universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

A Neurosymbolic Approach for the Adaptive Configuration of IoT Platforms

verfasst von | submitted by

Danial Mohammadi Amlashi BSc

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2024

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 926
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Wirtschaftsinformatik
Betreut von | Supervisor:	o. Univ.-Prof. Dr.techn. Dimitris Karagiannis

Acknowledgements

I extend my gratitude to Prof. Karagiannis for his invaluable guidance and support throughout the writing process of this paper, as well as over the years. I am also appreciative of my colleague Alexander Völz for his insightful feedback, which greatly contributed to improving the structure and readability of this work. Their contributions have been instrumental in shaping the outcome of this work.

Abstract

The rise of Internet of Things (IoT) technologies has led to the emergence of complex ecosystems where diverse IoT devices interact to enable various applications and services. This thesis introduces a novel framework for automating the configuration of IoT environments tailored to specific scenarios. Addressing the challenge of manual and error-prone configuration processes, the framework employs semantic enrichment through a neurosymbolic approach to streamline the generation of digital twins for IoT devices. The framework provides a structured approach to aligning physical environments hosting IoT devices with predefined scenarios. The prototypical implementation, termed the *Instantiator pipeline,* comprises modules for network sniffing, semantic enrichment, and OpenHAB platform configuration. Evaluation in a real-world smart house scenario demonstrates the effectiveness of the approach in automatically configuring the OpenHAB IoT platform based on identified IoT devices and their functionalities. By automating the configuration process, the framework enables efficient deployment of IoT solutions while minimizing errors and enhancing scalability. This work contributes to advancing IoT platform configuration methodologies and provides a foundation for future works and development in the field.

Kurzfassung

Der Aufstieg der Internet of Things (IoT)-Technologien hat zur Entstehung komplexer Ökosysteme geführt, in denen verschiedene IoT-Geräte interagieren, um verschiedene Anwendungen zu ermöglichen. In dieser Arbeit wird ein neues Framework zur Automatisierung der Konfiguration von IoT-Umgebungen vorgestellt, das auf spezifische Szenarien ausgerichtet ist. Um der Herausforderung manueller und fehleranfälliger Konfigurationsprozesse zu entgehen, nutzt das Framework semantische Anreicherung durch einen neurosymbolischen Ansatz, um die Erstellung digitaler Zwillinge für IoT-Geräte zu vereinfachen. Das Framework bietet einen strukturierten Ansatz, um physische Umgebungen, die IoT-Geräte beinhalten, mit vordefinierten Szenarien abzustimmen. Die prototypische Implementierung, die als Instantiator-Pipeline bezeichnet wird, umfasst Module für Netzwerk-Sniffing, semantische Anreicherung und OpenHAB-Plattformkonfiguration. Die Evaluierung in einem realen Smart-House-Szenario zeigt die Effektivität des Ansatzes bei der automatischen Konfiguration der OpenHAB-IoT-Plattform auf der Grundlage der identifizierten IoT-Geräte und ihrer Funktionalitäten. Durch die Automatisierung des Konfigurationsprozesses ermöglicht das Framework eine effiziente Bereitstellung von IoT-Lösungen bei gleichzeitiger Minimierung von Fehlern und Verbesserung der Skalierbarkeit. Diese Arbeit trägt dazu bei, die Methoden zur Konfiguration von IoT-Plattformen voranzutreiben und bietet eine Grundlage für zukünftige Forschung und Entwicklung in diesem Bereich.

Contents

List of Figures

List of Tables

Listings

List of Acronyms

AI Artificial Intelligence

cAI Connectionist Artificial Intelligence

DSMR design science research methodology

DTwin Digital Twin

GPS Global Positioning System

IoT Internet of Things

IoTIM IoT Integration Middleware

IS Information Systems

LTE Long-Term Evolution

MQTT Message Queue Telemetry Transport

NFC Near-field communication

RFID Radio Frequency Identification

sAI Symbolic Artificial Intelligence

1. Introduction

In the era of universal connectivity, Internet of Things (IoT) has emerged as a transformative paradigm, promising to revolutionize various domains. As IoT deployments continue to expand, the need for effective configuration and management of IoT platforms becomes increasingly crucial. Configuring an IoT platform involves identifying IoT devices, understanding their capabilities, and integrating them into a cohesive ecosystem tailored to specific scenarios or use cases. However, manual configuration processes are often labor-intensive, error-prone, and lack scalability, necessitating automated approaches to streamline the configuration process.

This thesis addresses the challenge of configuring an adaptive IoT platform by proposing a novel approach that leverages semantic enrichment and neurosymbolic reasoning to automate the generation of digital twins for IoT devices. The methodology adopted follows the Design Science Research Methodology (DSRM), providing a structured framework for designing, implementing, and evaluating IT artifacts to address identified problems. The proposed framework integrates insights from the fields of IoT, semantic enrichment, and neurosymbolic reasoning, offering a systematic approach to align physical environments hosting IoT devices with specific scenarios.

The thesis is organized as follows:

- Chapter 2 provides an overview of the problem statement and motivation behind the research, highlighting the importance of automated IoT platform configuration in facilitating efficient and scalable deployment of IoT solutions.
- Chapter 3 presents the theoretical foundations required for understanding the proposed approach. This chapter explores existing approaches and methodologies of neurosymbolic AI, as well as techniques employed in automating IoT platform configuration.
- Chapter 5 outlines the proposed methodology for configuring an IoT platform, focusing on the automatic generation of Digital Twin (DTwin) for IoT devices within the context of specific scenarios. The methodology is grounded in the Design Science Research Methodology (DSRM) - explained in Chapter 4 - and provides a structured framework for conducting research and designing IT artifacts.
- Chapter 6 describes the prototypical implementation of the proposed approach, termed the *Instantiator pipeline*. This chapter also presents an evaluation of the Instantiator pipeline in a real-world smart house scenario. The evaluation assesses the effectiveness and functionality of the implemented approach in automatically configuring the OpenHAB IoT platform based on identified IoT devices and their functionalities.

- Chapter 7 delves deeper into the implications and potential applications of the proposed framework for IoT platform configuration.
- Chapter 8 concludes the thesis by summarizing the key findings, discussing the implications of the research, and outlining potential avenues for future work. The conclusion emphasizes the contribution of the proposed approach to advancing IoT platform configuration methodologies and its potential for further research and development in the field.

Overall, this thesis aims to contribute to the ongoing discourse on IoT platform configuration by introducing a novel approach that combines semantic enrichment and neurosymbolic reasoning to automate the generation of digital twins for IoT devices. By providing a systematic framework and prototypical implementation, this research lays the foundation for future advancements in automated IoT platform configuration and management.

2. Motivation and Goals

The main motivation behind this master thesis stems from the recognition of a challenge in the domain of IoT and the complexity and inefficiency associated with configuring and deploying IoT platforms. Despite the potential of IoT technologies, the complexities involved in setting up these platforms pose significant barriers, particularly for non-expert users. This also leads to inefficiently configured environments, where IoT Devices are not used to their full potential, or even incorrectly. As IoT continues to grow across various industries and sectors, there is an urgent need for intelligent solutions that optimize the configuration process, enabling broader adoption and maximizing the benefits of IoT technologies.

The goal of this master thesis is to develop an innovative framework for the adaptive configuration of IoT platforms, leveraging a Neurosymbolic approach. By harnessing the synergies between symbolic reasoning on domain knowledge and neural computation, the aim is to propose a paradigm shift in the way IoT platforms are configured and deployed. This work attempts to bridge the gap between complex technical configurations and user-friendly experiences, allowing stakeholders to effortlessly instantiate and manage IoT platforms efficiently without requiring extensive technical expertise.

Specifically, the objectives can be summarized as follows:

1. Designing a configuration framework: Designing a new framework that integrates principles from symbolic reasoning and neural network architectures to enable the automatic configuration of IoT platforms. This framework will encompass algorithms, methodologies, and tools tailored to address the complexities of IoT platform configuration while prioritizing usability and efficiency.

2. Implementing a prototype configuration system: Building upon the proposed framework, a prototype configuration system will be developed that demonstrates the feasibility and effectiveness of the proposed approach. The system will be capable of automatically configuring and deploying IoT platforms based on predefined a provided physical environment.

3. Evaluating the prototype system: comprehensive evaluations will be conducted to assess the performance, usability, and efficacy of the prototype configuration system.

4. Contributing to the body of knowledge: Beyond practical implementation, this work seeks to contribute to the theoretical understanding of IoT environments and practical implementation of IoT platform configuration methodologies. By showcasing the principles and mechanisms underlying the Neurosymbolic approach, the work aims to advance scholarly discourse in the fields of IoT, artificial intelligence, and computational intelligence.

In summary, this master thesis attempts to address the critical challenges surrounding the configuration of IoT platforms based on the knowledge available on the problem domain through an innovative Neurosymbolic approach. By combining theoretical insights with practical implementation, it aims to aid stakeholders with intelligent solutions that unlock the full potential of IoT technologies in diverse applications and domains.

3. Theoretical Foundations and Related Work

In the following chapter, theoretical foundations and related work relevant to understanding the work at hand and the proposed approach will be introduced and explained.

3.1. Artificial Intelligence

The goal of this section is to clarify the concept of neurosymbolic artificial intelligence, which serves an important part in the primary approach of this thesis. However, comprehending neurosymbolic artificial intelligence necessitates a prior understanding of the evolution as well as different classifications of Artificial Intelligence (AI). The field of AI is conventionally categorized into two main domains: *Symbolic Artificial Intelligence (sAI)* and *Connectionist Artificial Intelligence (cAI)* [19]. [11] outline the interplay between these domains by drawing parallels between AI and human intelligence. While sAI attempts to replicate the human mind's capacity for symbolically representing the world, cAI aims to emulate intelligence by modeling the neural structure of the human brain. It is widely acknowledged that human intelligence necessitates a fusion of both symbolic and connectionist approaches. Consequently, for the realization of robust artificial intelligence, the union of sAI and cAI becomes crucial. This integration forms the basis of neurosymbolic AI.

In the upcoming sub-sections, an in-depth exploration of each of the three types of AI will be provided, offering comprehensive insights into their underlying principles, methodologies, and applications.

3.1.1. Symbolic Artificial Intelligence

Symbolic AI is the sum of all AI methods that are based on abstract symbolic representation of problems, which can be solved through manipulation and destruction of the symbol system using search and logic [16]. In the early 1970s, Newell and Simon described symbols as the "root of intelligent action", and in 1976 published a paper called "Computer Science as Empirical Inquiry: Symbols and Search" [33], which is commonly referred to as the birth of sAI research. In this paper, they presented two laws of qualitative structure for artificial intelligence. The first law, called "The physical symbol system hypothesis" states:

> *A physical symbol system has the necessary and sufficient means for general intelligent action [33].*

This hypothesis specifies a general class of systems, consisting of a set of entities (symbols), which are components of another type of entity called expressions (symbol structures), capable of intelligent actions. However, a symbol system does not provide a way of accomplishing intelligent actions but rather provides the foundation for this. They state that symbol systems require a "heuristic search" to accomplish intelligent actions. The second law, called the "Heuristic-search hypothesis" states:

> *The solutions to problems are represented as symbol structures. A physical symbol system exercises its intelligence in problem solving by search - that is, by generating and progressively modifying symbol structures until it produces a solution structure [33].*

In summary, to achieve intelligent behavior, a system of expressions consisting of symbols as well as the ability to search and manipulate said expressions is required.

One of the earliest approaches to sAI is *Cognitive Simulation*, which was introduced by Newell and Simon to simulate human cognitive abilities by defining heuristic algorithms [14]. In 1989, John McCarthy proposed an alternative to Newell and Simon's approach. His idea stated, that intelligent systems should be designed and implemented based on logical reasoning, instead of simulators of heuristic rules [31]. This idea was the basis for the development of logic-based reasoning and rule-based systems.

3.1.2. Connectionist Artificial Intelligence

Connectionist AI, also known as Sub-Symbolic AI, encompasses methods that establish correlations between input data and output variables by optimizing parameters of a transformation function. This optimization process, often referred to as "learning," lies at the core of connectionist approaches. This paradigm stands in contrast to symbolic AI, offering distinct advantages, particularly in handling large-scale data with minimal or no predefined knowledge provided beforehand [19][23].

The learning process within connectionist AI is typically categorized into three fundamental paradigms: *Supervised Learning*, *Unsupervised Learning*, and *Reinforced Learning* [32].

In the Supervised Learning paradigm, the model is trained using labeled data, where the desired output data is predefined. Through a series of iterative steps, the transformation function is adjusted to map input data to the desired output with a high degree of accuracy. This optimization involves complex computational techniques to refine the function's parameters. Once optimized, the function can effectively classify unlabeled data based on the patterns learned during the training phase. One prevalent type is the *neural network*, a versatile architecture inspired by the human brain's interconnected neurons. Neural networks consist of layers of nodes, or neurons, interconnected by weighted edges. Through forward propagation and backpropagation algorithms, neural networks iteratively adjust these weights during training to minimize prediction errors. Convolutional neural networks (CNNs) excel in processing grid-like data, such as images, by leveraging convolutional layers to extract hierarchical features. Recurrent neural

networks (RNNs), on the other hand, are designed to handle sequential data, exhibiting memory through recurrent connections [8][32].

In contrast to supervised learning, unsupervised learning is a paradigm within cAI where the model is trained on unlabeled data, without explicit guidance or predefined output. Instead of being provided with labeled examples, the model autonomously identifies patterns, structures, or relationships within the input data. Common techniques in unsupervised learning include clustering, dimensionality reduction, and density estimation. Clustering algorithms group similar data points together, revealing inherent structures or clusters within the data. Dimensionality reduction methods aim to simplify the data by extracting its essential features while preserving its meaningful information. Density estimation approaches model the probability distribution of the data, enabling insights into its underlying statistical properties. Unsupervised learning plays a crucial role in exploratory data analysis, anomaly detection, and feature learning, offering valuable insights into complex datasets without the need for labeled annotations [24][15].

One prominent example of an unsupervised learning model are Transformers. Transformer models are a type of deep learning model architecture that has gained significant popularity in natural language processing (NLP) tasks. Unlike traditional recurrent neural networks (RNNs) or convolutional neural networks (CNNs), transformer models rely on self-attention mechanisms to process input sequences in parallel, making them highly efficient for capturing long-range dependencies in text data. These models are often pre-trained on large text corpora using unsupervised learning objectives, such as masked language modeling or next sentence prediction, and fine-tuned on task-specific datasets to achieve optimal performance on downstream tasks [30].

Reinforcement learning is a dynamic paradigm within cAI where an agent learns to make sequential decisions by interacting with an environment to maximize cumulative rewards. Unlike supervised and unsupervised learning, reinforcement learning operates in a trial-and-error fashion, where the agent receives feedback in the form of rewards or penalties based on its actions. The agent learns to navigate the environment by exploring different actions and observing their outcomes, gradually discovering the optimal strategy through a process of trial, error, and reinforcement [25].

3.1.3. Neurosymbolic Artificial Intelligence

Neurosymbolic AI represents an innovative fusion of connectionist and symbolic approaches, aiming to leverage the strengths of both paradigms to overcome their respective limitations. By combining the representational power of symbolic systems with the learning capabilities of neural networks, neurosymbolic AI endeavors to enable machines to reason, learn, and generalize effectively across diverse domains.

As visualized in 3.1, neurosymbolic integration can be divided into two main approaches: *Unified* and *Hybrid*.

The unified approach seeks to seamlessly integrate neural and symbolic processing within a single architecture, thereby enabling synergistic interactions between the two paradigms. In this approach, neural networks and symbolic reasoning components are tightly coupled, allowing for a bidirectional flow of information and feedback between

Connectionism	neurosymbolic integration				Symbolicism
	Unified approaches		Hybrid approaches		
	Neuronal Symbol Proc.	Connectionist Symbol Proc.	Functional hybrids	Translational hybrids	
Segregation	*Neuronal eliminativism*	*Connectionist eliminativism Limitivism Revisionism*	*Hybridization or cohabitation*		*Segregation Implementation–alism*

Figure 3.1.: Synoptic view of neural, symbolic and nuerosymbolic approaches by [21]

the two levels of representation. This integration often involves embedding symbolic knowledge directly into neural network structures or incorporating neural components into symbolic reasoning systems. The unified approach holds promise for leveraging the complementary strengths of neural and symbolic techniques, enabling more robust and flexible AI systems capable of handling complex tasks that require both learning from data and reasoning with explicit knowledge. By unifying neural and symbolic processing, this approach aims to overcome the limitations of each paradigm in isolation and achieve greater AI capabilities [21][23].

In contrast to the unified approach, the hybrid approach to neurosymbolic integration maintains a more modular and loosely coupled architecture, where neural and symbolic components operate semi-independently and interact through well-defined interfaces. In this approach, neural networks and symbolic reasoning systems are developed and trained separately, each focusing on their respective strengths. The neural component typically handles tasks involving pattern recognition, learning from raw data, and capturing complex statistical patterns, while the symbolic component deals with tasks requiring explicit knowledge representation, logical reasoning, and rule-based inference. The interaction between the two components occurs at various levels, such as feature extraction, knowledge incorporation, or decision fusion. By combining the outputs of both neural and symbolic processing streams, the hybrid approach aims to harness the complementary strengths of each paradigm while mitigating their individual limitations. This modular architecture offers flexibility and scalability, allowing for the incorporation of diverse neural and symbolic techniques tailored to specific application domains. Moreover, the hybrid approach facilitates interpretability and explainability, as the reasoning process can be traced back to symbolic rules and knowledge representations, providing insights into the AI system's decision-making process. Overall, the hybrid approach represents a pragmatic strategy for neurosymbolic integration, balancing the benefits of both paradigms to achieve more robust and versatile AI systems [21][23].

In this work, the functional hybrid neurosymbolic approach is particularly relevant. The functional hybrid approach combines aspects of both the unified and hybrid approaches, aiming to strike a balance between tightly coupled integration and modular independence. It emphasizes the functional aspects of neural and symbolic processing, focusing on how each component contributes to the overall AI system's functionality. In this approach, neural networks and symbolic reasoning systems are designed to perform specific functions within the system, with well-defined interfaces for communication and coordination.

Neural networks excel at tasks such as pattern recognition, feature extraction, and statistical learning, while symbolic systems handle explicit knowledge representation, logical reasoning, and rule-based inference. By outlining the functional roles of each component and orchestrating their interactions effectively, the functional hybrid approach achieves a harmonious integration of neural and symbolic techniques. This integration enables the AI system to leverage the strengths of both paradigms while addressing their respective limitations, resulting in more robust, adaptable, and interpretable AI systems capable of tackling complex real-world challenges across diverse domains [21].

3.2. Knowledge Graphs

A further paradigm in sAI is the *structural knowledge representation*, which is used to represent declarative knowledge, defined as static knowledge concerning facts [43]. Structural models of knowledge representation are usually in the form of hierarchical graphs, providing a visual and intuitive framework for organizing and understanding complex relationships between entities. Constructing *Knowledge Graphs* (also referred to as *Ontologies*) is the main goal of applying such models. A knowledge graph is defined as a conceptualization of a certain domain (also called *explicit domain knowledge*), capturing the essential entities, relationships, and constraints within that domain. Through the application of reasoning techniques, ontologies enable the extraction of implicit domain knowledge, revealing hidden patterns, associations, and insights that may not be immediately apparent from the explicit representations alone. The most common notation used for representing knowledge in structural knowledge models are *Semantic Networks*, which are sets of triples in the form of subject-predicate-object [14]. These triples encode assertions about the relationships between entities, where the subject represents the entity or concept being described, the predicates describes the properties or attributes of entities, and objects represent the values or entities themselves. Semantic networks provide a flexible and expressive means of capturing the semantics of a domain, facilitating both human understanding and machine inference. They serve as the building blocks for constructing more complex knowledge graphs, laying the foundation for advanced reasoning and decision-making in intelligent systems. It's crucial to emphasize that knowledge graphs serve as a tool for representing knowledge rather than a mechanism for intelligent behavior. The true capability for decision-making emerges through the processes of reasoning and inference.

Semantic alignment plays a pivotal role in enhancing the expressiveness and interoperability of knowledge graphs. This can be achieved through semantic enrichment [51]. Semantic enrichment involves augmenting the existing knowledge graph with additional semantic information, such as ontological annotations, semantic metadata, or semantic relationships extracted from external sources. By aligning the semantics of entities and relationships across different knowledge graphs or domains, semantic enrichment enables more effective integration and fusion of heterogeneous data sources, enabling seamless information exchange and interoperability. Moreover, semantic alignment enhances the interpretability and accuracy of knowledge graphs, allowing the machines to understand

and reason over the underlying semantics more effectively. Through semantic enrichment, knowledge graphs become enriched with contextual information, enabling more precise and semantically rich inference and decision-making in intelligent systems.

3.3. Internet of Things

First proposed by Kevin Ashton in 1999 [4], IoT is a paradigm rapidly gaining attention in the domain of smart environment. Due to this fact, many definitions for IoT can be found in the literature. The author(s) in [5] define IoT as

> *the pervasive presence around us of a variety of things or objects – such as Radio-Frequency IDentification (RFID) tags, sensors, actuators, mobile phones, etc. – which, through unique addressing schemes, are able to interact with each other and cooperate with their neighbors to reach common goals.*

According to [28], IoT may be seen as

> *a global network infrastructure where physical and virtual objects with unique ID are discovered and integrated seamlessly (taking into account security and privacy issues) in the associated information network where they are able to offer and receive services which are elements of business processes defined in the environment they become active.*

In summary, IoT can be understood as a paradigm combining network communication and computation, where a set of uniquely identifiable devices often referred to as "things" are interconnected, allowing them to gather, exchange, process, and consume data [29][42].

The communication topology in an IoT environment can be divided into the following three categories, as proposed by [9] (visualized in figure 3.2):

- *centralized* networks, in which all devices (nodes) are connected to a central node, which acts as a controller.
- *decentralized* networks, which have no central node, and each node is connected to at least one or more neighboring nodes.
- *hierarchical* networks, in which multiple nodes act as central nodes, which again are connected in multiple levels of hierarchy.

3.3.1. IoT Device

IoT Devices, also commonly referred to as "Things", can be seen as the end devices of the IoT, which are located in the real world and are used to gather data and manipulate the environment to achieve certain goals [37][17].

In their most abstract form, these devices can be classified into one (or both) of the following two categories: *sensors* and *actuators*. The term *sensor* defines electronic

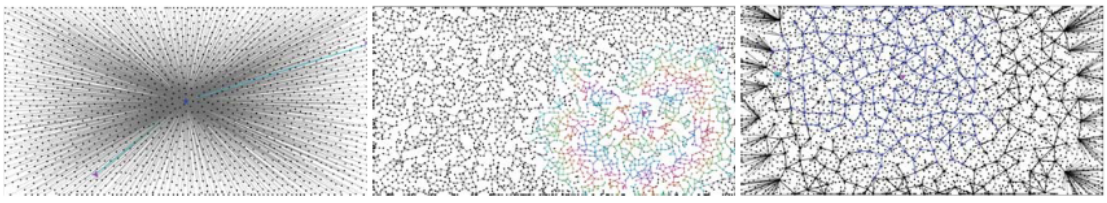


Figure 3.2.: Overview of three network typlogies visaualized by [9]: (a) centralized, (b) decentralized, and (c) hierarchical

objects, that can measure real-world parameters (E.g. temperature, light, humidity, etc.). Further, *Actuators* refer to electronic objects, that have the capability to manipulate their physical environment by performing actions over themselves, or on other devices (E.g. motors, radiator, etc.)[17][18]. However, in some literature, this categorization is extended with terms such as *single board computers* and *intelligent devices* to include more complex devices such as Raspberry Pi's and Arduino's into the definition of IoT Devices [50].

Communication among IoT devices can be achieved through various technologies and protocols. Some of the commonly employed communication technologies include:

- *WI-FI*: This technology utilizes the IEEE 802.11 standard, employing radio waves to establish robust and high-speed local area network connections, enabling seamless communication between devices within a confined geographical area [26][49].

- *Bluetooth*: Developed in 1994 (Bluetooth 1.0), Bluetooth is a wireless technology standard primarily used for short-range network communication, facilitating convenient and efficient data exchange between devices in close proximity [26][49].

- *Long-Term Evolution (LTE)*: LTE, also known as cellular network technology, originated in the 1990s and has evolved to its 5th generation (5G). It offers reliable, high-speed, and low-latency wireless communication over long distances [49].

- *ZigBee*: Similar to Bluetooth, Zigbee is utilized for short-range network communication. Due to its low data rate, ZigBee devices provide high transfer reliability and a prolonged lifespan. This makes them suitable for integration into smart building infrastructures where easy replacement is challenging, such as ground parking sensors [26][47][39].

- *Radio Frequency Identification (RFID)*: Developed in the early 20th century, RFID technology uses radio waves for peer-to-peer connections between objects. It comprises a tag for data, a reader for retrieval, and middleware for forwarding. With applications in retail, crime prevention, waste management, and healthcare, RFID has significant potential for various industries [26][47][39].

- *Near-field communication (NFC)*: Inspired by the principle of magnetic coupling between devices, NFC is an RFID-based technology that enables extremely short-range communication (up to 10cm). It facilitates bidirectional data transfer between devices containing NFC modules (NFC-Readers) and physical data media (NFC Tags or Chips), as well as with other devices equipped with NFC-Readers. Contactless payment is one of the most well-known applications of NFC technology [39][49].

- *Message Queue Telemetry Transport (MQTT)*: Developed as a part of the IBM MQ product line, MQTT is a topic-based publish/subscribe communication protocol, which relies on other underlying networks such as WI-FI and LTE for the communication. The advantage of this protocol is its lightweight nature, making it suitable for use in constrained environments such as IoT devices with limited processing power and bandwidth. Additionally, MQTT ensures efficient data transfer by minimizing overhead and supporting asynchronous messaging, enhancing scalability and reliability in distributed systems [22].

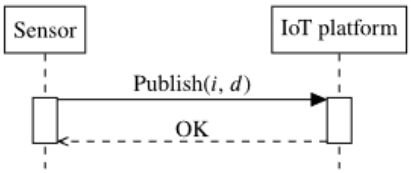
IoT Device Discovery

Navigating IoT environments often presents a significant limitation: the seamless integration of IoT devices into the ecosystem. The initial step of this process, which is discovering the devices, entails the identification and acknowledgment of all IoT devices intended for inclusion in the ecosystem. This can prove challenging due to the diverse nature of IoT devices, each with its unique communication protocols, data formats, and connectivity methods. Consequently, implementing robust mechanisms for device discovery becomes imperative. This may involve leveraging network scanning tools, device APIs, or even employing machine learning algorithms to recognize and catalog devices automatically.

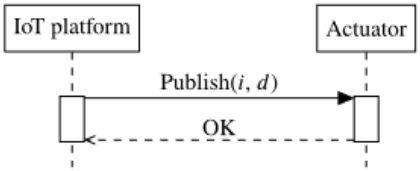
The author(s) of [40] propose an approach that leverages similarity algorithms applied to network traffic data for the purpose of discovering and identifying relevant devices. This method involves comparing the patterns and characteristics of network traffic against a predefined database, allowing for the automated recognition of devices within the network ecosystem.

Further, [1] proposes an automated approach to device integration, wherein devices are classified based on their communication properties, enabling the automatic generation of interfaces and integration drivers. By leveraging IoT communication data and a substantial device information database, the study demonstrates the feasibility of identifying devices within a network, thereby streamlining the device integration process and reducing the need for human intervention.

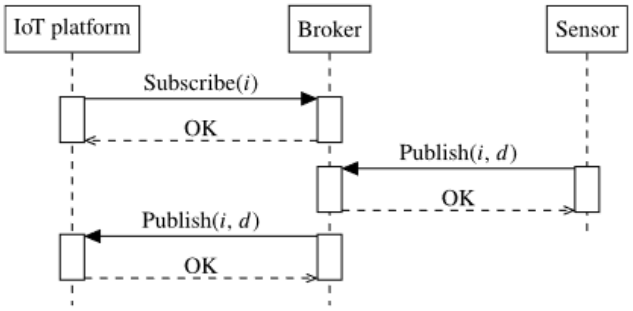
Lastly, [46] underscore the transition from traditional techniques like port-based and payload-based methods to the adoption of machine learning for enhanced accuracy in device identification. The paper highlights the current state of IoT device classification, acknowledging its nascent stage compared to other areas like network traffic classification.



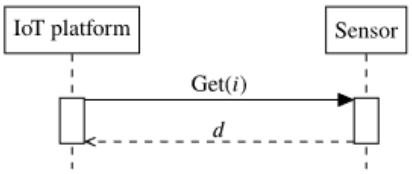
(a) Publish model: from a sensor to the IoT platform.



(b) Publish model: from the IoT platform to an actuator.



(c) Publisher-subscriber model: information exchange between a sensor and the IoT platform.



(d) Polling moddel: from the IoT platform to a sensor.

Figure 3.3.: Overview of data exchange models by [10]: (a) Publish model for sensors, (b) Publish model for actuators, (c) Publisher-subscriber model for sensors, (d) Polling model for sensors

3.3.2. IoT Platform

In the rapidly evolving landscape of the Internet of Things (IoT), the complexity of managing diverse devices, protocols, and data streams poses significant challenges for users seeking to harness the full potential of IoT technology. As such, there arises a critical need for a centralized instance to serve as the foundational infrastructure for deploying, managing, and extracting actionable insights from interconnected IoT devices and systems. The IoT Integration Middleware (IoTIM), commonly referred to as IoT Platform, serves as an integration layer for the IoT Devices, by creating a DTwin of the device. A DTwin refers to a virtual copy of a physical entity (physical twin), which can mimic and control its physical counterpart through real-time exchange of data [48]. Therefore, leveraging IoT platforms not only facilitates device management from inside, as well as outside of the ecosystem using HTTP-Based REST APIs, but also enables the creation of digital replicas that enhance monitoring, analysis, and optimization of IoT deployments, driving efficiency and innovation in various domains [18][13].

The process of integrating a device with an IoT platform typically involves establishing a representation of the device within the platform environment, sometimes referred to as an "Item". However, as discussed earlier, to create a DTwin of a device, there must be real-time data exchange between the physical device and its virtual counterpart. [10] outline three fundamental models of data exchange with IoT platforms, as depicted in Figure 3.3:

- *Publish model*: This model operates on a client-server architecture, facilitating the reception of real-time data from sensors (client) within the IoT platform (server). Similarly, it can transmit commands from the IoT platform (client) to an actuator (server). Although commonly used, this model may lack reliability due to its dependence on the connection session between the server and the client.

- *Publisher-subscribe model*: While less prevalent for actuators, this model overcomes the session-based approach of the publish model by introducing a message broker between the IoT platform and the sensor. Acting as a message queue, the broker stores incoming data from the sensor on specific topics, enabling one or more IoT platforms to access the data by subscribing to the desired topic.

- *Polling model*: Similar to the publish model, the polling model swaps the roles of the IoT platform and the sensor, with the IoT platform (client) requesting data from the sensor (server) at intervals.

Arguably, the most crucial advantage of IoT platforms lies in automating the IoT environment. With a digital representation of the surroundings in place, rules can be leveraged, which are activated in response to specific events. This automation empowers the system to autonomously execute predefined actions, enhancing efficiency, responsiveness, and overall operational effectiveness within the IoT ecosystem [10][18]. Such rules can be defined using intuitive graphical interfaces or specialized scripting languages, enabling users to tailor the behavior of their IoT deployments to suit specific requirements and optimize performance [13].

OpenHAB

OpenHAB, an acronym for Open Home Automation Bus, is an open-source platform designed to integrate and control various IoT devices within smart home environments. Developed as part of the Eclipse SmartHome project, OpenHAB offers a flexible and extensible framework for connecting and managing diverse IoT devices, protocols, and technologies. In OpenHAB, "Things," "Items," and "Rules" are fundamental concepts that play crucial roles in building and managing smart home automation systems [44][36][7]:

- A "Thing" represents a physical or virtual device that can be connected to the smart home system. This includes sensors, actuators, smart appliances, and other IoT devices. It is defined in the system configuration to establish a connection and communication link with the corresponding physical device. This involves specifying the device's type, communication protocol, network address, and other relevant parameters. OpenHAB supports a wide range of device types, including those based on standard protocols like MQTT, Zigbee, HTTP, and many others.

- An "Item" in OpenHAB represents a specific data point or functionality associated with a "Thing." It serves as an abstraction layer that encapsulates the device's state, sensor readings, control commands, or other data. They can be categorized into different types based on their functionality, such as Switch, Contact, Number, String, DateTime, and Group. Through "Items," users can read sensor data, control actuators, trigger events, and define automation rules based on device states or changes.

- Rules in OpenHAB define automation logic and behaviors that govern how the system responds to events. A rule typically consists of three components: a trigger, conditions, and actions. The trigger specifies the event or condition that initiates the rule, while conditions define additional criteria that must be met for the rule to execute. Actions specify the tasks or commands to be performed when the rule is triggered. Rules can be created using a rule-specific language like Xtend or through graphical rule editors available in the OpenHAB user interface.

4. Methodology and Approach

The methodology adopted for this work follows the design science research methodology (DSMR) framework proposed by [27]. DSMR provides a structured approach for conducting research aimed at designing and evaluating IT artifacts to address identified problems. It follows a systematic process outlined in six steps, as proposed by Archer [3], focusing on purpose-oriented designs to seek solutions for specific problems. These steps include programming, data collection and analysis, synthesis, development, prototyping, and documentation. DSRM emphasizes the importance of conceptual principles, practice rules, and a defined process for carrying out and presenting research. In the context of Information Systems (IS) research, DSRM involves creating and evaluating IT artifacts to solve significant business problems. [20] provided practice rules for conducting design science research in IS, emphasizing the importance of producing artifacts relevant to unsolved business problems, precisely evaluating their utility and quality, and communicating research findings to a fitting audience. Overall, DSRM offers a structured framework for conducting research that produces valuable, rigorous, and publishable results in IS research outlets.

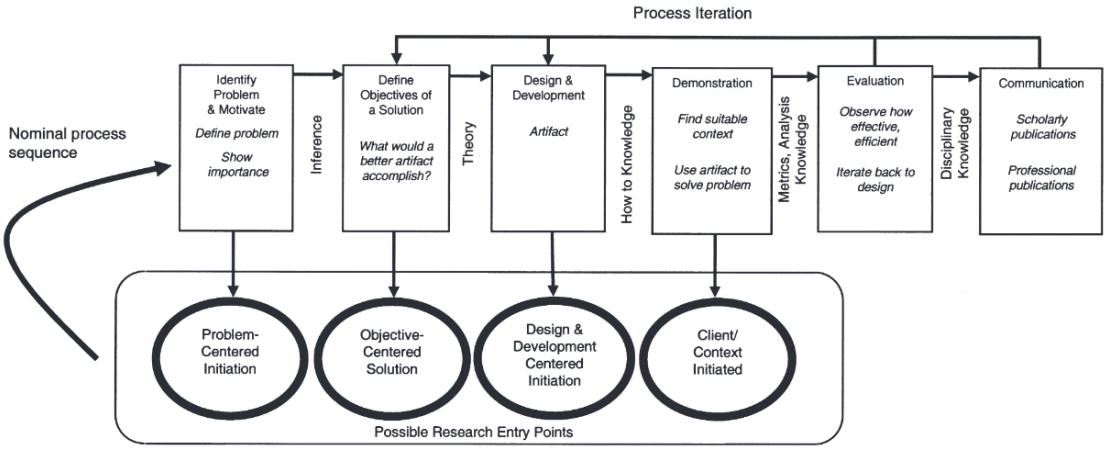


Figure 4.1.: DSMR process model proposed by [27]

As visualized in Figure 4.1, the initial phase of the DSMR is the problem definition. In the context of this work, the identified problem, as mentioned in Chapter 2, revolves around the difficulties associated with the configuration of IoT Platforms for average users. This challenge emphasizes the need for a solution, which sets the stage for the next step in the DSRM process: defining a solution through the development of a software artifact.

As explained in Chapter 2, the overall objective of this artifact is to enable the automatic configuration of an IoT Platform based on the collection of IoT devices available within the environment. This phase serves to define the conceptual framework essential for the creation of a software artifact tailored to address the identified problem. The conceptual framework and approach defined for this work will be introduced in Chapter 5.

Following the design and development phases of the artifact, it is necessary to establish a context for demonstrating and evaluating the usefulness of the proposed solution. This requires identifying a suitable environment or scenario where the artifact can be deployed and its functionality tested comprehensively. The evaluation process serves as a critical milestone for determining the viability and effectiveness of the solution in addressing the identified problem. Through evaluation, key insights are gained regarding the artifact's performance, usability, and alignment with user needs and expectations. This evaluative phase plays a key role in informing subsequent decisions, as it enables stakeholders to verify whether the solution meets the desired objectives or requires iterative refinement through a return to the prior stages of problem definition, solution design, and development. By leveraging evaluation outcomes, researchers can iteratively enhance the artifact's functionality and performance, ensuring its alignment with user requirements and maximizing its utility within the intended context. The proposed context and evaluation case designed for this work is introduced in Chapter 6.

5. A Domain Specific Neurosymbolic Approach for IoT Environment Configuration

In this thesis, the configuration of an IoT Platform refers to identifying IoT devices and generating an affiliated digital twin within the IoT Platform based on the active scenario. This process is necessary for a broader framework to semanticaly align physical environments hosting IoT Devices and physical elements — defined as objects lacking IoT attributes — with specific scenarios. The proposed framework is visualized in Figure 5.1, serving as a visual guide to be elaborated upon in subsequent sections. For simplicity, the framework is segmented into three main components, along with an alignment result section.

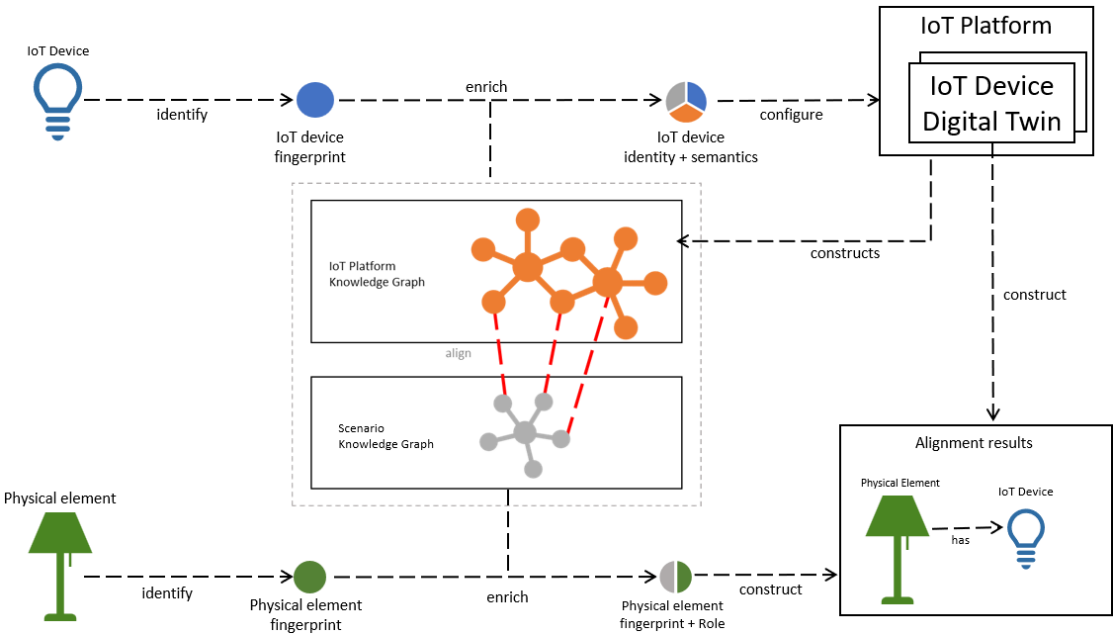


Figure 5.1.: The IoT environment alignment framework

5.1. Alignment Knowledge Graph

At the heart of the framework presented in Figure 5.1 lies a knowledge graph, serving as the foundational knowledge repository. This knowledge graph plays a crucial role in integrating insights about the scenario and the specific IoT platform utilized in the framework instance. It represents a unification of the structural and functional aspects of the IoT platform instance with the requirements of the scenario and its associated components. This alignment is constructed by establishing semantic relationships between the concepts inherent in the scenario and those representing the capabilities of the IoT platform.

5.2. Configuration of IoT Device Digital Twin

This process within the framework presented on the top side of Figure 5.1 concerns constructing digital twins of IoT devices. It encompasses the three steps of fingerprinting the devices, semantically enriching the collected data, and generating functioning digital counterparts within the utilized IoT platform. The following sub-sections will offer an elaborate explanation of each step within the process.

5.2.1. IoT Device Identification

The identification process of an IoT device involves generating a digital representation from the device (referred to as IoT device fingerprint in Figure 5.1), encapsulating information that can be obtained about its functionality and characteristics. However, it's important to note that this digital fingerprint cannot be classified as a complete digital twin at this stage, as it lacks bidirectional synchronization with the physical counterpart.

In this step, the two primary variables are the collected information and its respective source. The gathered information contains data related to the IoT device's attributes, functionalities, and operational parameters. These parameters provide crucial insights into the device's capabilities and behavior, enabling its integration and utilization within the IoT ecosystem.

The values assigned to various parameters can vary based on two main factors: the role of the IoT device within its operational environment and its technical classification. The role of the device influences the significance and interpretation of parameters, as different functions require distinct sets of attributes and functionalities. For instance, consider a temperature sensor and a smart light bulb. While both are IoT devices, their roles within the operational environment differ significantly. The temperature sensor primarily provides environmental data, focusing on parameters such as temperature, humidity, and pressure. In contrast, the smart light bulb is tasked with illumination and control, involving parameters related to brightness levels, color temperature, and power consumption. As a result, the importance and relevance of specific parameters vary based on the device's role. For example, the exact position parameter of a temperature sensor may be considered less critical compared to the exact position of a smart light bulb. While

the temperature sensor may primarily function indoors, providing localized environmental data, the precise location of the light bulb is vital for context-aware lighting control and automation.

Furthermore, the technical classification of the device also influences parameter selection and interpretation. Different types of IoT devices may possess unique features, functionalities, and operational requirements, leading to variations in parameter relevance and significance. For example, a motion sensor and a humidity sensor serve distinct purposes and operate based on different principles. While the motion sensor detects movement and triggers actions based on activity detection, the humidity sensor monitors moisture levels and enables environmental control systems.

Concerning the source of information, data can be acquired through direct observation of the IoT device, which may occur via physical or digital means. Physical observation involves direct inspection of the device's physical components, functionalities, and behaviors in real-world settings. This method allows observers to interact with the device firsthand, gaining insights into its tangible characteristics and operational behavior.

Physical observation can be done using various technologies, including Global Positioning System (GPS) and image recognition. GPS technology enables tracking and geolocating IoT devices, providing valuable spatial data regarding their positions and movements. This spatial information is essential for understanding the device's context within its operational environment, enabling effective location-based services and optimizations.

Image recognition systems leverage visual data to identify and analyze physical attributes or patterns associated with the devices. By analyzing images captured by cameras or sensors, observers can identify devices as well as their role in the local environment, detect anomalies, and assess their physical conditions. This visual inspection process enhances the configuration process with detailed real-world insights, enabling accurate fingerprinting.

Digital observation on the other hand requires accessing data streams generated by the device's sensors, internal systems, or communication interfaces. This approach involves monitoring digital outputs, sensor readings, telemetry data, logs, and other digital signals emitted by the device. By interfacing with device APIs, protocols, or communication interfaces, stakeholders can extract relevant information remotely, enabling real-time monitoring and analysis of device performance without direct physical interaction.

By leveraging digital observation techniques, stakeholders can gain comprehensive insights into IoT device behavior, enabling efficient configuration and management processes. This approach allows for proactive monitoring, timely intervention, and informed decision-making, ultimately enhancing the overall performance and reliability of IoT deployments.

IoT Device Fingerprinting is a method used to capture digital data within an environment by uniquely identifying devices based on their distinct characteristics or attributes. This technique involves analyzing various parameters associated with devices, such as network traffic patterns, hardware identifiers, software configurations, and communication protocols. Collecting this information generates a digital fingerprint for each device, enabling its identification and differentiation from others within the environment. Finger-

printing, as explained in chapter 3.3.1, serves as a valuable tool for inventory management, security monitoring, and configuration purposes in IoT ecosystems, facilitating accurate tracking and analysis of device activities [6].

The output of the IoT device identification step in the configuration process is a digital representation of the IoT device (referred to as "IoT device fingerprint" in Figure 5.1), which encapsulates the gathered information and its source. This digital fingerprint serves as an initial profile or snapshot of the device, containing essential attributes, functionalities, and contextual data relevant to its operation. While this representation is not yet considered a complete digital twin, it lays the foundation for further configuration and management activities.

5.2.2. Semantic Enrichment

The Semantic enrichment step involves enhancing existing data with contextual semantics by utilizing graph-based data structures such as RDF (Resource Description Framework). This process aims to enrich the understanding and interpretation of data by establishing meaningful connections and relationships between disparate datasets [51]. By leveraging RDF, semantic enrichment allows the integration of linked open data (LOD) sources, enabling the augmentation of existing data with additional context and insights from domain-specific (external) knowledge repositories. Semantic enrichment enhances the interoperability, discoverability, and interpretability of data within various scenarios and domains, allowing for more intelligent and context-aware information systems.

Utilizing semantic enrichment techniques, the digital representation of IoT devices can be enriched with additional contextual semantics, provided by the knowledge graph. This process involves augmenting the existing data associated with the device by establishing semantic relationships with relevant concepts, entities, and resources available in the available knowledge graph specific to the IoT Platform and/or scenario. By enriching the digital representation in this manner, the platform gains a deeper understanding of the device's attributes and functionalities, allowing more accurate and comprehensive interpretations of its data.

This enriched digital representation enables the classification and accurate creation of the device's digital twin within the IoT Platform. By leveraging semantic relationships established through the knowledge graph, the platform can effectively categorize and instantiate a digital twin that closely mirrors the physical device's characteristics and behavior. This digital twin serves as a virtual counterpart to the physical device, enabling simulation, monitoring, and control in a virtual environment. Moreover, it allows seamless integration with other IoT devices and systems, enhancing overall interoperability and functionality within the IoT ecosystem.

However, establishing a meaningful link between the digital fingerprint representing the IoT device and the provided knowledge graph requires a matching process to identify a relevant concept within the graph. This matching process is crucial for accurately enriching the digital representation with contextual semantics from the knowledge graph. One effective approach to achieve this matching is through the application of link prediction methods, which often rely on neurosymbolic AI approaches (explained in chapter 3.1).

Link prediction methods leverage machine learning algorithms, which may incorporate both symbolic reasoning and neural network architectures, to infer or predict connections between entities in a graph-based knowledge representation [35]. These methods analyze the attributes, features, and relationships embedded within the digital fingerprint and the knowledge graph to identify potential links or associations between them. By leveraging the information available in both the digital fingerprint and the knowledge graph, these methods can create semantic connections that enhance the understanding and interpretation of the IoT device's characteristics.

Specifically, in the context of IoT platforms, link prediction methods based on neurosymbolic AI play a crucial role in bridging the gap between digital fingerprints and domain knowledge graphs. These methods enable the seamless integration of contextual semantics by accurately identifying relevant concepts and relationships within the knowledge graph that correspond to the attributes and functionalities of the IoT device. This integration allows the creation of accurate digital twins within the platform, which closely mirror the behavior and characteristics of their physical counterparts.

The outcome of this phase (referred to as "IoT device Identity + semantics" in Figure 5.1) is an enriched digital fingerprint that contains both the device attributes and the essential data required for generating a digital twin within the IoT Platform. This enriched digital fingerprint incorporates contextual semantics extracted from the domain knowledge graph, enabling a more detailed description of the device's attributes, functionalities, and operational characteristics. Moreover, the enriched digital fingerprint encompasses additional information vital for the precise creation of a digital twin within the IoT Platform. This includes appropriate metadata, semantic annotations, and contextual relationships established through semantic enrichment techniques. By encapsulating all the data and semantics, the enriched digital fingerprint serves as a complete blueprint for instantiating an accurate digital twin that replicates the behavior and characteristics of the physical device within the IoT Platform.

5.2.3. Deployment on IoT Platform

Following the semantic enrichment process, the transformed digital fingerprint appears in a suitable format for configuration of DTwin within the IoT Platform. The configuration procedure may vary in complexity depending on the specific attributes of the device being represented. For instance, sensors may necessitate the configuration of data channels to enable data transmission and reception, whereas actuators may require remote control functionalities facilitated through the digital twin.

The configuration process requires deploying the enriched digital fingerprint onto the IoT Platform while ensuring compatibility with the platform's infrastructure and services. This includes configuring communication protocols, establishing robust data pipelines, and defining precise control mechanisms tailored to the device's capabilities and requirements. By appropriately instantiating the digital fingerprint within the IoT Platform, users can effectively harness its digital twin to monitor, manage, and interact with the corresponding physical device in real-time.

Furthermore, the configuration process may also involve additional considerations such

as security protocols, data privacy measures, and scalability aspects to ensure seamless integration and operation within the IoT ecosystem. By addressing these aspects during configuration, users can enhance the reliability, efficiency, and effectiveness of the digital twin deployment, thereby maximizing its utility and value within the IoT Platform.

5.3. Physical Element Alignment

Similar to the process described in Chapter 5.2, a comparable procedure is necessary to identify and utilize the physical elements present in the environment to determine their role in the scenario (visualized in the bottom side of Figure 5.1). This process consists of three steps: identifying the physical elements, enriching them with scenario semantics to determine their role in the scenario, and aligning them with an IoT device to utilize them in the scenario. It is important to note that within this framework, a physical element is considered either a single object or a collection of objects that can be perceived as a unified concept. This definition ensures clarity and consistency in the identification and semantic enrichment processes, allowing for effective alignment with the scenario requirements.

Similar to the identification of IoT devices described in Chapter 5.2.1, the physical elements of an environment can be identified through direct observation. However, only physical observation is possible, as physical elements do not produce a digital fingerprint. This can be achieved through the utilization of computer vision methods for visual recognition.

The process of semantic enrichment of the physical element fingerprints is also comparable to the process introduced in 5.2.2, which employs neurosymbolic AI methods to map attributes of the fingerprint to concepts in a knowledge graph and enrich them through reasoning. However, in this context, only the scenario knowledge graph is utilized, as the IoT Platform knowledge graph pertains only to IoT devices. The outcome of semantic enrichment provides insights into the role of the physical element within the context of the scenario. For instance, a chandelier might be identified as serving the role of illumination within the context of a room.

In the final step of the process, it is important to align the physical elements with corresponding IoT devices, if necessary. This alignment enables the creation of pairs between physical elements and the IoT devices captured in the process described in Chapter 5.2, ultimately defining the role of the IoT device within the context of the scenario. Furthermore, this alignment enables the optimization of IoT devices for various purposes within the scenario, enabling the devices to assume multiple roles as needed. It also separates the IoT capabilities required for a role from the physical presence of the IoT device, providing flexibility and adaptability in scenario configurations.

6. Prototypical Implementation and Evaluation Case

This chapter introduces a prototypical implementation of the adaptive IoT platform configuration using a physical experiment environment.

6.1. The Smart House Case

The chosen case for evaluating the prototypical implementation of the intelligent IoT platform configuration is the smart house. This case represents a comprehensive environment where diverse IoT devices interact and operate within a household setting to monitor and manage various aspects of the residents' daily lives, as well as specific scenarios. Figure 6.1 shows the layout utilized to represent the smart house within the physical experiment environment.

6.2. The Experiment Environment

The physical experimental environment employed for this case incorporates an OpenHAB IoT Platform instance, a Mosquitto MQTT message broker (both running on a Raspberry Pi), and numerous sensors and actuators, all interfaced with an Arduino microcontroller for data transmission and reception via the MQTT protocol. The initial setup of the experiment environment is visualized in figure 6.2

6.2.1. Mosquitto MQTT Broker

One crucial component of the experiment environment is the Mosquitto MQTT message broker [12]. As a key intermediary, Mosquitto facilitates communication between various IoT devices, sensors, actuators, and the OpenHAB IoT Platform instance. It plays a key role in enabling seamless data exchange and interaction within the smart house ecosystem. By utilizing the MQTT protocol, Mosquitto ensures efficient and reliable transmission of messages between the connected devices, thereby enabling real-time monitoring and control functionalities.

6.2.2. OpenHAB

OpenHAB [34], an open-source home automation platform, serves as the central hub for managing and controlling IoT devices within the smart house environment. With its adaptable architecture and extensive plugin ecosystem, OpenHAB provides a flexible and

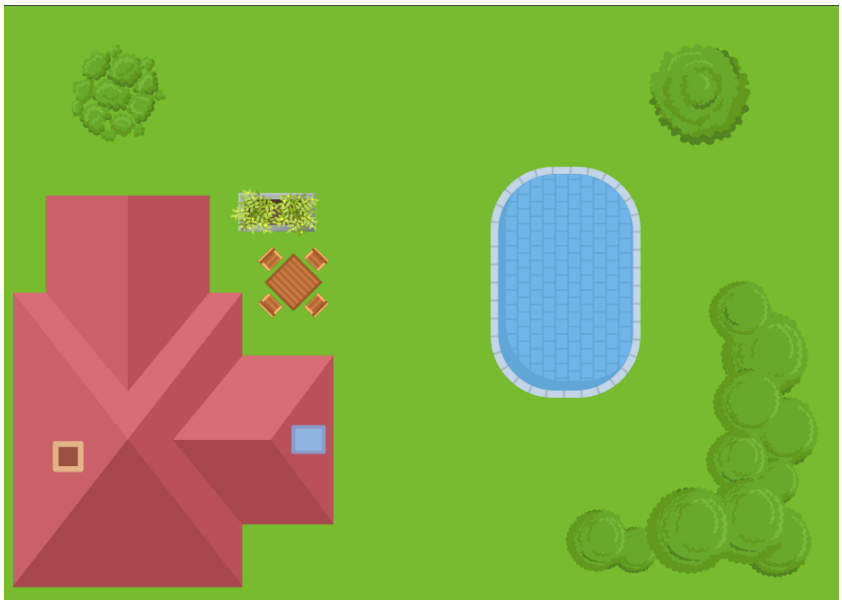


Figure 6.1.: Physical Experiment Layout

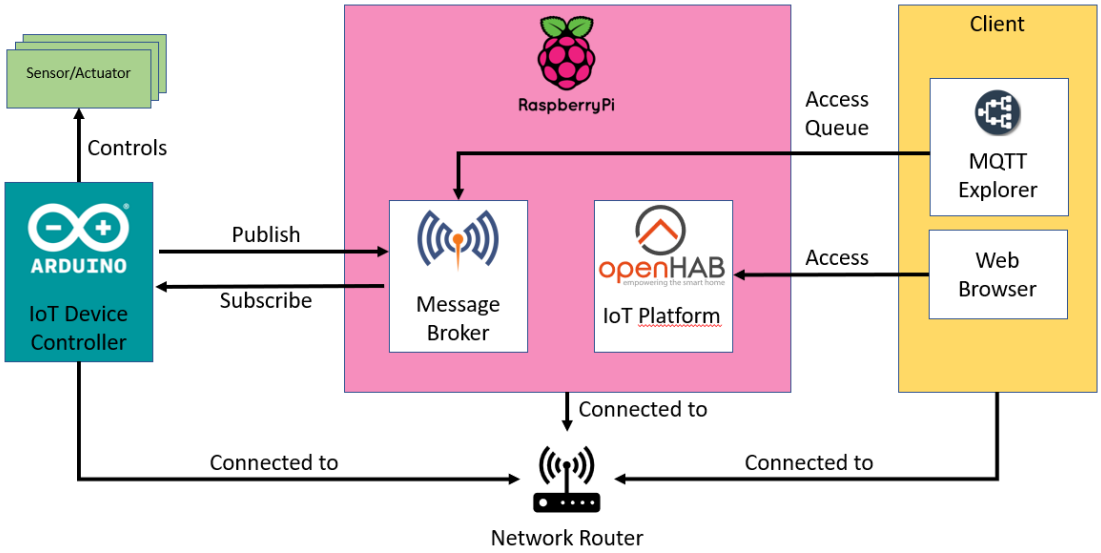


Figure 6.2.: Overview of the experiment environment architecture

Device pin	Arduino pin
Red LED	3
Green LED	4
Servo motor	5
Temperature and Humidity sensor	6
Button	2
Photoresistor	A0

Table 6.1.: The physical connections between the IoT device output pins and the Arduino pins

customizable framework for integrating diverse devices and services seamlessly. Due to its open-source nature and comprehensive APIs, OpenHAB has been selected as the IoT Platform instance utilized for this evaluation case.

6.2.3. IoT Devices

The ecosystem of IoT devices deployed within the smart house environment includes a diverse array of sensors and actuators controlled by an Arduino UNO WIFI microcontroller. The following sensors and actuators are utilized for this case:

- A red LED light
- A green LED light
- A pressable button
- A humidity and temperature sensor (one device for both sensors)
- A photoresistor sensor
- A servo Motor
- An RFID reader

Arduino pinout

The pinout of all devices except the RFID reader is outlined in Table 6.1 and visually represented in Figure 6.3. To simplify the table, the GND pin as well as the 5V pin of all sensors and actuators have been left out. For the RFID reader, its pinout is described separately in Table 6.2, and its visual representation is illustrated in Figure 6.4.

Arduino code

For sensors, each sensor publishes its data to a dedicated topic, allowing subscribers to receive real-time updates on the sensor readings. The implementation of this mechanism can be seen in the following arduino code:

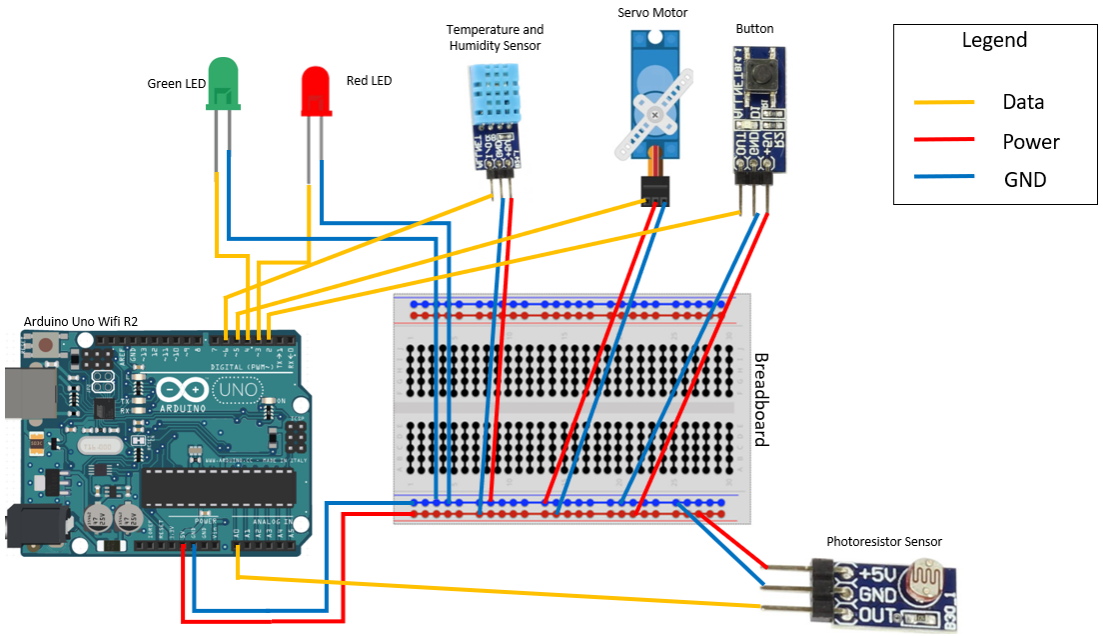


Figure 6.3.: The physical connections between the IoT devices and the Arduino UNO WIFI

RFID-RC522 pin	Arduino pin
3.3v	3.3v
RST	9
GND	GND
SDA	10
MISO	ISCP MISO
SCK	ISCP SCK
MOSI	ISCP MOSI

Table 6.2.: The physical connections between the RFID reader pins and the Arduino pins

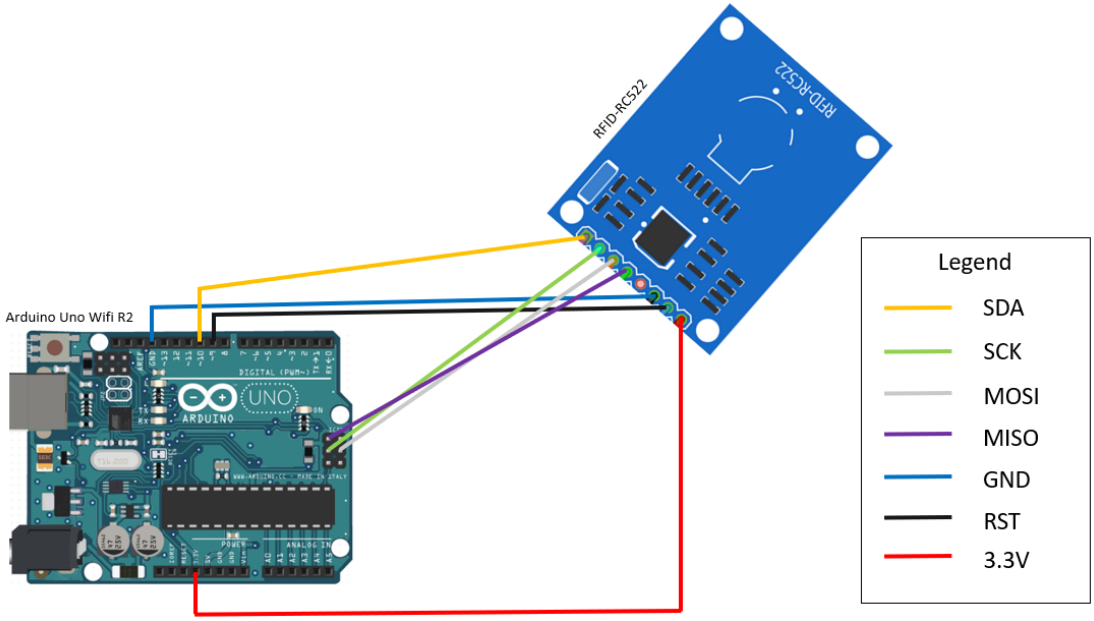


Figure 6.4.: The physical connections between the RFID reader and the Arduino UNO WIFI

```
MqttClient mqttClient(wifiClient);
// MQTT Broker Address
const char broker[] = "192.168.0.100";
int port = 1883;

// MQTT Topics
const char topicTemperature[] = "IoTDevice/Temperature";
const char topicHumidity[] = "IoTDevice/Humidity";

// DHT Sensor Configuration
DHT dht(6, DHT11); // Pin 6 connected to the DHT sensor

void setup() {
    // Connect to the WIFI Network
    // ...

    //Start the DHT Sensor reading
    dht.begin();
    // Attempt to connect to the MQTT broker
    Serial.print("Attempting to connect to the MQTT broker: ");
    Serial.println(broker);
    while (!mqttClient.connect(broker, port)) {
      Serial.print("MQTT connection failed! Error code = ");
      Serial.println(mqttClient.connectError());
```

```
25        delay(5000);
26      }
27      Serial.println("You're connected to the MQTT broker!");
28      Serial.println();
29  }
30
31  void loop() {
32      // Read temperature and humidity from the DHT sensor
33      float temperature = dht.readTemperature();
34      float humidity = dht.readHumidity();
35
36      // Publish temperature and humidity readings to dedicated
        topics
37      mqttClient.beginMessage(topicTemperature);
38      mqttClient.print(temperature);
39      mqttClient.endMessage();
40
41      mqttClient.beginMessage(topicHumidity);
42      mqttClient.print(humidity);
43      mqttClient.endMessage();
44  }
```

Listing 6.1: Example arduino code used for sensors

Since actuators do not typically generate data, they instead publish a message indicating their readiness and a list of available commands. Subscribers can then send control commands to the actuators via the specified topic, enabling remote management and operation. In this work, we assume the following format for the data published by the actuators: "init[list of commands separated by commas]". This standardized format allows external systems to control the actuator with correct commands without needing to inspect the code. The following code provides an example implementation of the mentioned mechanism for the servo motor:

```
1  #include <Servo.h>
2
3  MqttClient mqttClient(wifiClient);
4  // MQTT Broker Address
5  const char broker[] = "192.168.0.100";
6  int port = 1883;
7
8  // MQTT Topics
9  const char topicServoLid[] = "IoTDevice/Servo/Lid";
10
11 // Servo Configuration
12 Servo poolLid;
13
14 void setup() {
15     // Connect to the WIFI Network
16     // ...
```

```
17
18      // Attempt to connect to the MQTT broker
19      // ...
20
21      // Initialize Servo
22      poolLid.attach(servo_pin);
23
24      // Subscribe to topics
25      mqttClient.subscribe(topicServoLid);
26      mqttClient.beginMessage(topicServoLid);
27      mqttClient.print("init{1,0}");
28      mqttClient.endMessage();
29  }
30
31  void loop() {
32      // Call poll() regularly to allow the library to receive MQTT
        messages and
33      // send MQTT keep alive to avoid being disconnected by the
        broker
34      mqttClient.poll();
35
36      mqttClient.beginMessage(topicServoLid);
37      mqttClient.print("init{1,0}");
38      mqttClient.endMessage();
39  }
40
41  void onMqttMessage(int messageSize) {
42      // We received a message
43      String receivedMessage = ""; // Define a string to hold the
        received message
44
45      while (mqttClient.available()) {
46          char message = (char)mqttClient.read();
47          Serial.println(message);
48
49      // Append the received character to the message string
50      receivedMessage += message;
51      }
52      // Process the message only if it contains '0' or '1'
53      if (receivedMessage == "0" || receivedMessage == "1") {
54          // In case the message is for the Servo
55          else if (strcmp(topic, topicServoLid) == 0) {
56              if (receivedMessage == "0") {
57                  poolLid.write(100);
58                  Serial.print("--> Lid closed");
59              } else if (receivedMessage == "1") {
60                  poolLid.write(90);
61                  Serial.print("--> Lid opened");
62              }
```

```
63          }
64      }
65      // Clear the received message for the next message
66      receivedMessage = "";
67  }
```

Listing 6.2: Example arduino code used for actuators

The complete code used for the experiment environment can be found in Section A.1 of the appendix.

6.3. Implementation: The Instantiator Pipeline

The Instantiator pipeline implements the IoT Platform configuration approach proposed in chapter 5. Certain steps related to scenario knowledge graphs and the physical elements have been omitted to streamline the evaluation process and maintain a concentrated focus on IoT Platform configuration. Consequently, the knowledge graph utilized for enriching the IoT Device fingerprint exclusively contains the IoT Platform knowledge graph tailored for OpenHAB, which has been pre-established. The scope for the implementation is visualized in Figure 6.5. This deliberate simplification ensures that the evaluation remains tightly aligned with the primary objective of assessing IoT Platform configuration effectiveness without introducing further variables or complexities. The full implementation of the Instantiator pipeline can be found on the Gitlab instance of OMiLAB[1].

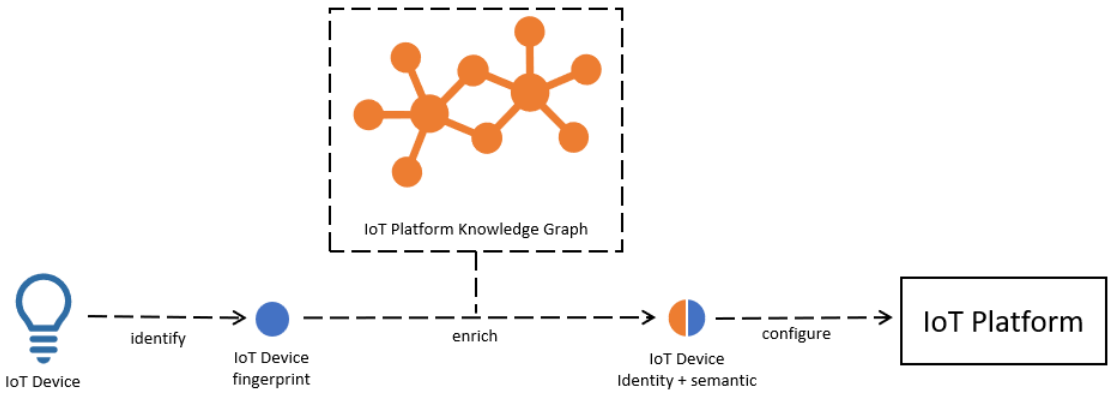


Figure 6.5.: The IoT Device configuration approach applied for the evaluation case

Therefore, the implementation contains three distinct modules, with each module aligned with one of the three steps essential for transitioning from an IoT Device to a digital twin object within an IoT Platform:

- *Network Sniffer*: Responsible for gathering and processing network traffic.

[1] https://code.omilab.org/danialm98/instantiator-pipeline

- *NeSy-Transformer*: Responsible for enhancing gathered data with semantic knowledge using a neurosymbolic algorithm.
- *OpenHAB Controller*: Interface to the openhab instance.

The technical architecture of the Instantiator is visualized in Figure 6.6. In the following sections, each module will be detailed and its functionality explained.

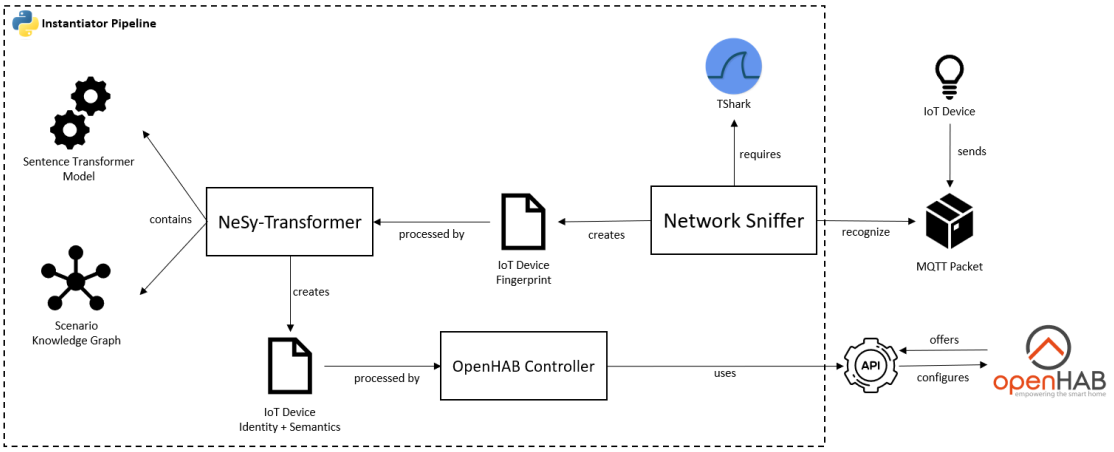


Figure 6.6.: Technical architecture of the Instantiator

6.3.1. Network Sniffer

The network sniffer is tasked with capturing and analyzing network traffic within the experiment environment. This module leverages the Pyshark library [38], a Python wrapper for TShark. TShark, in turn, is a terminal-oriented version of Wireshark, a powerful network packet analyzer tool used for intercepting and inspecting traffic traversing a network [45].

The main function of the network sniffer includes a loop that filters all network traffic to identify packets utilizing the MQTT protocol. An example of a packet gathered by the network sniffer is provided below:

```
192.168.0.101:52911   192.168.0.100:1883  MQTT  119
    Publish Message [IoTDevice/Lamp/Green],
    Publish Message [IoTDevice/Photoresistor]
```

Listing 6.3: Example packets received by Network Sniffer

This packet contains details regarding the source and destination of the MQTT messages, as well as the topics and message contents. This information is valuable for identifying potential MQTT brokers and individual IoT devices transmitting data. For instance, from the given message, the system can identify the address 192.168.0.100:1883 as an MQTT broker, as it is receiving publish messages. It is important to mention, that the

system ignores MQTT packages with the destination port between 49152 and 65535 since this range is commonly used as dynamic or ephemeral ports.

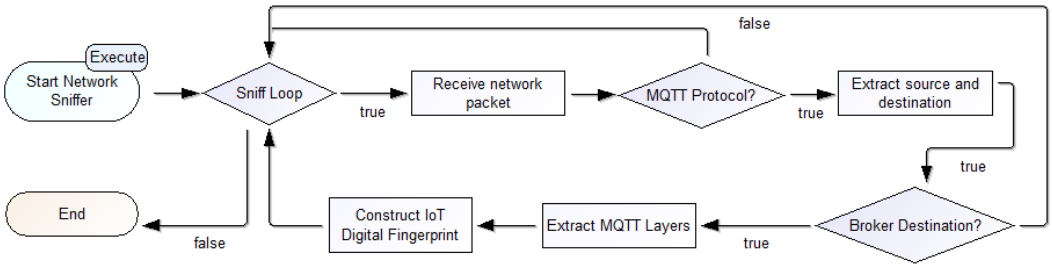


Figure 6.7.: The defined process in the Network Sniffer module.

Furthermore, the example packet includes two publish messages to the topics "IoT-Device/Lamp/Green" and "IoTDevice/Photoresistor" with message contents "init1,0" and "0", respectively. These messages represent data sent by IoT devices. The system recognizes each device and stores them in a map object, with the topic of the data serving as the key. Figure 6.7 further visualizes the system's process. The output provided by the system, referred to as the digital fingerprint of the IoT Device in chapter 5, follows a strict format, as shown in the example below:

```
{
    'IoTDevice/Photoresistor': {
        'messages': ['1','1'],
        'active': True,
        'last_activity': [TIMESTAMP],
        'broker': '192.168.0.100:1883'
    },
    'IoTDevice/Lamp/Green': {
        'messages': ['init{1,0}','init{1,0}'],
        'active': True,
        'last_activity': [TIMESTAMP],
        'broker': '192.168.0.100:1883'
    }
}
```

Listing 6.4: Example output generated by the Netwrok Sniffer

This data structure encapsulates all relevant information gathered by the system through packet sniffing. The 'messages' field stores the last 10 messages sent by each device. The 'last_activity' field is utilized by a scheduler within the system to monitor the activity of IoT devices. If the system fails to receive a message from a device within a certain timeframe (default 10 seconds), the device is labeled as inactive and the 'active' field is set to False.

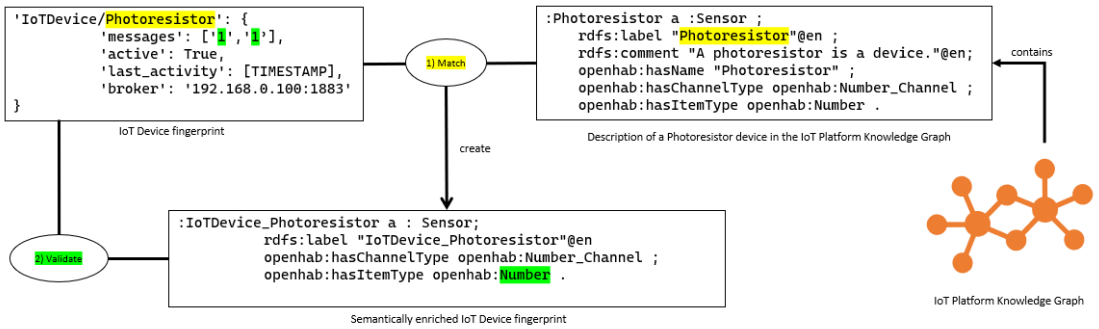


Figure 6.8.: The process of semantically enhancing the device fingerprint.

6.3.2. NeSy-Transformer

The NeSy-transformer module is tasked with semantically enriching the digital fingerprints of the IoT devices generated by the Network Sniffer using a neurosymbolic approach. This process involves attempting to find a device similar to the one being processed within a predefined knowledge graph using a sentence transformer deep learning model. The knowledge graph defines the concept of devices and various device types, along with their corresponding counterparts in OpenHAB. By matching the digital fingerprint to a concept within the knowledge graph, reasoning techniques can be applied to create a data structure necessary for creating a DTwin of the object within the OpenHAB instance.

Alignment knowledge graph

At the core of the knowledge graph lies the device concept, characterized by a name and belonging to the type openhab:Item. An Item in OpenHAB represents a unit of device and includes a datatype attribute. Within the device concept, there are two subclasses: sensor and actuator. The sensor Item features a channel attribute, which links it to a data channel. The channel also possesses a datatype attribute, which should align with the datatype of the device (Item). Additionally, the knowledge graph encompasses instances of devices (sensors and actuators), each defined by a name, label, description, ItemType, and, in the case of sensors, a channelType. The full knowledge graph used for this evaluation case can be found in Section A.2 of the appendix chapter.

Similarity matching

By identifying the device instance most similar to our digital fingerprint, we can utilize inference to extract all relevant information related to our device. This information will be crucial for creating a DTwin of the device in OpenHAB. To match the digital fingerprint of the IoT device with the best-fitting counterpart within the knowledge graph, we employed the paraphrase-MiniLM-L6-v2 sentence transformer model [41]. Additionally, to ensure the accuracy of the selected knowledge graph device, we compare the datatype

of the received messages with the datatype defined for the counterpart in the knowledge graph. An example of this process is visualized in Figure 6.8. The output produced for the example in figure 6.8 is the json below:

```
'IoTDevice/Photoresistor': {
    'device_type': {'Sensor': 'http://omilab.org/IoT#Sensor'},
    'device_datatype': {'Number':
   'http://omilab.org/openhab#Number'},
    'most_similar': 'Photoresistor','data': ['1', '1'],
    'states': None,'communication': 'mqtt',
    'broker': '192.168.0.100:1883',
    'topic': 'IoTDevice/Photoresistor',
    'channel_datatype': {'Number':
   'http://omilab.org/openhab#Number_Channel'}
}
```

Listing 6.5: Example output generated for a sensor by the NeSy-Transformer

For actuators, the output differs slightly since they do not require a channel but rather possible states. These states are extracted from the messages by parsing the values within the brackets and reading the potential states separated by commas. An example of the output provided for an actuator is as follows:

```
'IoTDevice/Servo/Lid': {
    'device_type': {'Actuator': 'http://omilab.org/IoT#Actuator'},
    'device_datatype': {'Switch':
   'http://omilab.org/openhab#Switch'},
    'most_similar': 'Servo',
    'data': ['init{1,0}', 'init{1,0}', 'init{1,0}'],
    'states': ['1', '0'],
    'communication': 'mqtt',
    'broker': '192.168.0.100:1883',
    'topic': 'IoTDevice/Servo/Lid',
    'channel_datatype': None
}
```

Listing 6.6: Example output generated for an actuator by the NeSy-Transformer

This output includes vital details such as device datatype, broker address, MQTT topic, and channel type, all of which are fundamental for generating a DTwin of an IoT device within OpenHAB.

6.3.3. OpenHAB Controller

The OpenHAB Controller serves as the interface between the Instantiator and the OpenHAB instance. This service is responsible for creating and configuring DTwins of IoT devices recognized and prepared by the previous steps of the pipeline, as well as removing inactive IoT devices. The service comprises two main processes: item creation and item deletion, each with two variations for sensors and actuators. In the following subsections, both main processes will be explained in further detail.

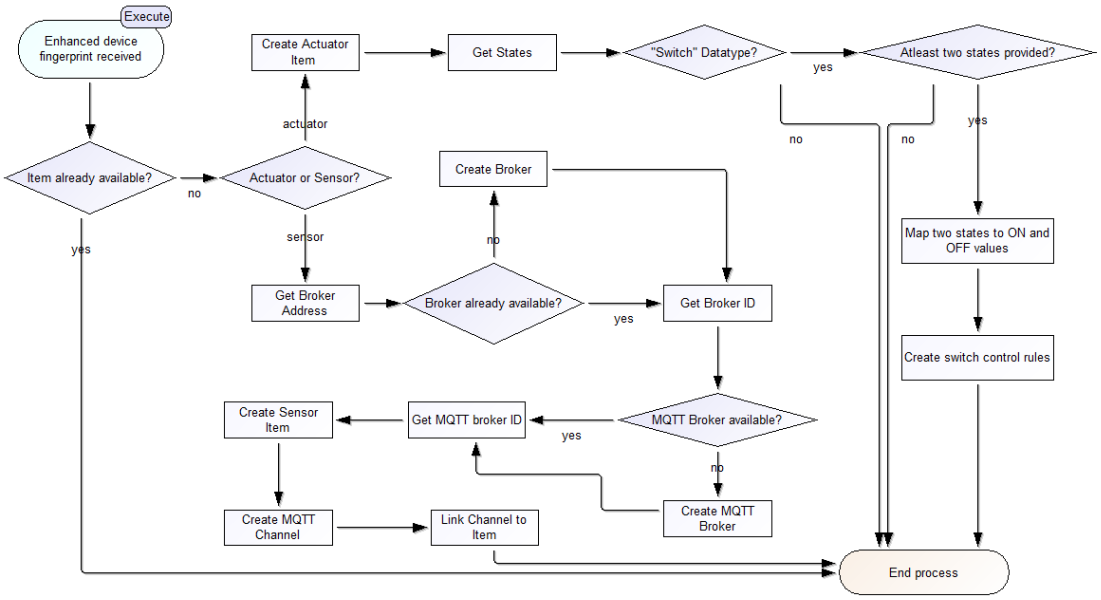


Figure 6.9.: Item creation pipeline in the OpenHAB controller

Item Creation

The item creation pipeline is responsible for generating DTwins of the recognized devices in the form of OpenHAB items. These items serve as virtual representations of the physical IoT devices within the OpenHAB ecosystem. However, for an item to be considered a DTwin, as [48], it needs to mimic and control its physical counterpart through real-time data exchange. Since items alone can only represent the state or value of a physical device, additional components are required to enable the data exchange.

Figure 6.9 illustrates the process from receiving a semantically enhanced device fingerprint to having a fully functional DTwin of the device in OpenHAB. As mentioned earlier, the process of creating an item differs between sensor and actuator devices.

Sensor devices necessitate a Channel component (Thing) linked to the item, which receives MQTT messages from the physical twin. This component requires not only the MQTT topic but also a Broker component (Thing) to establish a connection to the MQTT broker. Both of these components can be created using the information added to the device fingerprint during the semantic enhancement process.

Actuators, on the other hand, do not require a Channel component since no real-time data is published by the physical device. Instead, control functionalities are necessary to communicate state changes applied in the DTwin to the physical counterpart. These functionalities can be implemented using Rule components in OpenHAB. In the OpenHAB Controller service, control rules can be automatically created for devices with the Switch datatype. A switch item can have two states: ON and OFF. By matching these states to the possible states of the physical device, available in the enhanced device fingerprint,

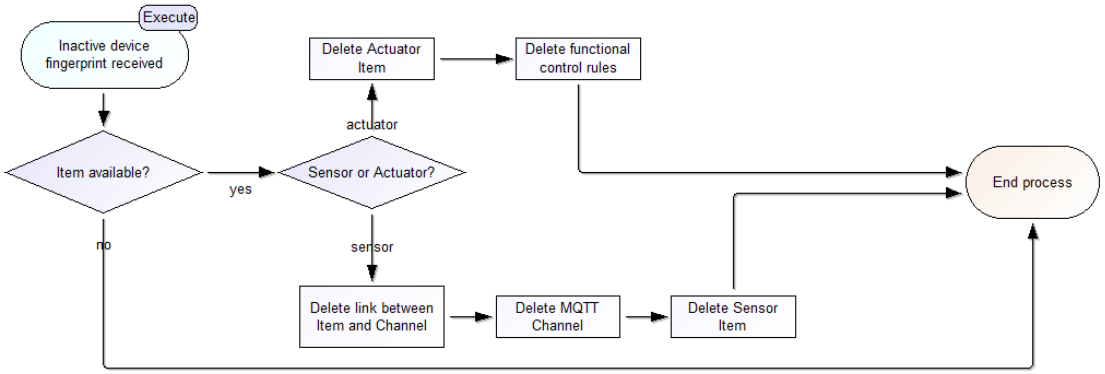


Figure 6.10.: Item deletion pipeline in the OpenHAB controller

two rules can be generated for switching between the two possible states using the DTwin. These rules are labeled with the "Functional" tag to distinguish them from other rules available in the OpenHAB instance.

Item Deletion

The item deletion process within the OpenHAB Controller service is responsible for removing DTwins of IoT devices that are no longer active or present within the environment. These items can be recognized by the value of the "active" key in the device fingerprint. This ensures that the OpenHAB instance remains synchronized with the physical IoT device landscape and prevents the accumulation of redundant or outdated DTwins.

Similar to the item creation pipeline, the item deletion process also varies depending on whether it involves deleting sensors or actuators. This distinction ensures that the deletion process is tailored to the specific characteristics and requirements of each device type. When deleting a sensor item, it is necessary to remove both the associated item and the channel responsible for receiving MQTT messages. On the other hand, when deleting an actuator item, the focus is on removing the functional rules responsible for controlling the device's state. The described process paths are visualized in Figure 6.10.

6.3.4. Results

The results of this implementation for the evaluation case involving the physical experiment environment mentioned above, which includes the DTwins of the sensors and actuators along with the necessary channels and rules, can be observed in Figures 6.11, 6.12, and 6.13.

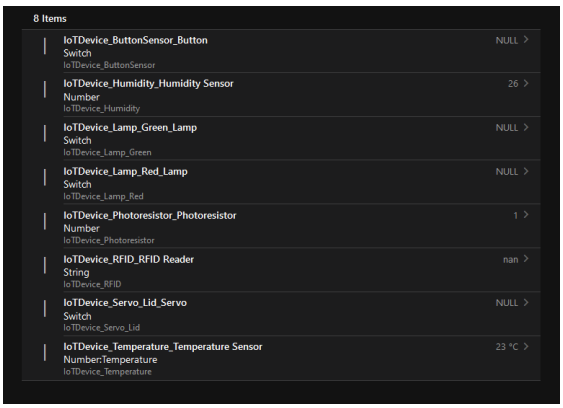


Figure 6.11.: Items generated by the Instantiator in OpenHAB

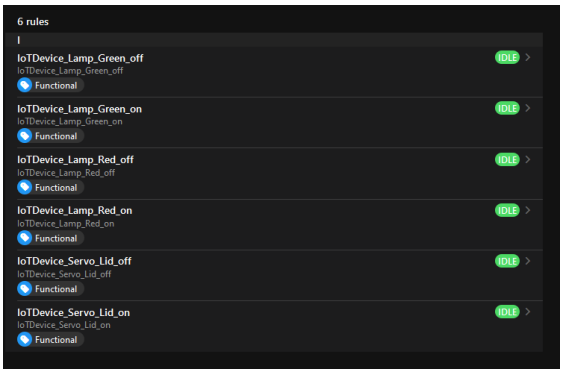


Figure 6.12.: Rules generated by the Instantiator in OpenHAB

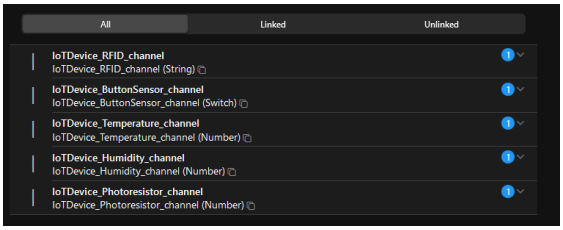


Figure 6.13.: Channels generated by the Instantiator in OpenHAB

7. Further Research

The approach presented in this thesis contributes significantly to advancing IoT platform configuration methodologies. By introducing an approach that harnesses semantic enrichment and neurosymbolic reasoning, the framework offers a promising solution to automate the process of DTwin generation. This approach represents a paradigm shift in how IoT environments are configured, moving away from manual and error-prone methods towards a more systematic and efficient approach. The demonstrated prototype serves as a proof of concept for the proposed approach, highlighting its feasibility and potential for practical implementation.

Considering these advancements, several questions emerge regarding the further utilization and implications of this framework:

- Interoperability: To what extent does the framework facilitate interoperability between different IoT devices, platforms, and standards?
- Integration with Existing Systems: How easily can the framework integrate with existing IoT systems and infrastructure?
- Cross-Domain Integration: How can the framework be extended to facilitate cross-domain integration, enabling seamless interoperability between IoT systems and other emerging technologies such as artificial intelligence, blockchain, or edge computing?

Looking ahead, there are several avenues for future research and development. One direction involves extending the Instantiator pipeline to support additional IoT platforms and scenarios. This expansion would broaden the applicability of the framework, making it more versatile and adaptable to diverse environments and use cases. Additionally, efforts can be made to enhance the scalability, robustness, and flexibility of the framework, ensuring its effectiveness in large-scale IoT deployments and dynamic environments.

Moreover, extending beyond its immediate application in IoT platform configuration, the framework holds significant promise for more general implications across various domains. By establishing abstract representations of physical environments through DTwins, the framework enables a seamless connection between physical IoT devices and conceptual models, similar to methodologies presented in [2]. This connection opens doors to leveraging domain-specific conceptual modeling techniques in diverse sectors such as smart cities, industrial automation, and healthcare. The integration of IoT technologies with domain-specific conceptual models not only advances technological capabilities but also facilitates the seamless integration of IoT solutions into real-world systems and applications, leading to enhanced efficiency, innovation, and ultimately, the improvement of quality of life for users.

8. Conclusion

This thesis has presented a comprehensive approach to the configuration of IoT platforms, focusing on the automatic generation of DTwins for IoT devices within the context of a smart house scenario. The methodology adopted follows the DSMR, providing a structured framework for conducting research aimed at designing and evaluating IT artifacts to address identified problems. The proposed framework in Chapter 5 integrates insights from the fields of IoT, semantic enrichment, and neurosymbolic reasoning, offering a systematic approach to align physical environments hosting IoT devices with specific scenarios. The implementation of the proposed approach was demonstrated through a prototypical instantiation pipeline, which was subsequently evaluated in a physical experiment environment representing a smart house scenario.

The prototypical implementation, termed the Instantiator pipeline, comprises three main modules: the Network Sniffer, the NeSy-transformer, and the OpenHAB Controller. The Network Sniffer module is responsible for capturing and analyzing network traffic to identify IoT devices and gather relevant data about their behavior. The NeSy-transformer module semantically enriches the gathered data using a neurosymbolic approach, matching IoT devices to concepts within a predefined knowledge graph and extracting essential information for creating DTwins. Finally, the OpenHAB Controller module interfaces with the OpenHAB instance to create and configure DTwins based on the semantically enhanced device data.

The evaluation case focused on a smart house scenario, providing a real-world context for testing the effectiveness and functionality of the Instantiator pipeline. The implementation successfully generated DTwins for both sensors and actuators present within the smart house environment, demonstrating its ability to automatically configure an IoT platform based on the identified devices and their functionalities. The output of the implementation, including the created items, channels, and rules within the OpenHAB instance, validated the effectiveness and completeness of the proposed approach.

The further research chapter underscores the transformative impact of the proposed framework on IoT platform configuration methodologies, emphasizing its potential to revolutionize the process. The chapter raises relevant questions about the framework's interoperability, integration with existing systems, and potential for cross-domain integration. Looking forward, avenues for future research include extending the Instantiator pipeline to support additional IoT platforms and scenarios, and exploring broader applications beyond IoT configuration. Notably, the framework's promise extends beyond IoT to domains like smart cities, industrial automation, and healthcare, offering a pathway to seamlessly integrate IoT solutions into diverse real-world systems, ultimately enhancing efficiency, innovation, and quality of life for users.

Bibliography

[1] A. Aksoy and M. H. Gunes. Automated IoT Device Identification using Network Traffic. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pages 1–7, Shanghai, China, May 2019. IEEE.

[2] V. Alexander, C. Muck, D. M. Amlashi, and D. Karagiannis. Bridging haptic Design Thinking and cyber-physical environments through Digital Twins using conceptual modeling. In *22nd International Conference on Perspectives in Business Informatics Research (BIR 2023)*, Sept. 2023.

[3] L. B. Archer. Systematic method for designers. *Design*, pages 56–59, 1964.

[4] K. Ashton and others. That 'internet of things' thing. *RFID journal*, 22(7):97–114, 2009. Publisher: Hauppauge, New York.

[5] L. Atzori, A. Iera, and G. Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[6] B. Bezawada, M. Bachani, J. Peterson, H. Shirazi, I. Ray, and I. Ray. Behavioral Fingerprinting of IoT Devices. In *Proceedings of the 2018 Workshop on Attacks and Solutions in Hardware Security*, ASHES '18, pages 41–50, New York, NY, USA, 2018. Association for Computing Machinery. event-place: Toronto, Canada.

[7] D. I. Borissova, V. K. Danev, M. B. Rashevski, I. G. Garvanov, R. D. Yoshinov, and M. Z. Garvanova. Using IoT for Automated Heating of a Smart Home by Means of OpenHAB Software Platform. *IFAC Workshop on Control for Smart Cities CSC 2022*, 55(11):90–95, Jan. 2022.

[8] M. Castelli, L. Vanneschi, and R. Largo. Supervised learning: classification. *por Ranganathan, S., M. Grisbskov, K. Nakai y C. Schönbach*, 1:342–349, 2018.

[9] P. C. Ccori, L. C. C. De Biase, M. K. Zuffo, and F. S. C. Da Silva. Device discovery strategies for the IoT. In *2016 IEEE International Symposium on Consumer Electronics (ISCE)*, pages 97–98, Sao Paulo, Brazil, Sept. 2016. IEEE.

[10] T. Domínguez-Bolaño, O. Campos, V. Barral, C. J. Escudero, and J. A. García-Naya. An overview of IoT architectures, technologies, and existing open-source projects. *Internet of Things*, 20:100626, 2022.

[11] H. L. Dreyfus and S. E. Dreyfus. Making a Mind versus Modeling the Brain: Artificial Intelligence Back at a Branchpoint. *Daedalus*, 117(1):15–43, 1988. Publisher: The MIT Press.

[12] Eclipse Foundation. Mosquitto - An Open Source MQTT Broker.

[13] M. Fahmideh and D. Zowghi. An exploration of IoT platform development. *Information Systems*, 87:101409, 2020.

[14] M. Flasiński. Symbolic Artificial Intelligence. In *Introduction to Artificial Intelligence*, pages 15–22. Springer International Publishing, Cham, 2016.

[15] L. Francis. Unsupervised learning. *Predictive modeling applications in actuarial science*, 1:280–312, 2014. Publisher: Cambridge University Press.

[16] M. Garnelo and M. Shanahan. Reconciling deep learning with symbolic artificial intelligence: representing objects and relations. *Artificial Intelligence*, 29:17–23, Oct. 2019.

[17] C. González García, D. Meana Llorián, C. Pelayo G-Bustelo, and J. M. Cueva-Lovelle. A review about Smart Objects, Sensors, and Actuators. *International Journal of Interactive Multimedia and Artificial Intelligence*, 4(3):7, 2017.

[18] J. Guth, U. Breitenbücher, M. Falkenthal, P. Fremantle, O. Kopp, F. Leymann, and L. Reinfurt. A Detailed Analysis of IoT Platform Architectures: Concepts, Similarities, and Differences. In B. Di Martino, K.-C. Li, L. T. Yang, and A. Esposito, editors, *Internet of Everything: Algorithms, Methodologies, Technologies and Perspectives*, pages 81–101. Springer Singapore, Singapore, 2018.

[19] M. Hassan, H. Guan, A. Melliou, Y. Wang, Q. Sun, S. Zeng, W. Liang, Y. Zhang, Z. Zhang, Q. Hu, Y. Liu, S. Shi, L. An, S. Ma, I. Gul, M. A. Rahee, Z. You, C. Zhang, V. K. Pandey, Y. Han, Y. Zhang, M. Xu, Q. Huang, J. Tan, Q. Xing, P. Qin, and D. Yu. Neuro-Symbolic Learning: Principles and Applications in Ophthalmology, July 2022. arXiv:2208.00374 [cs].

[20] A. Hevner and S. Chatterjee. A Science of Design for Software-Intensive Systems. In *Design Research in Information Systems: Theory and Practice*, pages 63–77. Springer US, Boston, MA, 2010.

[21] M. Hilario. An Overview of Strategies for Neurosymbolic Integration. 1995.

[22] U. Hunkeler, H. L. Truong, and A. Stanford-Clark. MQTT-S - A publish/subscribe protocol for Wireless Sensor Networks. In *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08)*, pages 791–798, Bangalore, India, Jan. 2008. IEEE.

[23] E. Ilkou and M. Koutraki. Symbolic Vs Sub-symbolic AI Methods: Friends or Enemies?

[24] G. James, D. Witten, T. Hastie, R. Tibshirani, and J. Taylor. Unsupervised Learning. In *An Introduction to Statistical Learning: with Applications in Python*, pages 503–556. Springer International Publishing, Cham, 2023.

[25] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4:237–285, May 1996.

[26] Keertikumar M., Shubham M., and R. Banakar. Evolution of IoT in smart vehicles: An overview. In *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, pages 804–809, Greater Noida, Delhi, India, Oct. 2015. IEEE.

[27] M. A. R. Ken Peffers, Tuure Tuunanen and S. Chatterjee. A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3):45–77, 2007. Publisher: Routledge _eprint: https://doi.org/10.2753/MIS0742-1222240302.

[28] D. Kiritsis. Closed-loop PLM for intelligent products in the era of the Internet of things. *Computer-Aided Design*, 43(5):479–501, 2011.

[29] S. Li, L. D. Xu, and S. Zhao. The internet of things: a survey. *Information Systems Frontiers*, 17(2):243–259, Apr. 2015.

[30] T. Lin, Y. Wang, X. Liu, and X. Qiu. A survey of transformers. *AI Open*, 3:111–132, Jan. 2022.

[31] J. McCarthy. Artificial Intelligence, Logic and Formalizing Common Sense. In R. H. Thomason, editor, *Philosophical Logic and Artificial Intelligence*, pages 161–190. Springer Netherlands, Dordrecht, 1989.

[32] V. Nasteski. An overview of the supervised machine learning methods. *HORIZONS.B*, 4:51–62, Dec. 2017.

[33] A. Newell and H. A. Simon. Computer Science as Empirical Inquiry: Symbols and Search. In J. Haugeland, editor, *Mind Design II: Philosophy, Psychology, and Artificial Intelligence*, page 0. The MIT Press, Mar. 1997.

[34] openHAB Foundation e.V. openHAB - a vendor and technology agnostic open source automation software for your home.

[35] J. Z. Pan, S. Razniewski, J.-C. Kalo, S. Singhania, J. Chen, S. Dietze, H. Jabeen, J. Omeliyanenko, W. Zhang, M. Lissandrini, R. Biswas, G. de Melo, A. Bonifati, E. Vakaj, M. Dragoni, and D. Graux. Large Language Models and Knowledge Graphs: Opportunities and Challenges, Aug. 2023. arXiv:2308.06374 [cs].

[36] R. C. Parocha and E. Q. B. Macabebe. Implementation of Home Automation System Using OpenHAB Framework for Heterogeneous IoT Devices. In *2019 IEEE International Conference on Internet of Things and Intelligence System (IoTaIS)*, pages 67–73, 2019.

[37] A. S. Petrenko, S. A. Petrenko, K. A. Makoveichuk, and P. V. Chetyrbok. The IIoT/IoT device control model based on narrow-band IoT (NB-IoT). In *2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, pages 950–953, Moscow, Jan. 2018. IEEE.

[38] K. Pimmel. PyShark: Python wrapper for tshark, allowing python packet parsing using wireshark dissectors, 2024.

[39] R. Porkodi and V. Bhuvaneswari. The Internet of Things (IoT) Applications and Communication Enabling Technology Standards: An Overview. In *2014 International Conference on Intelligent Computing Applications*, pages 324–329, Coimbatore, India, Mar. 2014. IEEE.

[40] P. R. J. Pêgo and L. Nunes. Automatic discovery and classifications of IoT devices. In *2017 12th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–10, 2017.

[41] N. Reimers and I. Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Nov. 2019.

[42] K. Rose, S. Eldridge, and L. Chapin. The internet of things: An overview. *The internet society (ISOC)*, 80:1–50, 2015. Publisher: Reston, VA.

[43] G. Ryle. *The Concept of Mind: 60th Anniversary Edition*. Routledge, 1 edition, May 2009.

[44] S. Saxena, S. Jain, D. Arora, and P. Sharma. Implications of MQTT Connectivity Protocol for IoT based Device Automation using Home Assistant and OpenHAB. In *2019 6th International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 475–480, Mar. 2019. Journal Abbreviation: 2019 6th International Conference on Computing for Sustainable Global Development (INDIACom).

[45] C. Sanders. *Practical packet analysis: Using Wireshark to solve real-world network problems*. No Starch Press, 2017.

[46] S. S. Shriyal and B. S. Ainapure. IoT Device Classification Techniques and Traffic Analysis - A Review. In *2021 International Conference on Technological Advancements and Innovations (ICTAI)*, pages 244–250, Tashkent, Uzbekistan, Nov. 2021. IEEE.

[47] M. Sikimic, M. Amovic, V. Vujovic, B. Suknovic, and D. Manjak. An Overview of Wireless Technologies for IoT Network. In *2020 19th International Symposium INFOTEH-JAHORINA (INFOTEH)*, pages 1–6, East Sarajevo, Bosnia and Herzegovina, Mar. 2020. IEEE.

[48] M. Singh, E. Fuenmayor, E. Hinchy, Y. Qiao, N. Murray, and D. Devine. Digital Twin: Origin to Future. *Applied System Innovation*, 4(2):36, May 2021.

[49] B. Stiller, E. Schiller, C. Schmitt, S. Ziegler, and M. James. An overview of network communication technologies for IoT. *Handbook of Internet-of-Things*, 12, 2020. Publisher: Springer Int.

[50] S. Tang, D. R. Shelden, C. M. Eastman, P. Pishdad-Bozorgi, and X. Gao. A review of building information modeling (BIM) and the internet of things (IoT) devices integration: Present status and future trends. *Automation in Construction*, 101:127–139, 2019.

[51] A. Voelz, D. M. Amlashi, and M. Lee. Semantic Matching Through Knowledge Graphs: A Smart City Case. In M. Ruiz and P. Soffer, editors, *Advanced Information Systems Engineering Workshops*, pages 92–104, Cham, 2023. Springer International Publishing.

A. Appendix

A.1. Arduino Code

```
#include <Servo.h>
#include <ArduinoMqttClient.h>
#include <WiFiNINA.h>
#include <DHT.h>
// RFID
#include <SPI.h>
#include <MFRC522.h>

// WiFi and MQTT Configuration
char ssid[] = "TP-Link_9C3A";        // your network SSID
char pass[] = "18994964";    // your network password

WiFiClient wifiClient;
MqttClient mqttClient(wifiClient);

const char broker[] = "192.168.0.100";
int port = 1883;

// MQTT Topics
const char topicRedLight[] = "IoTDevice/Lamp/Red";
const char topicGreenLight[] = "IoTDevice/Lamp/Green";
const char topicServoLid[] = "IoTDevice/Servo/Lid";
const char topicButton[] = "IoTDevice/ButtonSensor";
const char topicTemperature[] = "IoTDevice/Temperature";
const char topicHumidity[] = "IoTDevice/Humidity";
const char topicPhotoresistor[] = "IoTDevice/Photoresistor";
const char topicRFID[] = "IoTDevice/RFID";

// Pin Configuration
const int light_red = 3;
const int light_green = 4;
const int servo_pin = 5;
const int button_pin = 2;  // Pin connected to the touch sensor
const int photoresistor_pin = A0;  // Pin connected to the
   photoresistor

// DHT Sensor Configuration

DHT dht(6, DHT11); // Pin 6 connected to the DHT sensor
```

```
39
40 // RFID Definition
41 #define SS_PIN 10
42 #define RST_PIN 9
43 MFRC522 mfrc522(SS_PIN, RST_PIN);   // Create MFRC522 instance.
44
45 Servo poolLid;
46 bool recognised=false;
47
48 void setup() {
49   // Initialize servo and pin modes
50   poolLid.attach(servo_pin);
51   poolLid.write(90);
52   pinMode(light_red, OUTPUT);
53   pinMode(light_green, OUTPUT);
54   pinMode(button_pin, INPUT); // Set the button sensor pin as
      input
55
56   dht.begin();
57
58   // Initialize serial and wait for port to open
59   Serial.begin(9600);
60   while (!Serial) {
61     ; // Wait for serial port to connect. Needed for native USB
      port only
62   }
63
64   // Attempt to connect to WiFi network
65   Serial.print("Attempting to connect to SSID: ");
66   Serial.println(ssid);
67   while (WiFi.begin(ssid, pass) != WL_CONNECTED) {
68     // Failed, retry
69     Serial.print(".");
70     delay(5000);
71   }
72   Serial.println("You're connected to the network");
73   Serial.println();
74
75   // Attempt to connect to the MQTT broker
76   Serial.print("Attempting to connect to the MQTT broker: ");
77   Serial.println(broker);
78   while (!mqttClient.connect(broker, port)) {
79     Serial.print("MQTT connection failed! Error code = ");
80     Serial.println(mqttClient.connectError());
81     delay(5000);
82   }
83   Serial.println("You're connected to the MQTT broker!");
84   Serial.println();
85
```

```
86    //Setup RFID Reader
87    SPI.begin();      // Initiate  SPI bus
88    mfrc522.PCD_Init();   // Initiate MFRC522
89
90
91    // Set the message receive callback
92    mqttClient.onMessage(onMqttMessage);
93
94    // Subscribe to topics
95    Serial.print("Subscribing to topic: ");
96    Serial.println(topicRedLight);
97    Serial.println(topicGreenLight);
98    Serial.println(topicServoLid);
99    Serial.println(topicButton);
100   Serial.println();
101
102   mqttClient.subscribe(topicRedLight);
103   mqttClient.beginMessage(topicRedLight);
104   mqttClient.print("init{1,0}");
105   mqttClient.endMessage();
106
107
108   mqttClient.subscribe(topicGreenLight);
109   mqttClient.beginMessage(topicGreenLight);
110   mqttClient.print("init{1,0}");
111   mqttClient.endMessage();
112
113   mqttClient.subscribe(topicServoLid);
114   mqttClient.beginMessage(topicServoLid);
115   mqttClient.print("init{1,0}");
116   mqttClient.endMessage();
117
118
119   mqttClient.beginMessage(topicRFID);
120   mqttClient.print("nan");
121   mqttClient.endMessage();
122
123
124   // Topics can be unsubscribed using:
125   // mqttClient.unsubscribe(topic);
126   Serial.println();
127 }
128
129 void keepActuatorsAlive() {
130   mqttClient.beginMessage(topicRedLight);
131   mqttClient.print("init{1,0}");
132   mqttClient.endMessage();
133
134   mqttClient.beginMessage(topicGreenLight);
```

```
135   mqttClient.print("init{1,0}");
136   mqttClient.endMessage();
137
138   mqttClient.beginMessage(topicServoLid);
139   mqttClient.print("init{1,0}");
140   mqttClient.endMessage();
141 }
142
143 void loop() {
144   delay(500);
145   if (mfrc522.PICC_IsNewCardPresent() == 1 &&
       mfrc522.PICC_ReadCardSerial() == 1) {
146       Serial.print(" Tag UID: ");
147       String rfidUid = "";
148       for (byte i = 0; i < mfrc522.uid.size; i++) {
149         rfidUid += String(mfrc522.uid.uidByte[i] < 0x10 ? "0" :
      "");
150         rfidUid += String(mfrc522.uid.uidByte[i], HEX);
151       }
152       Serial.println(rfidUid);
153       Serial.println("");
154
155       recognised=true;
156
157       mqttClient.beginMessage(topicRFID);
158       mqttClient.print(rfidUid);
159       mqttClient.endMessage();
160
161   }
162
163   //remove the tag id from the MQTT published
164   if (! mfrc522.PICC_ReadCardSerial() && ! recognised ) {
165      mqttClient.beginMessage(topicRFID);
166       mqttClient.print("nan");
167       mqttClient.endMessage();
168   }
169   else {
170      recognised=false;
171   }
172
173
174   mfrc522.PICC_HaltA();
175   mfrc522.PCD_StopCrypto1();
176
177
178   // Call poll() regularly to allow the library to receive MQTT
       messages and
179   // send MQTT keep alive to avoid being disconnected by the
       broker
```

```
180   mqttClient.poll();
181
182   // Check if the button sensor is touched
183   if (digitalRead(button_pin) == false) {
184     // Publish "touch" message to the touch sensor topic
185     mqttClient.beginMessage(topicButton);
186     mqttClient.print("1");
187     mqttClient.endMessage();
188   } else {
189     // Publish "no touch" message to the touch sensor topic
190     mqttClient.beginMessage(topicButton);
191     mqttClient.print("0");
192     mqttClient.endMessage();
193   }
194
195   // Read temperature and humidity from the DHT sensor
196   float temperature = dht.readTemperature();
197   float humidity = dht.readHumidity();
198
199   // Publish temperature and humidity readings to dedicated topics
200   mqttClient.beginMessage(topicTemperature);
201   mqttClient.print(temperature);
202   mqttClient.endMessage();
203
204   mqttClient.beginMessage(topicHumidity);
205   mqttClient.print(humidity);
206   mqttClient.endMessage();
207
208   // Read light level from the photoresistor and publish it to a
       dedicated topic
209   int lightValue = analogRead(photoresistor_pin);
210   int threshold = 500;  // Adjust this threshold value as needed
211   int lightLevel = (lightValue > threshold) ? 1 : 0;
212   mqttClient.beginMessage(topicPhotoresistor);
213   mqttClient.print(lightLevel);
214   mqttClient.endMessage();
215
216   keepActuatorsAlive();
217 }
218
219 void onMqttMessage(int messageSize) {
220   // We received a message, print out the topic and contents
221   Serial.println("Received a message with topic '");
222   String topicStr = mqttClient.messageTopic();
223   char topic[topicStr.length() + 1];
224   topicStr.toCharArray(topic, topicStr.length() + 1);
225   Serial.print(topic);
226   Serial.print("', length ");
227   Serial.print(messageSize);
```

```
228   Serial.println(" bytes:");
229
230   // Use the Stream interface to print the contents
231   String receivedMessage = ""; // Define a string to hold the
       received message
232
233 while (mqttClient.available()) {
234     char message = (char)mqttClient.read();
235     Serial.println(message);
236
237     // Append the received character to the message string
238     receivedMessage += message;
239 }
240
241 // Process the message only if it contains '0' or '1'
242 if (receivedMessage == "0" || receivedMessage == "1") {
243     // In case the message is for the Red Light
244     if (strcmp(topic, topicRedLight) == 0) {
245         if (receivedMessage == "0") {
246             digitalWrite(light_red, LOW);
247             Serial.print("--> Turning red light OFF");
248         } else if (receivedMessage == "1") {
249             digitalWrite(light_red, HIGH);
250             Serial.print("--> Turning red light ON");
251         }
252     }
253     // In case the message is for the Green Light
254     else if (strcmp(topic, topicGreenLight) == 0) {
255         if (receivedMessage == "0") {
256             digitalWrite(light_green, LOW);
257             Serial.print("--> Turning green light OFF");
258         } else if (receivedMessage == "1") {
259             digitalWrite(light_green, HIGH);
260             Serial.print("--> Turning green light ON");
261         }
262     }
263     // In case the message is for the Servo
264     else if (strcmp(topic, topicServoLid) == 0) {
265         if (receivedMessage == "0") {
266             poolLid.write(100);
267             Serial.print("--> Lid closed");
268         } else if (receivedMessage == "1") {
269             poolLid.write(90);
270             Serial.print("--> Lid opened");
271         }
272     }
273 }
274 // Clear the received message for the next message
275 receivedMessage = "";
```

```
276
277
278
279   Serial.println();
280 }
```

A.2. Evaluation Case Knowledge Graph

```
1  @prefix :<http://omilab.org/IoT#> .
2  @prefix openhab:<http://omilab.org/openhab#> .
3  @prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
4  @prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#> .
5  @prefix xsd:<http://www.w3.org/2001/XMLSchema#> .
6
7
8  #------------------------------------ Devices
       ----------------------------------------
9  :Device a rdfs:Class ;
10     rdfs:label "Device"@en ;
11     rdfs:comment "A device is a physical object that can be
       controlled and/or monitored."@en ;
12     a openhab:Item .
13
14 :Sensor a rdfs:Class ;
15     rdfs:label "Sensor"@en ;
16     rdfs:subClassOf :Device .
17
18 :Actuator a rdfs:Class ;
19     rdfs:label "Actuator"@en ;
20     rdfs:subClassOf :Device .
21
22 #------------------------------------ Openhab Items
       ----------------------------------------
23 openhab:Item a rdfs:Class ;
24     rdfs:label "Item"@en ;
25     rdfs:comment "An item is a device that can be controlled
       and/or monitored."@en .
26
27 openhab:hasName rdf:type rdf:Property ;
28     rdfs:label "has name" ;
29     rdfs:comment "Denotes the name of an OpenHAB item." ;
30     rdfs:domain openhab:Item ;
31     rdfs:range xsd:string .
32
33 openhab:hasItemType rdf:type rdf:Property ;
```

```
34      rdfs:label "has type" ;
35      rdfs:comment "Denotes the type of an OpenHAB item, such as
        'Switch', 'Number', 'String', etc." ;
36      rdfs:domain openhab:Item ;
37      rdfs:range openhab:Type .
38
39  #---------------------------------------- Types
        ----------------------------------------
40  openhab:Type a rdfs:Class ;
41      rdfs:label "Type"@en ;
42      rdfs:comment "A type is a classification of an OpenHAB item,
        such as 'Switch', 'Number', 'String', etc."@en .
43
44  openhab:Switch a openhab:Type ;
45      rdfs:label "Switch"@en ;
46      rdfs:comment "A switch is a type of OpenHAB item that can be
        turned on or off."@en;
47      openhab:hasState "ON" ;
48      openhab:hasState "OFF" .
49
50  openhab:Number a openhab:Type ;
51      rdfs:label "Number"@en ;
52      rdfs:comment "A number is a type of OpenHAB item that can be
        set to a numerical value."@en .
53
54  openhab:Number_Temperature a openhab:Type ;
55      rdfs:label "Number:Temperature"@en ;
56      rdfs:comment "A temperature is a type of OpenHAB item that
        can be set to a temperature value."@en .
57
58  openhab:String a openhab:Type ;
59      rdfs:label "String"@en ;
60      rdfs:comment "A string is a type of OpenHAB item that can be
        set to a string value."@en .
61
62  #---------------------------------------- Link
        ----------------------------------------
63  openhab:Channel a rdfs:Class ;
64      rdfs:label "Channel"@en ;
65      rdfs:comment "A channel is a connection to a device that can
        be controlled and/or monitored."@en .
66
67  openhab:hasChannelType rdf:type rdf:Property ;
68      rdfs:label "has channel type" ;
69      rdfs:comment "Denotes the type of an OpenHAB channel, such as
        'Switch', 'Number', 'String', etc." ;
70      rdfs:domain openhab:Channel ;
71      rdfs:range openhab:ChannelType .
72
```

```
openhab:hasTopic rdf:type rdf:Property ;
    rdfs:label "has topic" ;
    rdfs:comment "Denotes the topic of an OpenHAB channel." ;
    rdfs:domain openhab:Channel ;
    rdfs:range xsd:string .

openhab:hasBridge rdf:type rdf:Property ;
    rdfs:label "has bridge" ;
    rdfs:comment "Denotes the bridge of an OpenHAB channel." ;
    rdfs:domain openhab:Channel ;
    rdfs:range openhab:Bridge .

openhab:ChannelType a rdfs:Class ;
    rdfs:label "Channel Type"@en ;
    rdfs:comment "A channel type is a classification of an
    OpenHAB channel, such as 'Switch', 'Number', 'String',
    etc."@en .

openhab:Switch_Channel a openhab:ChannelType ;
    rdfs:label "Switch"@en ;
    rdfs:comment "A switch is a type of OpenHAB channel that can
    be turned on or off."@en .

openhab:Number_Channel a openhab:ChannelType ;
    rdfs:label "Number"@en ;
    rdfs:comment "A number is a type of OpenHAB channel that can
    be set to a numerical value."@en .

openhab:String_Channel a openhab:ChannelType ;
    rdfs:label "String"@en ;
    rdfs:comment "A string is a type of OpenHAB channel that can
    be set to a string value."@en .

openhab:Bridge a rdfs:Class ;
    rdfs:label "Bridge"@en ;
    rdfs:comment "A bridge is a connection to a device that can
    be controlled and/or monitored."@en .

openhab:hasIdentifier rdf:type rdf:Property ;
    rdfs:label "has identifier" ;
    rdfs:comment "Denotes the identifier of an OpenHAB bridge." ;
    rdfs:domain openhab:Bridge ;
    rdfs:range xsd:string .

openhab:hasThingType rdf:type rdf:Property ;
    rdfs:label "has thing type" ;
    rdfs:comment "Denotes the type of an OpenHAB bridge, such as
    'mqtt', 'hue', etc." ;
    rdfs:domain openhab:Bridge ;
```

```
115     rdfs:range openhab:ThingType .
116
117 openhab:ThingType a rdfs:Class ;
118     rdfs:label "Thing Type"@en ;
119     rdfs:comment "A thing type is a classification of an OpenHAB
        bridge, such as 'mqtt', 'hue', etc."@en .
120
121 openhab:mqtt a openhab:ThingType ;
122     rdfs:label "MQTT"@en ;
123     rdfs:comment "MQTT is a type of OpenHAB bridge that can be
        connected to an MQTT broker."@en .
124
125 openhab:hasBroker rdf:type rdf:Property ;
126     rdfs:label "has broker" ;
127     rdfs:comment "Denotes the broker of an OpenHAB bridge." ;
128     rdfs:domain openhab:Bridge ;
129     rdfs:range openhab:Broker .
130
131 openhab:Broker a rdfs:Class ;
132     rdfs:label "Broker"@en ;
133     rdfs:comment "A broker is a connection to an MQTT broker."@en
        ;
134     openhab:hasHost  xsd:string .
135
136 #---------------------------------------- Instance
        ----------------------------------------
137 :Lamp a :Actuator ;
138     rdfs:label "Lamp"@en ;
139     rdfs:comment "A Lamp is a device that can be turned on or
        off."@en;
140     openhab:hasName "Lamp" ;
141     openhab:hasItemType openhab:Switch .
142
143 :Temperature_Sensor a :Sensor ;
144     rdfs:label "Temperature Sensor"@en ;
145     rdfs:comment "A temperature sensor is a device that can
        measure the temperature."@en;
146     openhab:hasName "TemperatureSensor" ;
147     openhab:hasChannelType openhab:Number_Channel ;
148     openhab:hasItemType openhab:Number_Temperature .
149
150 :Servo a :Actuator ;
151     rdfs:label "Servo"@en ;
152     rdfs:comment "A servo is a device that can be controlled to a
        specific position."@en;
153     openhab:hasName "Servo" ;
154     openhab:hasItemType openhab:Switch .
155
156 :Button a :Sensor ;
```

```
157     rdfs:label "Button"@en ;
158     rdfs:comment "A button is a device that can be pressed."@en;
159     openhab:hasName "Button" ;
160     openhab:hasChannelType openhab:Switch_Channel ;
161     openhab:hasItemType openhab:Switch .
162
163 :Humidity_Sensor a :Sensor ;
164     rdfs:label "Humidity Sensor"@en ;
165     rdfs:comment "A humidity sensor is a device that can measure
        the humidity."@en;
166     openhab:hasName "HumiditySensor" ;
167     openhab:hasChannelType openhab:Number_Channel ;
168     openhab:hasItemType openhab:Number .
169
170 :Photoresistor a :Sensor ;
171     rdfs:label "Photoresistor"@en ;
172     rdfs:comment "A photoresistor is a device that can measure
        the light intensity."@en;
173     openhab:hasName "Photoresistor" ;
174     openhab:hasChannelType openhab:Number_Channel ;
175     openhab:hasItemType openhab:Number .
176
177 :RFID_Reader a :Sensor ;
178     rdfs:label "RFID Reader"@en ;
179     rdfs:comment "An RFID reader is a device that can read RFID
        tags."@en;
180     openhab:hasName "RFIDReader" ;
181     openhab:hasChannelType openhab:String_Channel ;
182     openhab:hasItemType openhab:String .
```