption

The Home Assistant metadata model is initialized during the startup phase of the MQTT integration component. This initialization process involves hierarchical data retrieval and structuring. Initially, all floors defined in the Home Assistant configuration are retrieved. For each floor, the associated areas (e.g. rooms) are linked accordingly. Subsequently, the relevant areas are categorized into an "areas" group. Finally, additional metadata related to measurements and units is collected and categorized. Once the complete metadata is collected, the model is logically split into sub-models to facilitate efficient context matching. Specifically, two sub-models are constructed: one capturing spatial context (location model) and the other capturing the measurement context (measurement model). The code representing this first phase can be found in the Appendix A.

5.3.4 System overview

To illustrate the role of the proposed artifact within the Home Assistant environment, this section presents an overview of the system architecture. Initially on how the current Home Assistant architecture looks like, and then on where the proposed artifact is fitting within the architecture.

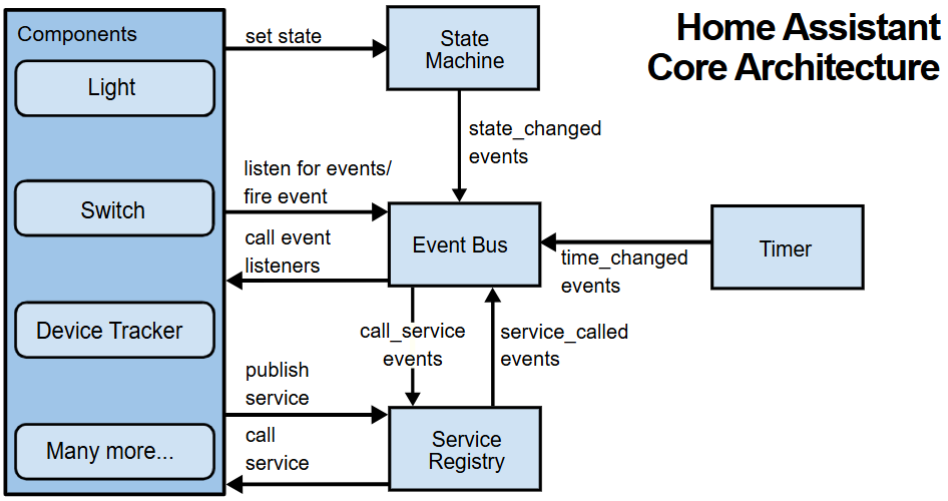

Figure 5.4: Home Assistant Core architecture as adopted from [Hom25b]

The architecture of Home Assistant compromises of distinct layers with the goal to promote modularity, scalability, and separation of concerns, the overview of the general architechture can be seen in [Hom25a]. The Core of Home Assistant is the core runtime environment for all operations, it consists of the state machine, event bus, service registry, and timer, this structure can be seen in Figure 5.4. As extensions to the Core there are integrations. They provide connections to external devices and services and translate external data into the internal representation as needed by the core. Each integration can expose one or more platforms (e.g., sensor, light, switch) and interacts with the event bus and state machine to publish updates or respond to changes, the integration of integrations in Home Assistant is presented in Figure 5.5 . The MQTT integration is an example of this architecture such that it provides a connection to an external MQTT broker, subscribes to relevant topics, and translates received messages into events or state updates which can be processed by the Core.

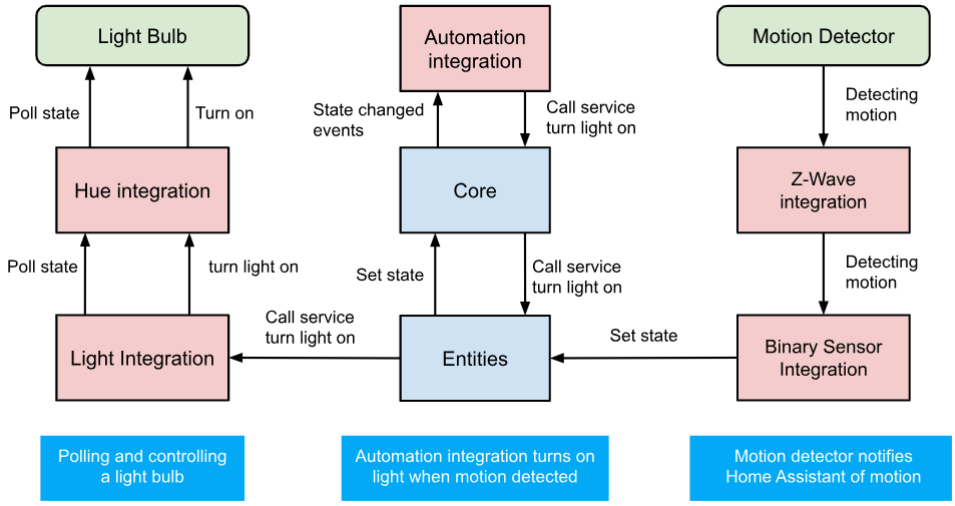

Figure 5.5: Home Assistant Integrations architecture as adopted from [Hom25c]

The proposed artifact will be incorporated into the MQTT Integration as also the documentation notes that it is generally not allowed for integrations that interact with external devices and services to consume the state of entities from other integrations [Hom25d]. The system overview including the artifact can be seen in Figure 5.6. On startup(regular or restart) the MQTT Integration is also initialized. During the initialization of MQTT Integration also the Metadata Model is reinitialized and stored in memory. With each incoming message the MQTT Integration runs the Metadata Processor and stores in memory the matched topic names so that the matching process can be faster in later processing. After this the response is sent to the artifact to be processed further and the regular processing of the message continues.

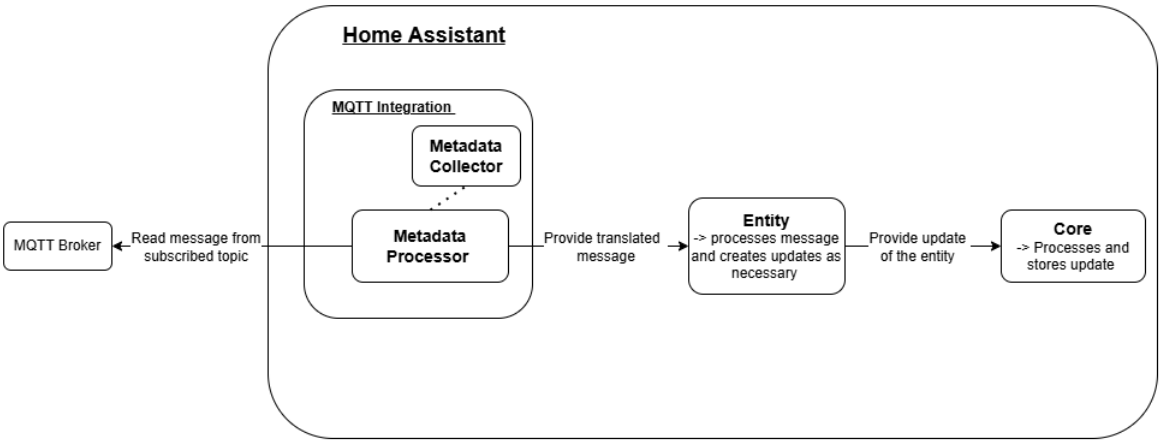

Figure 5.6: Home Assistant system overview with the proposed artifact

5.4 Implementation

After the algorithm was defined and the IoT platform was selected, the next step was to implement the artefact. The implementation was based on the requirements identified in the literature, the design decisions made during algorithm development, and technical constraints imposed by the chosen platform, Home Assistant.

The core implementation of the metadata matching algorithm was developed in Python. This choice was based on the need to integrate with Home Assistant, which is written in Python, as well as by Python's suitability for rapid prototyping and manipulation of hierarchical data structures. The detailed source code of the artefact is available in a public GitHub repository [1].

Although initially implemented with Home Assistant integration in mind, the artefact is designed to be platform independent. Its architecture allows it to be incorporated into any IoT platform, if the host environment supports Python. The artefact functions as a standalone module that receives MQTT topic names as input and returns a structured metadata map based on a provided domain model. The required metadata model can be supplied either through a configuration file or dynamically computed at startup and stored in memory for fast access during runtime. However, as each system and implementation are different, the metadata model generation cannot be abstracted.

From a system integration perspective, the artefact should be placed where MQTT messages are first received in the process, but before they are passed on for further processing or state updates. This would ensure that all components can benefit from

[1] https://github.com/mimarija97/masterThesis

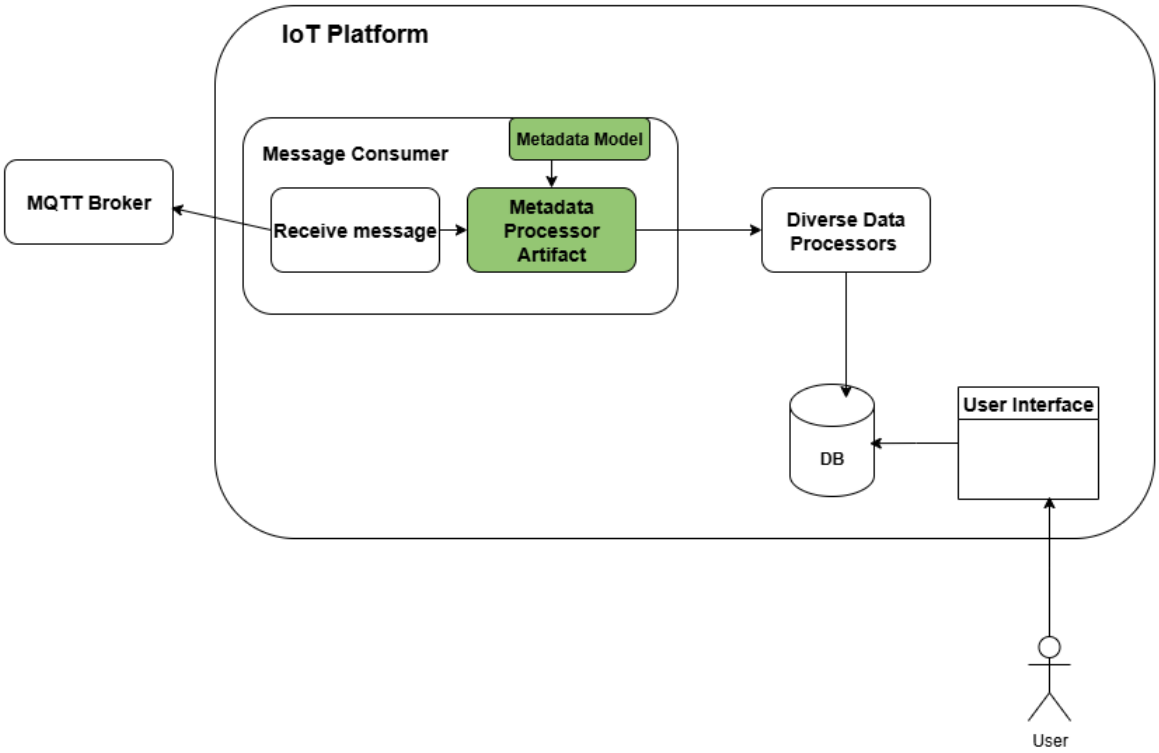

Figure 5.7: Positioning of the Artifact within any IoT Platform

semantically enriched metadata and that there will be no message loss. Visually this can be seen in Figure 5.7.

An example of this integration strategy can be seen in the Home Assistant proof-of-concept implementation. For demonstration purposes, the Home Assistant core repository was cloned and modified to embed the artefact within the MQTT integration module [2]. Due to limitations in Python's default object referencing behavior, the full metadata structure had to be deep-copied for each new topic name to prevent shared state issues, which resulted in increased processing time per message. Nonetheless, this prototype serves as a validation of the artefact's interoperability with real-world platforms.

[2] https://github.com/mimarija97/core

CHAPTER 6

Evaluation

To correctly evaluate the performance and correctness of the artifact, the evaluation will be done on the artifact alone and not on the proof of concept in Home Assistant. Additionally, the base use case will be done based on the metadata model from the previous chapter and the evaluation data below. As the complexity of the algorithm depends on the average number of topic levels, metadata categories, and instances, the performance will be evaluated as well on these factors. Furthermore, the correctness will be assessed as these factors change. The tests used for the evaluations can be found in my Github Repository [1].

6.1 Data for Evaluation

As described in the motivating example in Chapter 1, there is an implementation of a house with two floors and a garden. There are light and temperature sensors in each room, but there are two temperature sensors in the living room. All devices use MQTT. Based on the motivating example, as well as metadata modeled by Home Assistant, basic and extended metadata models were created and linked in the appendix B. Examples of the sensors used for testing, along with the topic names as named based on the basic or extended metadata, are:

- Ground floor living room light sensor – in the basic test with topic name "home/ground_floor/living_room/light" and extended test "home/ground_floor/living_room/lightDevice1/light"

[1] https://github.com/mimarija97/masterThesis

- Ground floor living room section 1 temperature sensor – in the basic test with topic name “home/ground_floor/livingroom/temperature” and extended test “home/ground_floor/livingroom/section1/tempDevice1/temperature”
- Ground floor living room section 2 temperature sensor – in the basic test with topic name “home/ground_floor/livingroom/temperature” and extended test “home/ground_floor/livingroom/section2/temperature”
- Ground floor kitchen humidity sensor – in the basic test with topic name “home/ground_floor/kitchen/humidity” and extended test “home/ground_floor/kitchen/humidityDevice1/humidity”
- Ground floor kitchen temperature sensor – in the basic test with topic name “home/ground_floor/kitchen/temperature” and extended test “home/ground_floor/kitchen/tempDevice2/temperature”
- Ground floor kitchen smoke sensor – “home/ground_floor/kitchen/smoke”
- Ground floor hallway light sensor – in the basic test with topic name “home/ground_floor/hallway/light” and extended test “home/ground_floor/hallway/lightDevice1/light”
- First floor suite temperature sensor – “home/first_floor/suite/temperature”
- First floor bedroom2 motion sensor – in the basic test with topic name “home/firstFloor/bedroom2/motion” and extended test “home/firstFloor/bedroom2/motionSensor1/motion”
- First floor bedroom3 motion alert – in the basic test with topic name “home/firstFloor/bedroom3/motion” and extended test “home/firstFloor/bedroom3/motionSensor2/motion”
- Garden grass moisture sensor - with topic name “home/garden/sittingArea/motion”
- Garden motion sensor – with topic name “home/garden/moisture”
- Garage door contact sensor – in the basic test with topic name “home/garden/garage/garage_door” and extended test “home/garden/garage/garageContactSensor1/garage_door”

6.2 Evaluation Results

In this section, initially, the evaluation of the performance is presented, and then the findings on the correctness of the matching algorithm are presented.

6.2.1 Performance Evaluation

The performance of the implemented artifact (both metadata extraction and integration into the message structure) was evaluated along three primary dimensions: the number of metadata categories, the length and complexity of the topic names, and the number of instances within each metadata category. For each test one message per source was processed and the resulting processing time is the average time per message.

Impact of Metadata Categories

The initial performance was established using a basic metadata model and the 13 simple topic names as described in the previous section. Under these conditions, the system achieved an average processing time of 0.447507 milliseconds per topic. This test case served as the basis for subsequent comparisons.

Next, the evaluation was repeated using the extended metadata model from B, and the simple topic names, same as in the previous evaluation. Processing times increased just little as a result of this test, averaging 0.530769 milliseconds per topic. To extend on this test, the topic names were modified to match the structure and complexity of the extended model, which further increased the average processing time to 0.618615 milliseconds. This test was necessary to check whether there are matches from the lower parts of the metadata tree.

The model was expanded by copying every existing category, adding the suffix "_copy" to each duplicate category, and integrating them as distinct metadata trees in order to test scalability with regard to the amount of metadata categories. The number of categories that needed to be processed was essentially doubled as a result. As anticipated, performance suffered greatly as a result, with the average processing time rising to 2.561784 ms per topic. These findings support the algorithm's theoretical complexity by demonstrating that processing time grows exponentially with the number of metadata categories.

Impact of Topic Name Length

The second round of testing focused on the influence of the topic name length on the performance. The evaluation is done using the extended metadata model, while topic names were constructed to include varying numbers of topic levels. The extended topic names were constructed by adding multiple instances of the metadata categories which are not found only once within the topic name (in the extended metadata model it is the labels category), as well as values which are not foreseen it the metadata model (For example “tuwien”, “ac”, “at”, “master”, and “thesis”.).

In the baseline configuration with five hierarchical levels, the processing time remained at 0.530769 milliseconds. The results, visualized in Figure 6.1, demonstrate a gradual performance degradation as additional levels were added. Specifically, every five additional

levels increased the processing time by approximately 1–2%, reflecting a relatively linear increase in cost associated with topic name parsing.

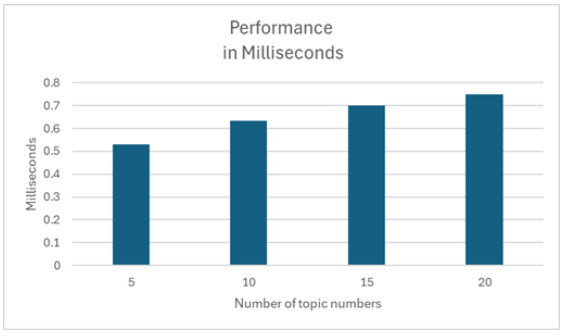

Figure 6.1: Results of evaluation as the size of the topic name increases

Impact of the Number of Instances per Category

Finally, the third performance dimension examined the effect of the number of metadata instances per category on the performance of the artifact. Again, as a baseline, the basic metadata model with the basic topic names was used. The basic metadata model has an average of 22 instances per category. Under this configuration, the average processing time was 0.530769 milliseconds.

To simulate a simplified model, the metadata model was filtered to contain only essential instances, reducing the average number of instances to 6. This optimization improved performance, with the processing time decreasing to 0.354846 milliseconds per topic.

In contrast, to test performance under an increased number of instances, the model was extended with 10 distinct floors and five areas per floor, raising the average to 37 instances per category. Processing times almost doubled as a result of this test, averaging 1.063476 milliseconds per topic.

Effect of Parent-Child Relationships

It is important to note that these tests were conducted using metadata models with fully linked parent-child relationships. As a result, the effective number of instances processed during matching was sometimes lower than the raw count, due to filtering by parent references—particularly for measurement and binary_measurement categories.

Next, the parent-child relationships were removed by setting "has_parent" to None in all categories, to isolate the impact of hierarchical filtering. With the no-parent version of the model containing 37 instances per category, the average processing time increased to 1.706269 milliseconds on average. However, in the two simpler models, removing parent relationships improved performance. For the baseline model without parents, the processing time dropped to 0.411554 milliseconds on average, and in the cleaned-up model with fewer instances, it further decreased to 0.1609 milliseconds on average.

These results show a trade-off: although hierarchical relationships can filter out irrelevant instances by processing fewer elements, they also add overhead since deep copying of Python structures is required. In larger and more complex models, this extra expense becomes substantial and impacts overall performance.

6.2.2 Qualitative Evaluation

Extensive tests were performed to evaluate the accuracy of the matching findings in addition to the system's performance. The purpose of these tests was to confirm whether the metadata extraction and matching procedure generated reliable results in a range of scenarios. Each test had a single message per source, and the results were examined to make sure they contained both the right matches and the appropriate number of matches. The average of each source was used to determine correctness at the conclusion of each test.

A collection of fundamental test cases was used as the starting point for the validation procedure. These included both the basic scenario, consisting of the basic metadata model (each category has only one parent) and basic topic names from Section 6.1, and an extended scenario, which involved the extended metadata model and longer topic names. The metadata structure was linear in both cases, and the system matched the metadata pieces 100% of the time with no false positives. This shows that every match was accurate and that the technology did not mistakenly link irrelevant data points.

To assess the system's resilience to incorrect or partial topic names and metadata models, a number of edge scenarios were looked at in addition to the standard test cases. Topic names with a misspelled or improperly provided metadata class were one example of an edge case. Even if later segments would have matched correctly in a typical scenario, the testing showed that in these cases, the algorithm appropriately discarded the whole subtree that followed the misspelled category. The degree of the mismatch determined the effect of such an inaccuracy. No matches for the matching metadata model were retrieved if the issue occurred at the first level. The system appropriately matched the first-level metadata if the error happened at the second level. False negatives for those deeper parts resulted from the failure to match the subsequent levels.

Topic structures with a parent metadata category including several child items, some of which might be interdependent, constituted another edge case. For example, with Home Assistant, areas correspond to floors, and gadgets can be linked to both floors and areas. Locating a device "outside" without naming a specific location or as part of a floor "garden" is a typical use case. The matching process was examined in various scenarios to make sure it accurately deduced the dependencies and hierarchical linkages. With no false positives or false negatives detected, the system once again produced 100% accuracy when mapping the devices according to the corresponding areas and/or floors.

Finally, as discussed in Section 4.3, another variation was tested where topic names include the metadata category names explicitly before each value. For example, a topic might follow the structure: home/floors/ground_floor/areas/kitchen/binary_measurement/smoke.

This format, while more verbose, was also handled correctly by the system. The system can adapt to various naming conventions without sacrificing accuracy, as evidenced by the matching accuracy maintaining 100%.

Together, these findings show that the matching method is not only very effective but also quite dependable, especially when dealing with topic names that are incomplete, distorted, or have different structures.

6.3 Proof of concept

As proof of concept, the developed artifact is integrated into Home Assistant. As described in Section 5.3 the proof of concept is done by extending the Home Assistant's MQTT integration [2] with the artifact.Limitation of Home Assistant which was found during the evaluation is that does not expose internal timing or processing metrics for its MQTT integration. As a result, it is not possible to directly measure or benchmark the processing time of metadata handling or entity registration within the platform. This limitation restricts the scope of performance evaluation to external measurements, which may not accurately reflect internal processing latency. In addition to this, internal Home Assistant processing might affect the time of the artifact. For these reasons the tests were done on the artifact alone, while on Home Assistant it is only a proof that it can enrich the messages.

A proof of concept was developed using the Home Assistant platform to demonstrate the feasibility of this integration. Based on the model created in the first stage, ten sample MQTT messages were published to various hierarchical topic names that encode both spatial and measurement-related metadata. For example, a message was published to the topic "home/groundfloor/kitchen/smoke", representing if smoke was detected in the kitchen on the ground floor. The artifact processed this message and translated it into an enriched format, reflecting the metadata extracted from the topic structure. The message processing as well as the values reported for the kitchen smoke sensor and the suite temperature sensor are shown in Figures 6.3, 6.2, 6.4, and 6.5.

[2] https://www.home-assistant.io/integrations/mqtt

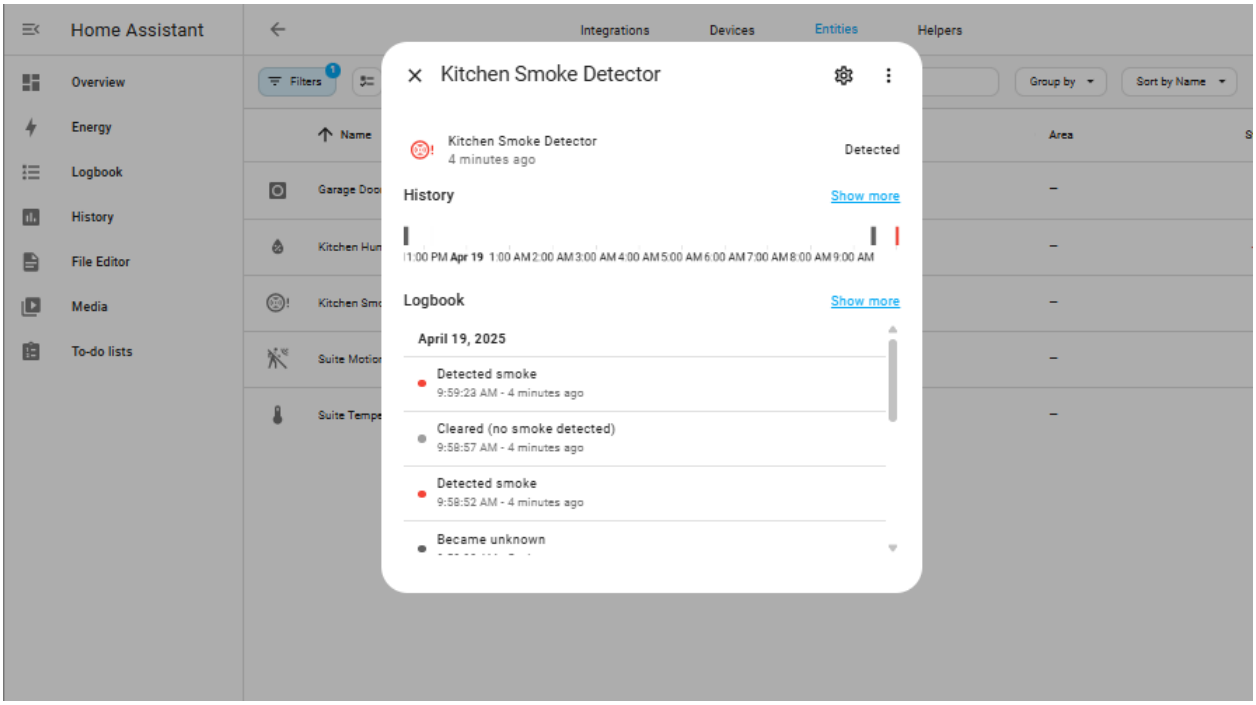

Figure 6.2: Home Assistant values of the smoke detector in the kitchen as received and stored by Home Assistant

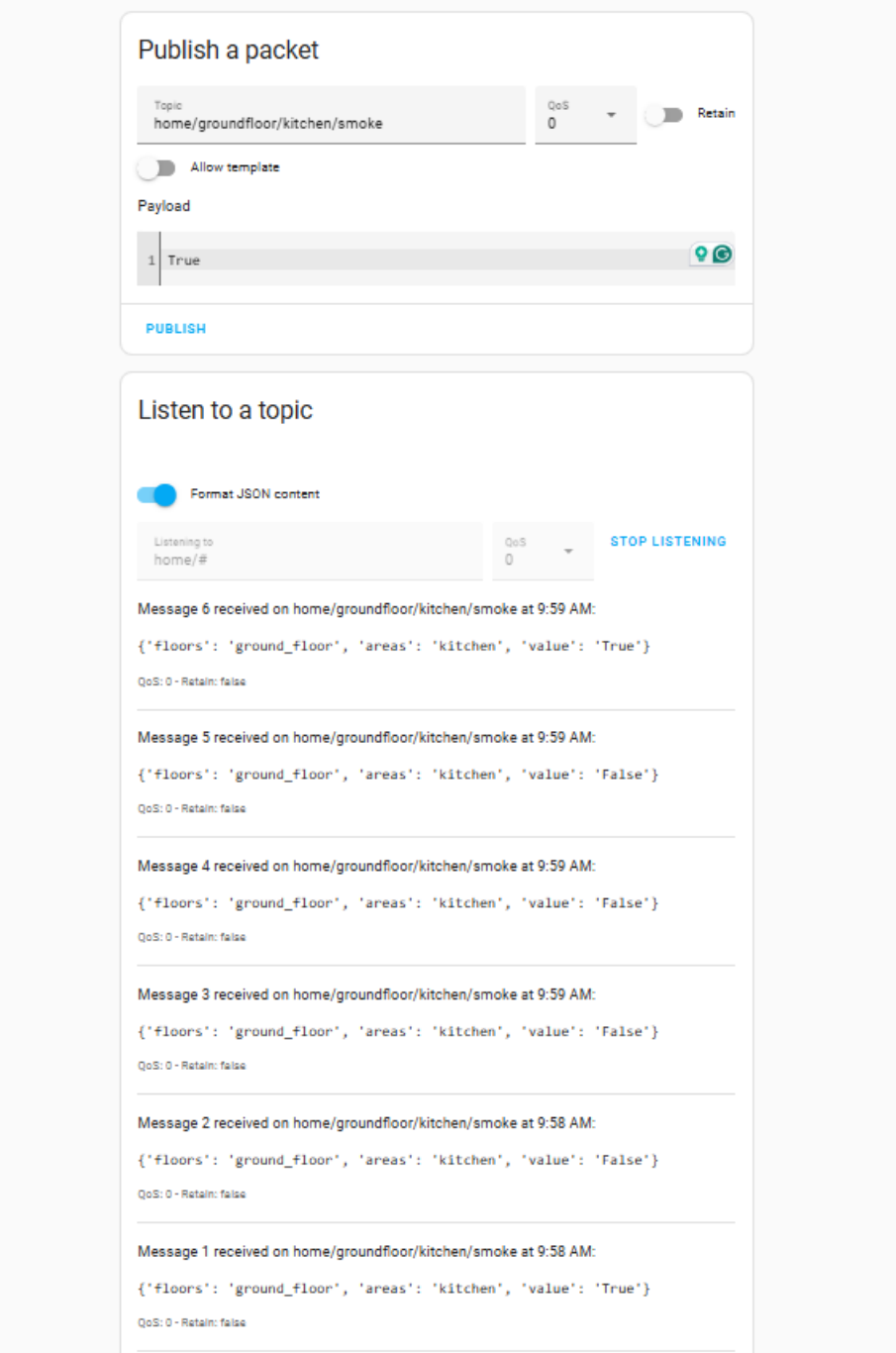

Figure 6.3: Home Assistant messages for the smoke detector in the kitchen

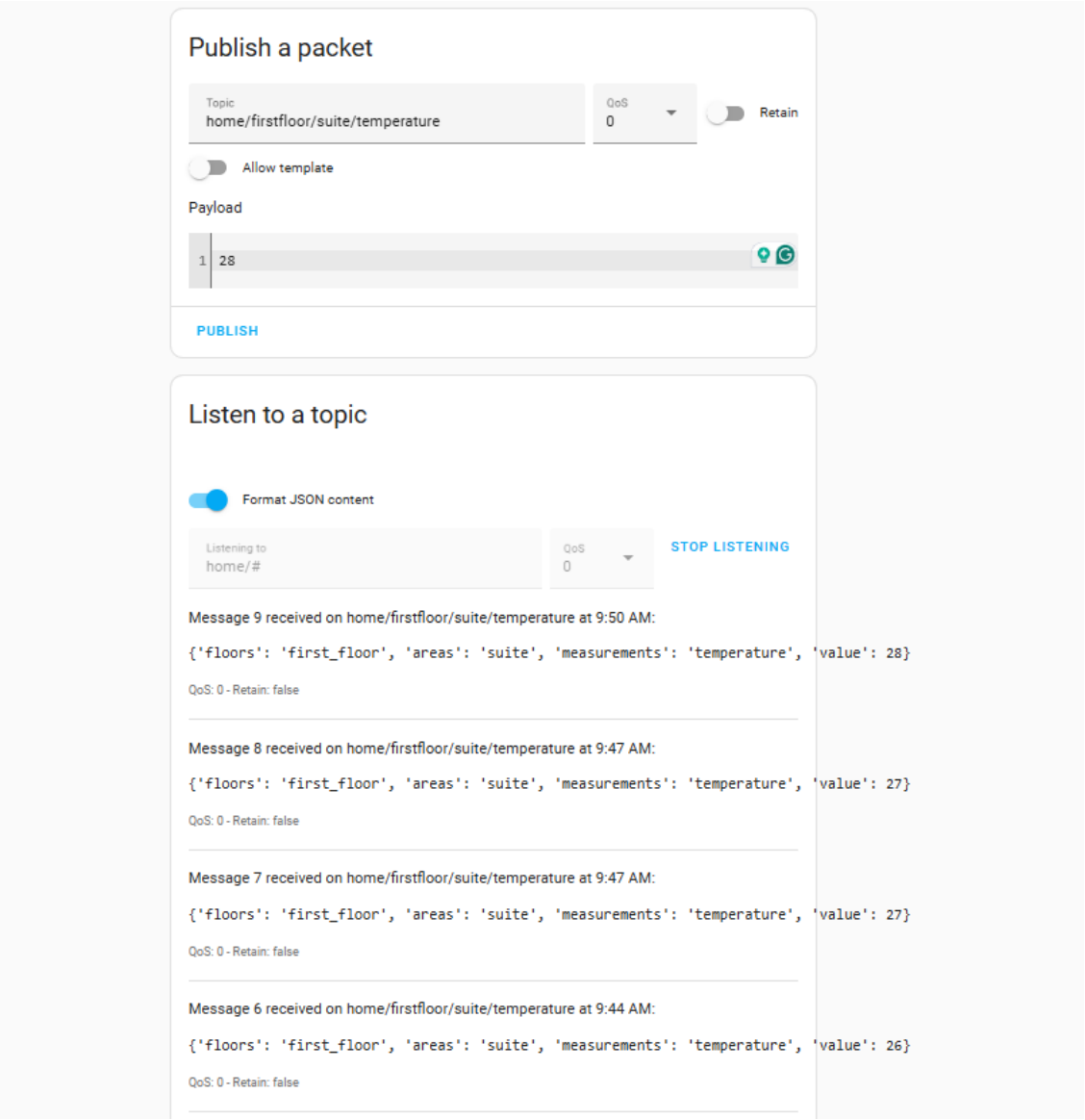

Figure 6.4: Home Assistant messages for measured temperature in the Suite on the first floor

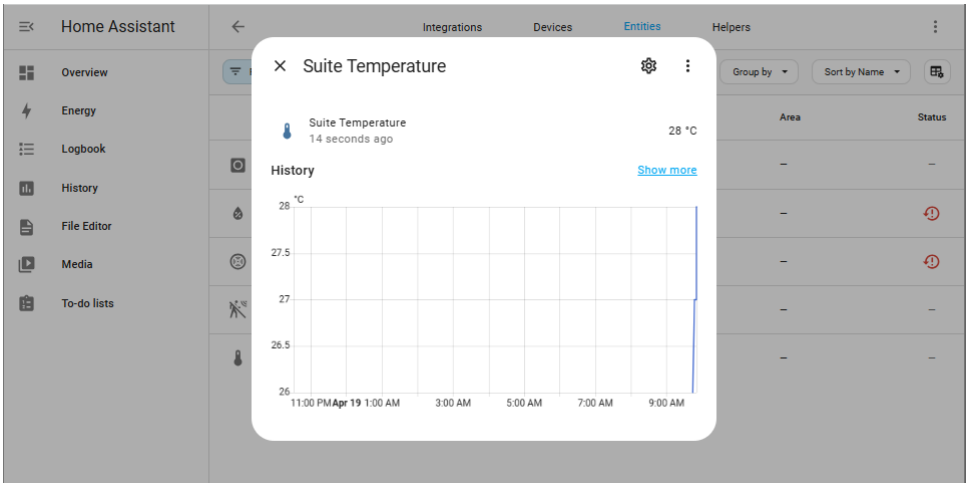

Figure 6.5: Home Assistant stored measured temperature in the Suite on First Floor

CHAPTER 7

Conclusion

In this section, initially, the thesis is summarized, including key contributions, interpretations, and limitations, and in the second part, future work is discussed.

7.1 Summary of the thesis

The goal of this Thesis was to assist IoT and data-as-a-service platforms in context processing by harvesting metadata from the resource name and incorporating it into the message. The research is done in multiple steps. Initially, a Multivocal Literature Research is performed to discover requirements as well as existing implementations in the academic and gray literature. In the second stage, based on the results from the literature review, an artifact is developed and evaluated based on an implementation. Finally, a proof of concept that this artifact can be incorporated into a platform is also implemented, such that the artifact was implemented as an extension to the current MQTT integration of the Home Assistant platform.

The Literature Review showed that there is a lot of missing research in the field of semantically processing resource names. However, it helped to answer the first and second research question. In regards to the first research question - what kind of contextual information can be effectively encoded in MQTT topic names? - it provided us with the understanding that the metadata encoded in the resource name can be either device/sensor related (such as location) or message related (such as message format). Furthermore, the research showed existing research on the model of metadata stored in a topic name and the relationships between the levels of the topic name. Regarding the second research question, what are the key challenges in harmonizing extracted metadata across different MQTT topic naming conventions, the literature review allowed insight into the challenges of harmonizing the processed information into the received message, showing that the biggest challenge is the format of the message.

Finally based on those findings the third research question - How can relevant metadata be accurately and efficiently extracted from MQTT topic names and then systematically structured and integrated into the message payload? - was answered. A model for the metadata storing was created and an artifact for metadata matching was developed using the model. The evaluation of the artifact showed that it matched the topic names perfectly as long as there were no typos or missing links in the layer, e.g., no floor but a room was provided. It further showed that although correctly linking the structure of the metadata, such as which rooms belong to which floors, improves the correctness of the matching algorithm, it shows an inverse relation between performance and the number of instances (e.g., the number of floors and rooms). This points to the conclusion that for smaller models with a lack of overlap of names/aliases of instances, it is better to create a model without a hierarchical structure. It showed that the performance per message was at most millisecond, much below the 10 millisecond restriction in the research question definition. Furthermore, without typos or user mistakes it provided a correctness of 100% which is again above the 95% restriction defined.

Despite the contributions of this thesis, several limitations should be acknowledged. First, while the primary focus has been on the extraction of metadata from the topic name and its integration into the message payload, it lays the groundwork for semantic reasoning with the received data, which is not in the scope of this thesis. The thesis does not explore reasoning mechanisms that could leverage the enriched messages. The second limitation is the protocol supported by the research as well as the artifact, although the initial objective was to ensure protocol-agnostic applicability, due to time constraints, the study was limited to the MQTT protocol. This decision was made based on the assumption that, regardless of the underlying communication protocol, the artifact would receive a resource identifier and a message as input and process the data. The third limitation is that the artifact currently supports only JSON and raw message formats. While the artifact can process the metadata regarding the message format, if specified in the metadata model, this metadata is not used in the harmonization process. The final limitation is that this artifact was created for a smart home environment, while IoT is used in many different environments with many different resources and requirements. While it covers many cases and requirements, there are many in the world of IoT, and not every case is covered in this thesis.

7.2 Future Work

Building upon the limitations identified in this thesis, several directions for future work emerge. Extension of the current work in the future can be investigated in two areas. Firstly, specifically, the artifact could be extended so that it has a more well-rounded function. Secondly, the implementing system can be extended such that it takes full use of the artifact.

Initially, the extensions to the artifact are presented. As IoT is a heterogeneous environment, and it is highly possible that multiple protocols are used within one implementation,

the first extension point would be to test how the artifact works when data is received from different sources. This way, users can get the full advantage of the artifact; otherwise, they would need to process data from different sources in different ways semantically. The heterogeneous IoT environment also means that there are many different message formats used by the devices or even edge nodes. Expanding the support for additional message formats and dynamically processing them based on the metadata provided would increase the flexibility and applicability of the artifact in real-world deployments.

Secondly, we can look into the extensions of the implementing system. The initial step that should be taken is to enable the system to semantically reason with the enriched message, which would help understand how the harmonization of extracted metadata influences the efficiency of the data processing and decision-making in IoT applications. Furthermore, the artifact can be tested as part of another system, specifically a system that deals with another domain - an IoT platform that is not specialized in home automation. This test would help understand how the artifact can be adapted to support various domains or applications beyond the initial implementation.

List of Figures

1.1 A multi-level approach to context extraction and reasoning, adopted from [HFHN24] 3
1.2 Meta model of the MDE Methodology, adopted from [HFHN24] 5
1.3 A literature-based IoT metamodel, as adopted from [Mar23] 8
1.4 CM-IoT Ontology for context modeling as adopted from [PKPV21] 10

3.1 A multi-level view of MDE Methodology. Adopted from [TDOY16] 25

4.1 Topic Structure for Agriculture Implementation, as adopted from [MEBAEK24] 28
4.2 MQTT Topic Tree adopted from van den Bossche et. al. [vdBGV+18] . . 29

5.1 First message Example 45
5.2 Specification on how features of SmartHome Platforms relate to different use cases. Adopted from [SGI+21]. 50
5.3 Caption 53
5.4 Home Assistant Core architecture as adopted from [Hom25b] 54
5.5 Home Assistant Integrations architecture as adopted from [Hom25c] . . . 55
5.6 Home Assistant system overview with the proposed artifact 56
5.7 Positioning of the Artifact within any IoT Platform 57

6.1 Results of evaluation as the size of the topic name increases 62
6.2 Home Assistant values of the smoke detector in the kitchen as received and stored by Home Assistant 65
6.3 Home Assistant messages for the smoke detector in the kitchen 66
6.4 Home Assistant messages for measured temperature in the Suite on the first floor 67
6.5 Home Assistant stored measured temperature in the Suite on First Floor 68

List of Algorithms

B.1 Topic Matching algorithm 94
B.2 Instance Cleanup 95
B.3 Choosing next metadata categories 95

Acronyms

ABAC Attribute-Based Access Control. 27

ALP Application Layer Protocols. 9, 10, 14

CoAP Constrained Application Protocol. 2, 14

DSL Domain Specific Language. 19

HMI Human Machine Interface. 8

IoT Internet of Things. xi, xv, 1, 2, 8, 10, 13, 14, 32, 47

MDE Model-Driven Engineering. xi, xiii, xv, 21, 24, 25, 41

MOF Meta Object Facility. 18

MQTT Message Queuing Telemetry Transport. iii, xi, xiii, xv, 2, 10, 11, 14–16, 23, 24, 28–41, 48, 51–57, 59, 64, 69, 70, 73, 93

OWL Web Ontology Language. 9, 42, 43

QoS Quality of Service. 15

TCP Transmission Control Protocol. 2

Bibliography

[AHDM22] Fatima Zahra Amara, Mounir Hemam, Meriem Djezzar, and Moufida Maimor. Semantic web and internet of things: Challenges, applications and perspectives. *Journal of ICT Standardization*, 10(2):261–291, 2022.

[Alm10] Christian Almazan. *Rover: Architectural Support for Exposing and Using Context*. PhD thesis, University of Maryland, College Park, 2010.

[Ama21] Amazon Web Services. Designing mqtt topics for aws iot core. Technical report, Amazon Web Services, dec 2021. Accessed: 2025-03-29.

[Ana16] Christos Anagnostopoulos. Intelligent contextual information collection in internet of things. *International journal of wireless information networks*, 23(1):28–39, 2016.

[APS19] Charilaos Akasiadis, Vassilis Pitsilis, and Constantine D. Spyropoulos. A multi-protocol iot platform based on open-source frameworks. *Sensors (Basel, Switzerland)*, 19(19):4217, 2019.

[ASH16] R. Adams, P. Smart, and A. S. Huff. Shades of grey: guidelines for working with the grey literature in systematic reviews for management and organizational studies. *International Journal of Management Reviews*, 19:432–454, 2016.

[AZ19] Zahrah A Almusaylim and Noor Zaman. A review on smart home present state and challenges: linked to context-awareness internet of things (IoT). *Wirel. Netw.*, 25(6):3193–3204, August 2019.

[Bat19] Maximilian Batz. Mqtt topic tree design best practices, tips & examples. `https://pi3g.com/mqtt-topic-tree-design-best-practices-tips-examples/`, May 2019. Accessed: 2025-03-11.

[BBH+10] Claudio Bettini, Oliver Brdiczka, Karen Henricksen, Jadwiga Indulska, Daniela Nicklas, Anand Ranganathan, and Daniele Riboni. A survey of context modelling and reasoning techniques. *Pervasive and mobile computing*, 6(2):161–180, 2010.

[BGG19] Emmanuel Bruno, Romane Gallier, and Alban Gabillon. Enforcing access controls in iot networks. In Tran Khanh Dang, Josef Küng, Makoto Takizawa, and Son Ha Bui, editors, *Future Data and Security Engineering*, pages 429–445, Cham, 2019. Springer International Publishing.

[Bra23] Lukas Brandl. Mqtt topic tree & topic matching: Challenges and best practices explained. https://www.hivemq.com/blog/mqtt-topic-tree-matching-challenges-best-practices-explained/, mar 2023. Accessed: 2025-03-20.

[Com20] Home Assistant Community. How do you name your mqtt topics? https://community.home-assistant.io/t/how-do-you-name-your-mqtt-topics/169634, February 2020. Accessed: 2025-02-15.

[Com24] Tasmota Community. Commands over mqtt. https://tasmota.github.io/docs/MQTT/#commands-over-mqtt, 2024. Accessed: 2025-04-18.

[CuRK23] Jonathan Cook, Sabih ur Rehman, and M. Arif Khan. Security and privacy for low power iot devices on 5g and beyond networks: Challenges and future directions. *IEEE Access*, 11:39295–39317, 2023.

[Dey01] A. K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5:4–7, 2001.

[Dil21] Sam Dillard. Mqtt topic and payload parsing with telegraf. https://www.influxdata.com/blog/mqtt-topic-payload-parsing-telegraf/, December 2021. Accessed: 2025-04-11.

[dMAT+15] Everton de Matos, Leonardo A. Amaral, Ramão Tiburski, Willian Lunardi, Fabiano Hessel, and Sabrina Marczak. Context-aware system for information services provision in the internet of things. In *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–4, 2015.

[DMM19] Radu Dobrescu, Daniel Merezeanu, and Stefan Mocanu. Context-aware control and monitoring system with iot and cloud support. *Computers and Electronics in Agriculture*, 160:91–99, 2019.

[DRD+14] Keling Da, Philippe Roose, Marc Dalmau, Joseba Nevado, and Riadh Karchoud. Kali2much: a context middleware for autonomic adaptation-driven platform. In *Proceedings of the 1st ACM Workshop on Middleware for Context-Aware Applications in the IoT*, M4IOT '14, page 25–30, New York, NY, USA, 2014. Association for Computing Machinery.

[EGH24] Yahya El Gaoual and Mohamed2024 Hanine. Critical overview of model driven engineering. In Mohamed Ben Ahmed, Anouar Abdelhakim Boudhir, Rani El Meouche, and İsmail Rakıp Karaş, editors, *Innovations in Smart Cities Applications Volume 7*, pages 87–97, Cham, 2024. Springer Nature Switzerland.

[FJSGK22] Norisvaldo Ferraz Junior, Anderson A.A. Silva, Adilson E. Guelfi, and Sergio T. Kofuji. Performance evaluation of publish-subscribe systems in iot using energy-efficient and context-aware secure messages. *Journal of Cloud Computing*, 11(1), January 2022.

[FP11] Martin Fowler and Rebecca Parsons. *Domain-specific languages*. The Addison-Wesley signature series. Addison-Wesley, Upper Saddle River, NJ, 2011.

[GBKN23] Dimitris Gkoulis, Cleopatra Bardaki, George Kousiouris, and Mara Nikolaidou. Transforming iot events to meaningful business events on the edge: Implementation for smart farming application. *Future Internet*, 15(4), 2023.

[GGB20] Alban Gabillon, Romane Gallier, and Emmanuel Bruno. Access controls for iot networks. *SN computer science*, 1(1):24, 2020.

[GIC20] Dimitrios Glaroudis, Athanasios Iossifides, and Periklis Chatzimisios. Survey, comparison and research challenges of iot application protocols for smart farming. *Computer Networks*, 168:107037, 2020.

[GPP+18] Maria Ganzha, Marcin Paprzycki, Wiesław Pawłowski, Paweł Szmeja, and Katarzyna Wasielewska. Towards semantic interoperability between internet of things platforms. In *Integration, Interconnection, and Interoperability of IoT Systems*, Internet of Things. Springer International Publishing AG, Switzerland, 2018.

[GT23] Rustam Gamzayev and Mykola Tkachuk. Development of problem-specific modeling language to support software variability in "smart home" systems. *Sučasnij stan naukovih doslìdžen' ta tehnologìj v promislovostì (Online)*, 1 (23):45–56, 2023.

[HDZ+20] Kai Hwang, Meijiao Duan, Yanmei Zhang, Sheng Gao, Jianming Zhu, Peng Liu, Fu Chen, and Youwei Wang. Improving topic-based data exchanges among iot devices. *Security and communication networks*, 2020(2020):1–14, 2020.

[HFHN24] Amal Hallou, Tarik Fissaa, Hatim Hafiddi, and Mahmoud Nassar. Context-aware iot system development approach based on meta-modeling and reinforcement learning: A smart home case study. *International Journal*

of Online and Biomedical Engineering (iJOE), 20(06):pp. 25–42, Apr. 2024.

[Hiv24] HiveMQ Team. MQTT Topics, Wildcards, & Best Practices – MQTT Essentials: Part 5. https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/, feb 2024. Accessed: 2025-03-20.

[HKN+05] N. Honle, U.-P. Kappeler, D. Nicklas, T. Schwarz, and M. Grossmann. Benefits of integrating meta data into a context model. In *Third IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 25–29. IEEE, 2005.

[HKS17] Marina Harlamova, Marite Kirikova, and Kurt Sandkuhl. A survey on challenges of semantics application in the internet of things domain. *Applied Computer Systems*, 21, 01 2017.

[Hom24] Homie. Homie: An mqtt convention for iot/m2m. https://github.com/homieiot/convention/blob/develop/convention.md#topic-ids, December 2024. Accessed: 2025-02-15.

[Hom25a] Home Assistant. Architecture overview — home assistant developer docs. https://developers.home-assistant.io/docs/architecture_index/, July 2025. Accessed 2025-07-27.

[Hom25b] Home Assistant. Core architecture — home assistant developer docs. https://developers.home-assistant.io/docs/architecture/core/, July 2025. Accessed 2025-07-27.

[Hom25c] Home Assistant. Integration architecture — home assistant developer docs. https://developers.home-assistant.io/docs/architecture_components/, July 2025. Accessed 2025-07-27.

[Hom25d] Home Assistant. Mqtt — home assistant integration. https://www.home-assistant.io/integrations/mqtt, August 2025. Accessed 2025-07-24.

[JLW+20] Yang Jiao, Jiacheng Li, Jiaman Wu, Dezhi Hong, Rajesh Gupta, and Jingbo Shang. SeNsER: Learning cross-building sensor metadata tagger. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 950–960, Online, November 2020. Association for Computational Linguistics.

[JMHL19] Pouya Jamborsalamati, Mojtaba Moghimi, Jahangir Hossain, and Junwei Lu. Design and implementation of a hierarchical hybrid communication platform for multi-microgrid applications. In *Sustainability in Energy and*

Buildings 2018, volume 131 of *Smart Innovation, Systems and Technologies*, pages 199–208. Springer International Publishing AG, Switzerland, 2019.

[Lab24] OPC Labs. Opc ua pubsub topic tree. https://opclabs.doc-that.com/files/onlinedocs/OPCLabs-OpcStudio/Latest/User's%20Guide%20and%20Reference-OPC%20Studio/OPC%20UA%20PubSub%20Topic%20Tree.html, 2024. Accessed: 2025-03-29.

[Lin19] Cirrus Link. Sparkplug™ specification. https://sparkplug.eclipse.org/specification/version/2.2/documents/sparkplug-specification-2.2.pdf, 10-11-2019. [Accessed 17-11-2024].

[Lud03] Jochen Ludewig. Models in software engineering ? an introduction. *Software and Systems Modeling*, 2(1):5–14, March 2003.

[Mar23] Schnellmann Marianne. Internet of things open source plattformen: Eine modellierungsmethode zur abstraktion von iot-umgebungen. Master's thesis, University of Vienna, 2023.

[MEBAEK24] Basma M. Mohammad El-Basioni and Sherine M. Abd El-Kader. Designing and modeling an iot-based software system for land suitability assessment use case. *Environmental monitoring and assessment*, 196(4):380–380, 2024.

[MLJ+18] F. Montori, K. Liao, P. P. Jayaraman, L. Bononi, T. Sellis, and D. Georgakopoulos. Classification and annotation of open internet of things datastreams. *Lecture Notes in Computer Science*, pages 209–224, 2018.

[mqt] MQTT Topic Trees - Raspberry Valley. https://raspberry-valley.azurewebsites.net/MQTT-Topic-Trees/. Accessed: 2025-03-29.

[MV24] Behnaz Motamedi and Balázs Villányi. A reliable publish–subscribe mechanism for internet of things-enabled smart greenhouses. *Applied Sciences*, 14(15):6407, 2024.

[NAG18] M. Noura, M. Atiquzzaman, and M. Gaedke. Interoperability in internet of things: taxonomies and open challenges. *Mobile Networks and Applications*, 24:796–809, 2018.

[OAS19] OASIS. MQTT Version 5.0 Specification. https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html, note = Accessed: 2024-12-20, March 2019. OASIS Standard.

[OPC24] OPC Foundation. Opc ua part 14: Pubsub - 7.3.5 mqtt. https://reference.opcfoundation.org/Core/Part14/v105/docs/7.3.5, 2024. Accessed: 2025-03-29.

[PCV24] PCVue Solutions. Mqtt topic naming convention. https://www.pcvue.com/ProductHelp/PcVue/en/Content/Extras/mqtt_topic.php, 2024. Accessed: 2025-04-18.

[PDTS24] Konstantinos Panayiotou, Constantine Doumanidis, Emmanouil Tsardoulias, and Andreas L. Symeonidis. Smauto: A domain-specific-language for application development in smart environments. *Pervasive and Mobile Computing*, 101:101931, 2024.

[Pet19] Christos Petropoulos. A detailed guide to the world of mqtt. https://hackernoon.com/a-detailed-guide-to-the-world-of-mqtt-bo1d63cay, aug 2019. Accessed: 2025-03-20.

[PKPV21] Preeja Pradeep, Shivsubramani Krishnamoorthy, Rahul Krishnan Pathinarupothi, and Athanasios V. Vasilakos. Leveraging context-awareness for internet of things ecosystem: Representation, organization, and management of context. *Computer Communications*, 177:33–50, 2021.

[PMK22] Tobias Christian Piller, David Maria Merz, and Abdelmajid Khelil. Mqtt-4est: Rule-based web editor for semantic-aware topic naming in mqtt. In *2022 IEEE 19th Annual Consumer Communications Networking Conference (CCNC)*, pages 1–8, 2022.

[Pon] Ponlakshmi. Getting to know mqtt topics & its basics. https://www.bevywise.com/blog/mqtt-topics/. Accessed: 2025-03-20.

[PZCG14] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Context aware computing for the internet of things: A survey. *IEEE Communications surveys and tutorials*, 16(1):414–454, 2014.

[Pé12] Xose Pérez. Mqtt topic naming convention. https://tinkerman.cat/post/mqtt-topic-naming-convention, dec 2012. Accessed: 2025-03-30.

[RMD+06] Vincent Ricquebourg, David Menga, David Durand, Bruno Marhic, Laurent Delahoche, and Christophe Loge. The smart home concept : our immediate future. In *2006 1ST IEEE International Conference on E-Learning in Industrial Electronics*, pages 23–28, 2006.

[RMG20] Ahlem Rhayem, Mohamed Ben Ahmed Mhiri, and Faiez Gargouri. Semantic web technologies for the internet of things: Systematic literature review. *Internet of Things*, 11:100206, 2020.

[Rod15] Alberto Rodrigues da Silva. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures*, 43:139–155, 2015.

[SAF22] SAF Tehnika. Aranet mqtt functionality and integration with microsoft azure. https://management.saftehnika.com/pub/Aranet_MQTT_functionality_and_integration_with_Microsoft_Azure.pdf, feb 2022. Accessed: 2025-03-29.

[SGI+21] Brian Setz, Sebastian Graef, Desislava Ivanova, Alexander Tiessen, and Marco Aiello. A comparison of open-source home automation systems. *IEEE Access*, 9:167332–167352, 2021.

[SRC18] David Sembroiz, Sergio Ricciardi, and Davide Careglio. Chapter 10 - a novel cloud-based iot architecture for smart building automation. In Massimo Ficco and Francesco Palmieri, editors, *Security and Resilience in Intelligent Data-Centric Systems and Communication Networks*, Intelligent Data-Centric Systems, pages 215–233. Academic Press, 2018.

[SS24] Renu Sharma and Anil Sharma. Iot interoperability framework for smart home: Mda-inspired approach. *Cluster computing*, 27(5):6305–6322, 2024.

[SSdCG24] E. Saavedra, A. Santamaria, G. del Campo, and I. Gomez. Leveraging iot harmonization: an efficacious nb-iot relay for integrating 6lowpan devices into legacy ipv4 networks. *Applied Sciences*, 14:3411, 2024.

[STC+23] R. Shiina, S. Tamaki, H. Che, T. Hatano, T. Yamada, T. Shimada, and T. Taniguchi. Proposal for low-layer metadata collection technology to enhance iot metadata utilization. *IEEE Access*, 11:11078–11088, 2023.

[Ste24] Steve. Mqtt topic and payload design. http://www.steves-internet-guide.com/mqtt-topic-payload-design-notes/, may 2024. Accessed: 2025-03-29.

[TDOY16] Okan Topçu, Umut Durak, Halit Oğuztüzün, and Levent Yilmaz. *Model Driven Engineering*, pages 23–38. Springer International Publishing, Cham, 2016.

[TOH+16] Nasi Tantitharanukul, Kitisak Osathanunkul, Kittikorn Hantrakul, Part Pramokchon, and Paween Khoenkaw. Mqtt-topic naming criteria of open data for smart cities. In *2016 International Computer Science and Engineering Conference (ICSEC)*, pages 1–6, 2016.

[Vai24] Vaisala. *CL51 User Guide: MQTT Topics*, 2024. Accessed: 2025-03-29.

[vdBGV+18] Adrien van den Bossche, Nicolas Gonzalez, Thierry Val, Damien Brulin, Frédéric Vella, Nadine Vigouroux, and Eric Campo. Specifying an mqtt tree for a connected smart home. In Mounir Mokhtari, Bessam Abdulrazak, and Hamdi Aloulou, editors, *Smart Homes and Health Telematics, Designing a Better Future: Urban Assisted Living*, pages 236–246, Cham, 2018. Springer International Publishing.

[WBK+18] Weimin Wang, Michael R. Brambley, Woohyun Kim, Sriram Somasundaram, and Andrew J. Stevens. Automated point mapping for building control systems: Recent advances and future research needs. *Automation in Construction*, 85:107–123, 2018.

[WDD+22] Guangxi Wan, Xiaoting Dong, Qingwei Dong, Yunpeng He, and Peng Zeng. Context-aware scheduling and control architecture for cyber-physical production systems. *Journal of Manufacturing Systems*, 62:550–560, 2022.

[Wei99] Mark Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):3–11, jul 1999.

[WSS21] David Waterworth, Subbu Sethuvenkatraman, and Quan Z. Sheng. Advancing smart building readiness: Automated metadata extraction using neural language processing methods. *Advances in Applied Energy*, 3:100041, 2021.

APPENDIX A

Metadata Collection Home Assistant

```
def get_areas(hass: HomeAssistant):
    """Retrieve all areas."""
    area_hass = ar.async_get(hass)
    area_reg = area_hass.async_list_areas()
    _LOGGER.info("returned areas: " + str(area_reg))

    result_dictionary = {"has_parent": ["floors"], "single": True}
    result_dictionary.update(
        {area.id: {"aliases": [area.name, *area.aliases]} for area in area_reg}
    )
    _LOGGER.info("result areas: " + str(result_dictionary))
    return result_dictionary


def get_floors(hass: HomeAssistant):
    """Retrieve all floors."""
    floors_hass = fr.async_get(hass)
    floor_reg = floors_hass.async_list_floors()

    result_dictionary = defaultdict(list)
    result_dictionary.update({"has_parent": None, "single": True})
    result_dictionary.update(
        {
            floor.floor_id: {
                "aliases": [floor.name,floor.normalized_name,str(floor.level), *floor.aliases]
            }
            for floor in floor_reg
        }
    )

    area_registry = ar.async_get(hass)

    for floor_id, floor in result_dictionary.items():
        areas_on_floor = ar.async_entries_for_floor(area_registry, floor_id=floor_id)
        area_keys = [area.id for area in areas_on_floor]
        if isinstance(floor, dict):
```

```
                floor["areas"] = area_keys

        return result_dictionary


    def get_measurement_types():
        """Retrieve all measurement types."""
        measurement_types = defaultdict(list)
        measurement_types.update({"has_parent": None, "single": True})
        for key, value in DEVICE_CLASS_UNITS.items():
            measurement_types.update({key.value: {"aliases": []}})
        return measurement_types


    def get_labels(hass: HomeAssistant):
        """Get the labels defined by the user."""
        label_reg = lr.async_get(hass)
        result_dictionary = {"has_parent": "floors", "single": True}
        result_dictionary.update(
            {
                label.label_id: {"aliases": [label.name]}
                for label in label_reg.async_list_labels()
            }
        )
        return result_dictionary

    # preprocessing metadata
    def filter_metadata(metadata, match_key):
        subtree = {}

        def dfs(current_key):
            if current_key not in metadata or current_key in subtree:
                return
            subtree[current_key] = metadata[current_key]
            for key, value in metadata.items():
                parents = value.get("has_parent")
                if isinstance(parents, list) and current_key in parents:
                    dfs(key)
        dfs(match_key)
        return subtree


    def preprocess_metadata_into_subtrees(metadata_dictionary):
        grouped_metadata = []
        for groupName, group in metadata_dictionary.items():
            if group["has_parent"] is not None:
                continue
            grouped_metadata.append(filter_metadata(metadata_dictionary, groupName))
        return grouped_metadata


    def get_metadata_dictionary(hass: HomeAssistant):
        """Create the metadata dictionary by which the topic name will be processed."""
        result = {}
        result.update({"floors": get_floors(hass)})
        result.update({"areas": get_areas(hass)})
        result.update({"labels": get_labels(hass)})
        result.update({"measurements": get_measurement_types()})
        return preprocess_metadata_into_subtrees(result)
```

APPENDIX B

Metadata Home Assistant Test Model

B.1 Initial basic metadata

```
{
 "floors": {"has_parent": null,
            "single": true,
            "groundFloor": {
                "aliases": ["ground_floor","groundfloor"],
                "areas": ["livingroom", "kitchen", "hallway", "bathroom1"]
            },
            "firstFloor": {
                "aliases": ["first_floor","firstfloor"],
                "areas": ["suite", "bedroom2", "bedroom3", "bathroom2"]
            },
            "garden": {
                "aliases": [],
                "areas": ["garage", "sittingArea"]
            }
            },
 "areas": {
     "has_parent": ["floors"],
     "single": true,
     "bathroom": { "aliases": [] },
     "kitchen": {
         "aliases": [],
         "measurement":["humidity", "temperature"],
         "binary_measurement":["smoke"]
     },
     "livingroom": {
         "aliases": ["living_room"],
         "measurement":["light", "temperature"],
         "binary_measurement":[]
     },
     "hallway": {
         "aliases": [],
```

```
            "measurement":["light"],
            "binary_measurement":[]
        },
        "suite": {
            "aliases": ["bedroom1"],
            "measurement":["temperature"],
            "binary_measurement":[]
        },
        "bedroom2": {
            "aliases": ["bedroom2"],
            "measurement":[],
            "binary_measurement":["motion"]
        },
        "bedroom3": {
            "aliases": ["bedroom3"],
            "measurement":[],
            "binary_measurement":["motion"]
        },
        "bathroom2": { "aliases": [] },
        "garage": {
            "aliases": [],
            "measurement":[],
            "binary_measurement":["garage_door"]
        },
        "sittingArea": {
            "aliases": [],
            "measurement":[],
            "binary_measurement":["motion"]
        }
    },
    "measurement": {
        "has_parent": ["floors","areas"],
        "single": true,
        "apparent_power": { "aliases": [] },
        "aqi": { "aliases": [] },
        "area": { "aliases": [] },
        "atmospheric_pressure": { "aliases": [] },
        "battery": { "aliases": [] },
        "blood_glucose_concentration": { "aliases": [] },
        "co": { "aliases": [] },
        "co2": { "aliases": [] },
        "conductivity": { "aliases": [] },
        "current": { "aliases": [] },
        "data_rate": { "aliases": [] },
        "data_size": { "aliases": [] },
        "distance": { "aliases": [] },
        "duration": { "aliases": [] },
        "energy": { "aliases": [] },
        "energy_distance": { "aliases": [] },
        "energy_storage": { "aliases": [] },
        "frequency": { "aliases": [] },
        "gas": { "aliases": [] },
        "humidity": { "aliases": [] },
        "illuminance": { "aliases": [] },
        "irradiance": { "aliases": [] },
        "moisture": { "aliases": [] },
        "nitrogen_dioxide": { "aliases": [] },
        "nitrogen_monoxide": { "aliases": [] },
        "nitrous_oxide": { "aliases": [] },
        "ozone": { "aliases": [] },
        "ph": { "aliases": [] },
        "pm1": { "aliases": [] },
```

```
        "pm10": { "aliases": [] },
        "pm25": { "aliases": [] },
        "power_factor": { "aliases": [] },
        "power": { "aliases": [] },
        "precipitation": { "aliases": [] },
        "precipitation_intensity": { "aliases": [] },
        "pressure": { "aliases": [] },
        "reactive_power": { "aliases": [] },
        "signal_strength": { "aliases": [] },
        "sound_pressure": { "aliases": [] },
        "speed": { "aliases": [] },
        "sulphur_dioxide": { "aliases": [] },
        "temperature": { "aliases": [] },
        "volatile_organic_compounds": { "aliases": [] },
        "volatile_organic_compounds_parts": { "aliases": [] },
        "voltage": { "aliases": [] },
        "volume": { "aliases": [] },
        "volume_flow_rate": { "aliases": [] },
        "volume_storage": { "aliases": [] },
        "water": { "aliases": [] },
        "weight": { "aliases": [] },
        "wind_direction": { "aliases": [] },
        "wind_speed": { "aliases": [] }
    },
    "binary_measurement": {
        "has_parent": ["floors","areas"],
        "single": true,
        "battery_charging": { "aliases": [] },
        "cold": { "aliases": [] },
        "connectivity": { "aliases": [] },
        "door": { "aliases": [] },
        "garage_door": { "aliases": [] },
        "heat": { "aliases": [] },
        "light": { "aliases": [] },
        "lock": { "aliases": [] },
        "motion": { "aliases": [] },
        "moving": { "aliases": [] },
        "occupancy": { "aliases": [] },
        "opening": { "aliases": [] },
        "plug": { "aliases": [] },
        "presence": { "aliases": [] },
        "problem": { "aliases": [] },
        "running": { "aliases": [] },
        "safety": { "aliases": [] },
        "smoke": { "aliases": [] },
        "sound": { "aliases": [] },
        "tamper": { "aliases": [] },
        "update": { "aliases": [] },
        "vibration": { "aliases": [] },
        "window": { "aliases": [] }
    }
}
```

B.2 Extension to the metadata

To test different cases, the basic metadata was extended with new metadata categories, and the links of the existing metadata instances were extended. The new categories are:

```
{
```

```
    "labels":{
        "has_parent": ["areas"],
        "single": False,
        "area1": {
            "aliases": []
        },"area2": {
            "aliases": []
        },"security": {
            "aliases": []
        },"energy": {
            "aliases": ["energy-saving"]
        },"binary": {
            "aliases": []
        },"automations": {
            "aliases": []
        },"health": {
            "aliases": []
        },"ambience": {
            "aliases": []
        }
    },
    "device":{
        "has_parent": ["floors","areas"],
        "single": False,
        "tempDevice1":{
                "aliases": []
        },
        "tempDevice2":{
                "aliases": []
        },
        "lightDevice1":{
                "aliases": []
        },
        "humidityDevice1":{
                "aliases": []
        },
        "garageContactDevice1":{
                "aliases": [],
                "sensor": ["garageContactSensor1"]
        }
        },
    "sensor":{
        "has_parent": ["device","areas","labels"],
        "single": False,
         "garageContactSensor1":{
                "aliases": []
        },
         "motionSensor1":{
                "aliases": []
        },
         "motionSensor2":{
                "aliases": []
        }
    }
}
```


In addition to this, the new categories and their instances are linked in the existing instances such that the living room, kitchen, and hallway on the ground floor link to the devices, the garage points to the garageContactDevice1 device, and the bedrooms on the second floor point to the motionSensors.

B.3 Home Assistant metadata collection code

In this section of the appendix the code which collects the metadata from an instance of Home Assistant is presented. This is done on each startup/restart of the instance. The full changes of the MQTT integration can be seen in my Github Repository which was extracted from the Core Repository of Home Assistant.[1]

[1] https://github.com/mimarija97/core

Algorithm B.1: Topic Matching algorithm

```
1  Function MATCH_TOPIC(topicName, metadataCategoriesList):
2      topicPartsList = topicName.split;
3      result = {};
4      foreach topicPart ∈ topicPartsList do
5          nextMetadataCategoriesToCheck =
            FIND_NEXT_CATEGORIES(metadataCategoriesList);
6          foreach metadataCategory ∈ nextMetadataCategoriesToCheck do
7              metadataCategory =
                CLEANUP_INSTANCE(metadataCategory, result);
8              match = MATCH_CATEGORY(topicPart, metadataCategory);
9              if match ≠ NULL then
10                 result.update(match);
11                 if metadataCategory.single = True then
12                     metadataCategoriesList.remove(metadataCategory)
13                 end
14             end
15         end
16     end
17     return result
18 Function MATCH_CATEGORY(topicPart, metadataCategory):
19     foreach instance ∈ metadataCategory do
20         result = MATCH_INSTANCE(topicPart, instance);
21         if result ≠ NULL then
22             return (metadataCategory.ID,result);
23         end
24     end
25     return NULL
26 Function MATCH_INSTANCE(topicPart, instance):
27     if instance.ID = topicPart then
28         return instance;
29     end
30     foreach alias ∈ instance do
31         if alias == topicPart then
32             return instance.ID;
33         end
34     end
35     return NULL;
```

Algorithm B.2: Instance Cleanup

```
1  Function CLEANUP_INSTANCE(metadataCategory, result, metadataCategoriesList):
2      possibleInstancesBasedOnParents ={};
3      foreach parent ∈ metadataCategory.parents do
4          if parent not in result then
5              continue
6          end
7          instanceIDsFromParent=metdataCategoriesList.get(parent);
8          foreach instanceID ∈ instanceIDsFromParent do
9              possibleInstancesBasedOnParents.add(metadataCategory.get(instanceID))
10         end
11     end
12     return possibleInstancesBasedOnParents
```

Algorithm B.3: Choosing next metadata categories

```
1  Function FIND_NEXT_CATEGORIES(listOfMetadataCategories):
2      nextCategory ← {};
3      foreach category ∈ listOfMetadataCategories do
4          if category.hasParent = NULL then
5              nextCategory insert category
6          end
7      end
8      if nextCategory ≠ {} then
9          return nextCategory;
10     end
11     return listOfMetadataCategories;
```