Original Article

Artificial Intelligence and Internet of Things: A Neuro-Symbolic Approach for Automated Platform Configuration

Danial M Amlashi[1] (iD)**, Alexander Voelz[1]** (iD) **and Dimitris Karagiannis[2]**

Abstract

Complex Internet of Things (IoT) environments present significant challenges in device discovery and platform configuration, especially as the number of interconnected devices continues to grow. This research addresses these challenges by leveraging the complementary strengths of symbolic and connectionist artificial intelligence (AI) within a neuro-symbolic system. We propose a comprehensive approach to IoT platform configuration that integrates neuro-symbolic reasoning and conceptual modelling techniques, enhancing both efficiency and explainability. Following the process model of design science research, our work introduces two key artifacts: the *Instantiator Pipeline* and the *IoT2Model* method. The *Instantiator Pipeline* automates the discovery and integration of IoT devices, minimising manual intervention and ensuring seamless interoperability. During the initial iteration of the applied research methodology, it was recognised that effective IoT platform configuration must extend beyond device registration to include the creation of scenario-specific rules. To address this, the *IoT2Model* method was developed, enabling users to define models that represent these rules, thereby tailoring the behaviour of their IoT deployments to meet specific requirements. By utilising existing open technologies, our solution ensures flexibility and adaptability, overcoming the limitations of related works. The proposed neuro-symbolic AI system not only streamlines the configuration process but also provides a robust, scalable and explainable solution for managing complex IoT environments, paving the way for future advancements of intelligent information systems.

1 Introduction

Over the past decade, developments regarding Big Data, Industry 4.0, and the ubiquitous connectivity going along with them have shaped emerging technologies as well as innovations. In this context, the Internet of Things (IoT) has been established as a fundamental paradigm that enables communication between physical objects and their digital counterparts within complex environments. Such environments still pose significant challenges, particularly in terms of device

[1]Doctoral School Computer Science, Faculty of Computer Science, University of Vienna, Vienna, Austria
[2]Department of Knowledge Engineering, Faculty of Computer Science, University of Vienna, Vienna, Austria

Corresponding Author:
Alexander Voelz, Doctoral School Computer Science, Faculty of Computer Science, Währinger Straße 29, 1090 Vienna, Austria.
Email: alexander.voelz@univie.ac.at

discovery and IoT platform configurations. This becomes especially relevant when considering the constantly growing number of interconnected devices and the diversity of communication protocols.

Recent advancements in artificial intelligence (AI) offer promising solutions to these challenges. More precisely, the intersection of symbolic and connectionist AI, known as neuro-symbolic AI, has emerged as a powerful approach with the potential to address complex problems by harnessing the strengths of both paradigms. Building on this notion, the motivation for this research is rooted in the growing need for intelligent systems that seamlessly integrate symbolic reasoning and connectionist capabilities, particularly in the domain of IoT platform configuration.

The primary contribution of this work is the development of a neuro-symbolic system for automated IoT platform configuration. By leveraging the complementary strengths of symbolic and connectionist AI, we propose a system that streamlines the configuration process of IoT platforms by following a design science research approach. The proposed artifact not only improves the efficiency of platform configuration but also provides an explainable AI solution. After the initial design science iteration, it became evident that the configuration of IoT platforms extends beyond the discovery and registration of devices, requiring the creation of scenario-specific rules. Consequently, *IoT2Model* was incorporated as an independent artifact within the research approach. The resulting artifacts, comprising the *Instantiator Pipeline* and the *IoT2Model* method, demonstrate the benefits of combining neuro-symbolic AI and conceptual modelling techniques to streamline the complete configuration process of IoT platforms.

The remainder of this paper is structured as follows: Section 2 provides a comprehensive theoretical background, covering the fundamentals of IoT, neuro-symbolic AI and the development of domain-specific modelling methods. In Section 3, we detail the methodology used in our research, outlining the steps taken to design and implement our neuro-symbolic approach. Section 4 presents the configuration of IoT platforms using the approach, with a focus on the *Instantiator Pipeline* and a corresponding evaluation case. In addition, the *IoT2Model* method is introduced as an extension to the *Instantiator Pipeline*. In Section 5, we discuss the preliminary results and potential future directions of our research. Finally, Section 6 concludes the contributions by summarising key findings and insights.

2   Theoretical Foundations

2.1   *Internet of Things and Internet of Things Platforms*

First proposed by Kevin Ashton in 1999 (Ashton, 2009), the Internet of Things (IoT) is a rapidly evolving paradigm. It involves various objects, like sensors and actuators, which are uniquely identifiable and capable of interacting with each other (Atzori et al., 2010). Such interactions require the sending and receiving of messages of both physical and virtual objects, thus forming a network of integrated things (Kiritsis, 2011). Essentially, IoT combines network communication and computation, allowing interconnected devices to gather, exchange, process and consume data in a given environment (Li et al., 2015).

IoT devices, often referred to as 'things', are the end nodes in IoT networks situated in the real world to gather data and manipulate the environment. These devices can be categorised in a simplified manner as sensors, which measure and transform physical parameters like temperature into electrical signals, and actuators, which perform actions such as controlling motors based on the signals (González García et al., 2017; Siciliano et al., 2009). Additionally, complex devices like single-board computers (e.g., Raspberry Pi, Arduino) can also be considered IoT devices in a more holistic view (Tang et al., 2019). Communication among IoT devices is facilitated by various technologies, including Wi-Fi for high-speed local connections, Bluetooth for short-range communication, LTE for long-distance wireless communication and ZigBee for low-data-rate and high-reliability applications. Other technologies include RFID for peer-to-peer connections, NFC for short-range communication and MQTT for lightweight, topic-based publish/subscribe communication (Hunkeler et al., 2008; Stiller et al., 2020).

In the complex IoT landscape, managing diverse devices and data streams requires a centralised infrastructure, often referred to as *IoT Platform*. An IoT platform, or IoT middleware, acts as an integration layer (Razzaque et al., 2016), creating a digital model[1] of physical devices to mimic and control them in real-time (Singh et al., 2021). This central management enables efficient device handling, monitoring and analysis of IoT deployments through HTTP-based REST APIs (Fahmideh & Zowghi, 2020). Integrating a device with an IoT platform involves establishing its representation to enable real-time data exchange between the physical device and its virtual counterpart (often referred to as *item*). This integration follows various data exchange models, such as the publish model for real-time sensor data reception, the publisher-subscribe model for storing and accessing data via a message broker and the polling model for interval-based data requests (Domínguez-Bolaño et al., 2022). The main advantage of IoT platforms lies in automating the given environment by defining rules that trigger associated actions, thereby enabling users to tailor the behaviour of IoT deployments to suit specific requirements (Domínguez-Bolaño et al., 2022; Fahmideh & Zowghi, 2020).

2.2 Symbolic, Connectionist and Neuro-Symbolic Artificial Intelligence

AI can be categorised into two main streams: symbolic AI and connectionist AI. Symbolic AI focuses on the manipulation of symbols and the use of rule-based systems to emulate human reasoning. It relies on the formal representation of knowledge through symbols and logical rules, enabling the system to perform tasks like problem-solving and planning through heuristic searches. This approach, rooted in the early work of Newell and Simon (Newell & Simon, 2007), emphasises the role of symbolic representation in achieving intelligent behaviour. Knowledge Graphs, commonly referred to as ontologies, form such representations that are used in symbolic AI to model declarative knowledge in the form of static information about facts (Flasiński, 2016). Beyond representing knowledge, the true potential of Knowledge Graphs lies in reasoning and inference to reveal hidden insights that are not immediately apparent. Consequently, it has been argued that Knowledge Graphs must not only cover the structural knowledge representation (i.e., an ontology) but also a corresponding reasoning engine to facilitate inference (Ehrlinger & Wöß, 2016). Symbolic AI systems are, therefore, particularly effective in environments where the knowledge can be explicitly encoded, making them valuable for applications based on Knowledge Engineering approaches (e.g., expert systems) (Feigenbaum, 1984).

Connectionist AI, also known as sub-symbolic AI, has the goal of establishing correlations between input data and output variables by optimising the parameters of a transformation function. This optimisation process is labelled as *learning* within connectionist AI, as exemplified by the subdomains *Machine Learning* and *Deep Learning*. One major advantage of this way of learning is the capability to process large-scale data that is used for improving the performance of the learning system over time (Jordan & Mitchell, 2015). The common learning approaches are supervised, unsupervised and semi-supervised, while new developments often rely on reinforcement learning (Arulkumaran et al., 2017). Among many established learning algorithms, Ray (2019), deep learning algorithms have become increasingly popular due to their application for artificial neural networks (ANNs) (Schmidhuber, 2015) and, most recently, for the transformer model architecture that provides the basis of large language models (LLMs) (Vaswani et al., 2017). The resulting applications with millions up to billions of parameters highlight the capabilities of connectionist AI paradigms to process vast amounts of diverse data.

Neuro-symbolic AI seeks to combine the strengths of both symbolic and connectionist approaches to overcome their individual limitations. This fusion involves integrating the representational power of symbolic systems with the learning capabilities of neural networks. There are two primary strategies for this integration: unified and hybrid approaches (Hilario, 1997). The unified approach seamlessly blends neural and symbolic processing within a single architecture, enabling synergistic interactions and direct embedding of symbolic knowledge into neural structures. In contrast, the hybrid approach maintains modular independence, with neural and symbolic components interacting through defined interfaces, allowing each to focus on their strengths – pattern recognition for neural networks and logical reasoning for symbolic systems (Hilario, 1997). This combination aims to achieve robust, explainable and flexible AI systems capable of handling complex tasks requiring both learning from data and reasoning with explicit knowledge.

2.3 Conceptual Modelling and Domain-Specific Modelling Methods

In the context of this research, conceptual modeling and specifically the development of domain-specific modelling methods (DSMMs), becomes increasingly relevant as we address the requirement R6 later in this contribution (cf. Section 3.2). Conceptual modelling offers powerful means of simplifying complex systems by abstracting them into understandable representations (Mayr & Thalheim, 2021). To achieve such simplifications through the development of a DSMM, two foundational frameworks from the conceptual modelling domain are considered: the generic modelling method framework (GMMF) and agile modelling method engineering (AMME).

DSMMs are tailored to capture the unique characteristics of specific application domains (Karagiannis et al., 2016, 2022). The GMMF provides a structured approach for designing modelling methods, according to which each method consists of a modelling language, a modeling procedure and mechanisms and algorithms (Karagiannis & Kühn, 2002). The modeling language covers the syntax, notation and semantics of the respective modelling method, which are specified as a machine-processable metamodel. The modelling procedure contains the intended process of applying the method. Finally, mechanisms and algorithms can be implemented to provide modelling functionalities that go beyond mere representation purposes, such as model transformation or code generation capabilities (Karagiannis et al., 2016). The development process of a DSMM requires a metamodelling platform, such as ADOxx[2]. The open ADOxx platform allows users to realise DSMMs through the built-in support for GMMF components. In addition, ADOxx provides a low-code-based approach by serving as a compiler for the domain-specific language for modelling method realisation (MM-DSL) called ADOSCRIPT, which can be used to implement method-specific functionalities or to extend the platform's capabilities (Karagiannis, 2024; Völz et al., 2024). Reusable components from over 50 developed modelling tools are available as well, ultimately

enabling rapid prototyping and efficient adaptation to evolving requirements. Lastly, the five phases of the AMME life cycle offer a systematic process model for the development and iterative refinement of DSMMs (Karagiannis, 2015).

To conclude, the GMMF provides a structured approach for designing modelling methods, while AMME provides a comprehensive life cycle for their development and iterative refinement. By leveraging the design principles of the GMMF and AMME, an integrated approach to the agile development of DSMM is established.

3   Methodology

This contribution follows the Design Science Research Methodology (DSRM) proposed by Peffers et al. (2007). The corresponding process model is commonly applied in the context of information systems research, as it provides a structured approach for creating and evaluating design artifacts that address previously identified problems (Wieringa, 2014). Consequently, the problem statement and design requirements of this contribution are presented below.

3.1   Problem Statement

The initial phase of the DSRM process model is concerned with the identification of a specific problem for which a solution has to be motivated (Peffers et al., 2007). In the context of IoT platform configuration and deployment, we base our problem statement on related literature (cf. Section 3.2) from which several challenges regarding the effective management of IoT environments can be derived:

- *Support for Device Discovery*: Automated IoT device identification is an important issue for monitoring devices connected to a certain network (Aksoy & Gunes, 2019). A variety of approaches to device discovery exist for both established and newly developed IoT platforms (cf. Javed et al., 2020). Nevertheless, identifying and integrating diverse devices into the ecosystem of an IoT platform is a complex task that has to be considered independent of existing solutions.
- *Complex Platform Configuration and Deployment*: Building upon the previous challenge, the integration of identified IoT devices is a labor-intensive process if no automated solution is provided. As a result, even platform-independent solutions for dynamic configurations of smart environments have been proposed (Mayer et al., 2016). Still, the manual configuration and deployment of IoT platforms remain complex and labor-intensive.
- *Scalability and Adaptability in Heterogeneous Environments*: The number of IoT devices is anticipated to increase significantly in the near future (Zikria et al., 2021). This growth poses a scalability challenge for current IoT platforms and established device discovery methods (Javed et al., 2020; Shahid et al., 2018). Ensuring that novel approaches can handle diverse device types, communication protocols and environments is a critical challenge for future IoT deployments.
- *Open-Source Principle*: It has been noted that differences exist between the pricing of consumer-oriented and enterprise IoT platform solutions (Babun et al., 2021). In fact, the majority of available options have a specific pricing model and do not follow the open-source principle (Babun et al., 2021; Javed et al., 2020). Adherence to the free and open-source principles can thus pose considerable challenges for systems promoting adaptability, interoperability and widespread adoption.
- *Lack of Support for Conceptual Scenario Modelling*: While existing approaches to IoT platform configuration focus on device discovery and integration, they often lack support for defining and managing conceptual scenarios that govern how devices interact within their environment. This limits the ability of users to tailor IoT deployments to specific requirements or contextual conditions, particularly in dynamic environments.

By addressing the highlighted challenges, our research aims to develop a comprehensive approach for automated IoT platform configuration that integrates conceptual scenario modelling capabilities while also enhancing the discovery, integration and management of IoT devices in a scalable manner. Under these considerations, specific requirements for the design science artifact to be created are derived in the following section.

3.2   Related Works and Requirements Specification

Before deriving design requirements of a solution to the formulated problem statement as part of the second DSRM stage (Peffers et al., 2007), related works addressing similar issues are contrasted for further consideration.

The literature offers a variety of approaches and solutions regarding IoT device discovery and platform configuration, each tackling specific aspects of the problem. However, to the best knowledge of the authors, a holistic pipeline that

Table 1. Overview of Requirements to be Addressed by the Design Artifact for Automating ioT Platform Configuration.

ID	Requirement	Description
R1	Device Discovery	The system must provide mechanisms for seamless identification of IoT devices. It should be able to recognise a wide range of device types automatically, reducing the need for manual configuration.
R2	Platform Configuration	The system must automate the platform configuration and deployment process, which includes setting up communication channels, integrating discovered devices and ensuring device interoperability.
R3	Platform Independence	The system should support modular components that allow to exchange the current IoT platform with minimal effort. This has the purpose of adapting to evolving needs and technologies.
R4	Scalability	The system must be scalable to accommodate an increasing number of devices without performance issues. It should efficiently manage resources and maintain high performance as the network grows.
R5	Openness	The system must follow the open-source principle to be freely available and extensible. This includes supporting open standards and utilising free APIs for developers to extend the system functionality.
R6	Model-based Scenario Specification	The system must support conceptual scenario modelling that governs device interactions. Ultimately, a modelling-based extension should enable user-friendly adaptions of IoT deployments.

integrates device discovery, platform configuration and scalability has not been proposed yet. Subsequently, we summarise relevant works in this domain, highlighting their contributions and limitations.

Several solutions for effective IoT device discovery have been proposed. One of these approaches presents a method for device identification using device fingerprinting techniques based on deep learning in the form of ANNs to improve the accuracy of device identification (Aneja et al., 2018). Similarly, systems have been developed for automated classification of IoT devices based on their network traffic. Utilising different machine learning techniques, these systems achieve high accuracy in identifying device types from single network traffic packets (Aksoy & Gunes, 2019; Shahid et al., 2018). Moreover, solutions that automate the integration process of new IoT devices by classifying them based on their communication properties have been proposed (Pêgo & Nunes, 2017). Following the same objective of flexible device handling, a home automation system focused on the diversity of IoT devices and communication protocols was developed using a specific IoT platform, namely OpenHAB (Parocha & Macabebe, 2019). These methods and systems reduce the need for manual configuration and frequent application updates. While these works advance the automation of processes related to the identification, classification and integration of IoT devices, they do not address subsequent steps, such as holistic platform configuration.

Addressing this open issue, a dynamic configuration approach of smart environments has been presented in Mayer et al. (2016), leveraging a service composition system that combines semantic metadata and visual modelling tools. The resulting system can adapt to dynamic environments independent of a specific IoT platform. A similar approach towards platform-independent IoT deployments is realised by *X-IoT*, a model-driven solution for IoT application portability across platforms (Corradini et al., 2023). However, these solutions do not include device discovery or scenario-specific rule definition.

More holistically, the work by Javed et al. (2020) addresses the need for open and scalable IoT solutions. The authors propose a layered IoT platform designed to enhance interoperability, discovery and scalability within smart environments. By adopting edge computing principles, the platform aims to manage the vast amount of data generated by connected devices efficiently. However, this solution introduces its own platform, focusing primarily on scalability and heterogeneity, but does not fully integrate existing technologies in an open framework.

Considering the problem statement and related works, our approach requires a pipeline that addresses device discovery, scalability, platform configuration, platform independence, openness and model-based scenario specification. The corresponding requirements detailed in Table 1 ensure that our solution not only discovers and integrates IoT devices efficiently but also maintains flexibility and adaptability by supporting existing and open technologies. The requirement R6 specifically captures the need for defining scenario-specific rules in a user-friendly manner, which is addressed through the integration of the system-independent extension *IoT2Model* (cf. Section 4.3).

In summary, the goal of this contribution is captured through the formulation of the complete design problem by following the DSRM template (Wieringa, 2014):
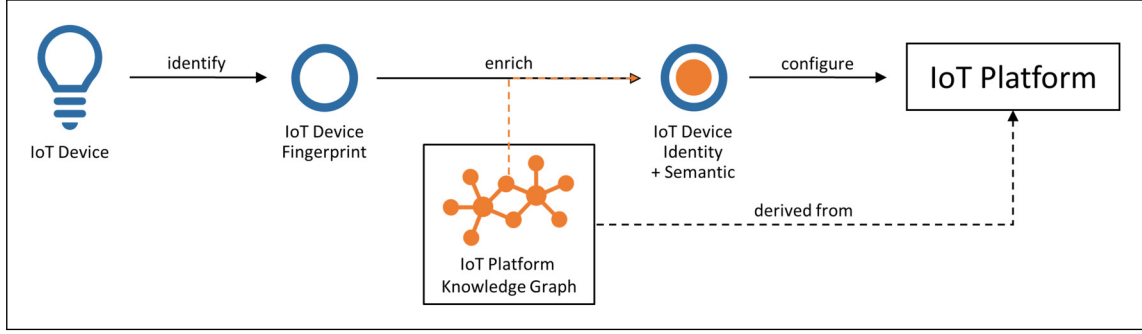
Figure 1. Conceptual Architecture of the IoT Device Discovery and IoT Platform Configuration Approach.

Improve the efficiency of IoT platform configuration (**problem context**)
. . . by developing a neuro-symbolic AI-based pipeline (**artifact**)
. . . that enables device discovery, platform configuration, platform independence,
scalability, openness and model-based scenario specification (**requirements**)
. . . to provide a holistic system for IoT environment configuration. (**goal**)

4 Neuro-Symbolic Architecture of the IoT Platform Configuration Approach

The goal of this contribution is the development of a design science artifact that enables the automated configuration of IoT environments. For this purpose, the above-described problem statement and derived requirements serve as foundational building blocks for the development. The conceptual architecture of the resulting artifact is displayed in Figure 1 and further detailed in the following.

The first step of our approach is the identification of IoT devices using fingerprinting techniques (cf. Section 3.2). This process is represented on the left side of Figure 1, in which the fingerprint of a physical IoT device is extracted to ensure accurate discovery and classification within the system. By implementing such identification mechanisms, we satisfy the *Device Discovery* requirement (R1), thus enabling seamless integration of diverse IoT devices.

Once a device is identified, it needs to be configured with the corresponding IoT platform. This configuration requires additional semantic information that is specific to each platform. As depicted on the right side of Figure 1, the IoT device identity is enriched with semantic data, guiding the configuration process. This satisfies the requirement for automated *Platform Configuration* (R2), streamlining the setup and reducing the need for manual intervention.

The platform-specific information necessary for the respective configuration process is contained within the IoT platform Knowledge Graph. This component is designed to be exchangeable, thus enabling a modular approach in which the Knowledge Graph can be replaced or updated as needed. This aspect of the architecture satisfies the requirement for *Platform Independence* (R3), ensuring flexibility and adaptability to evolving environments.

The two requirements of *Scalability* and *Openness* (R4, R5) apply to the complete system displayed in Figure 1. Each component of the architecture, including the IoT device fingerprinting, IoT platform Knowledge Graph and configuration processes, must adhere to these principles. By ensuring that every part of the system follows open standards and is scalable, the overall architecture can guarantee to meet these requirements.

These explanations align the conceptual architecture with the formulated requirements, demonstrating how each part of the design contributes to the overall goal of automated IoT platform configuration. In the upcoming subsections, we first present the specification of the design artifact, namely the *Instantiator Pipeline*, which operationalises the conceptual framework outlined above, before a concrete configuration case based on this implementation is showcased. Lastly, the implications of the *Model-based Scenario Specification* requirement (R6) are discussed.

4.1 Automated Device Discovery and Configuration: The Instantiator Pipeline

The *Instantiator Pipeline*[3] implements an instance of the presented conceptual architecture for automating IoT platform configuration (cf. Figure 1). To enable such an instance, the implementation contains three distinct modules, each dedicated to one of the three foundational requirements (R1-R3) identified previously (cf. Table 1). Figure 2 displays the complete overview of the *Instantiator Pipeline* implementation that contains these three modules:

– *Network Sniffer*: responsible for device discovery through gathering and processing of network traffic
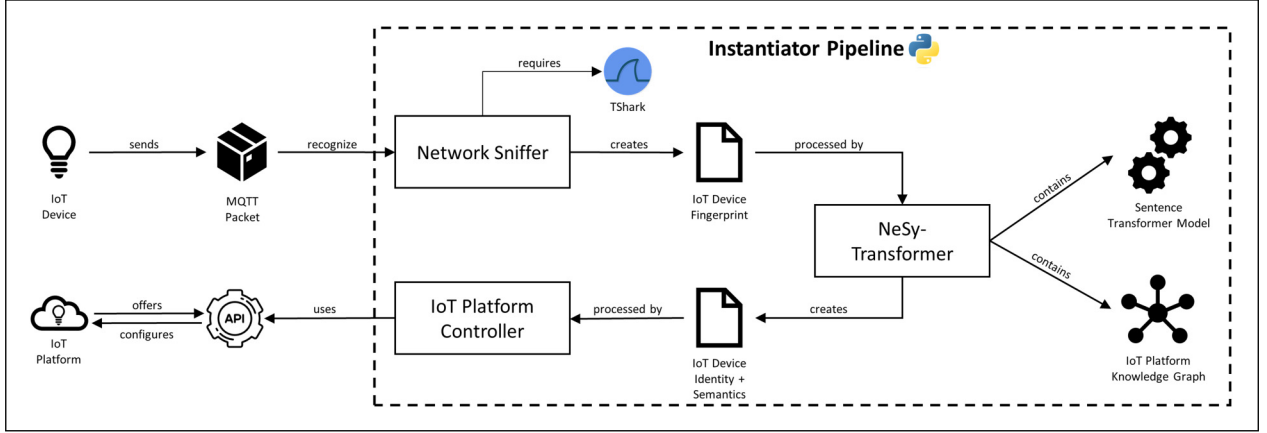
Figure 2. Neuro-Symbolic Implementation for IoT Device Discovery and IoT Platform Configuration: The Instantiator Pipeline.
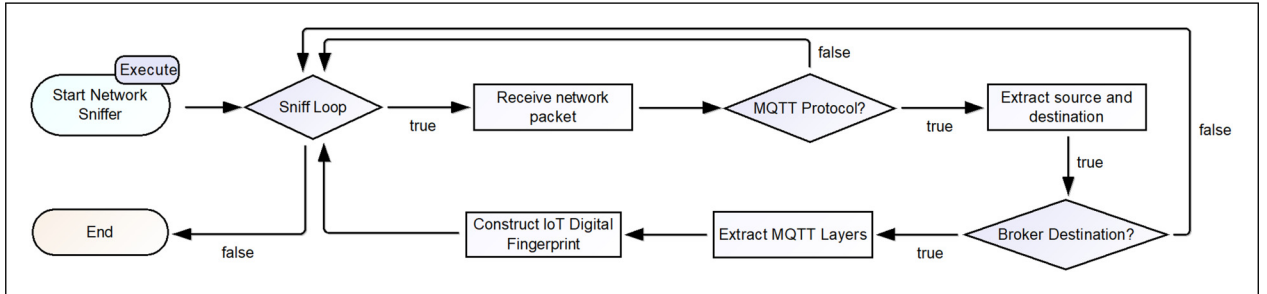

Figure 3. Process of the Network Sniffer loop Visualised as Flowchart Model.

– *NeSy-Transformer*: enhances gathered device information with platform-specific semantics
– *IoT Platform Controller*: provides the configuration interface to the respective IoT platform

The *Network Sniffer* captures and analyses network traffic within the given IoT environment. Using the *Pyshark* library[4], a Python wrapper for *TShark*, the sniffer intercepts network packets. *TShark* itself is a terminal-oriented version of *Wireshark*, a powerful network packet analyser tool (Sanders, 2011). The interception process of the *Network Sniffer* involves a continuous loop that filters network traffic to identify and log packets from IoT devices using the MQTT protocol. Such packets include the source, destination, topic and content of MQTT messages, which helps to identify MQTT brokers and individual IoT devices transmitting data. A flowchart model representing the process of the *Network Sniffer* loop is displayed in Figure. 3. The developed system maintains a map of devices and their recent activities, ensuring active monitoring and status updates of each device. This module provides the digital fingerprint of IoT devices (i.e. *Device Discovery* requirement) crucial for subsequent processing steps.

The *NeSy-Transformer* semantically enriches the digital fingerprints produced by the *Network Sniffer* in a platform-specific way using a neuro-symbolic approach. Based on recent advances regarding LLMs, a sentence transformer model is employed to align device fingerprints with concepts in a predefined Knowledge Graph. This alignment is achieved by matching the fingerprint information to corresponding nodes in the graph, which defines the concept of devices and various device types, along with their corresponding counterparts of the respective IoT platform. For the involved matching process, the *paraphrase-MiniLM-L6-v2* sentence transformer model[5] is employed. By aligning the digital fingerprint with a concept in the Knowledge Graph, the system applies reasoning techniques to derive actionable insights for device configuration and management. Semantic alignment techniques have been proven suitable for such problems, as shown in our previous work (Voelz, Amlashi, & Lee, 2023). The enriched data structure that results from the processing by the *NeSy-Transformer* is essential for accurately configuring and managing IoT devices within the environment. Moreover, the IoT platform Knowledge Graph that provides the basis for the semantic alignment process can be replaced or updated according to the respective platform of choice, thus satisfying the *Platform Independence* requirement. Having said that, a critical aspect of this approach is the strong dependency of the matching process on the content and structure of the Knowledge Graph, as the transformer always attempts to find the best matching concept. Consequently, if the Knowledge
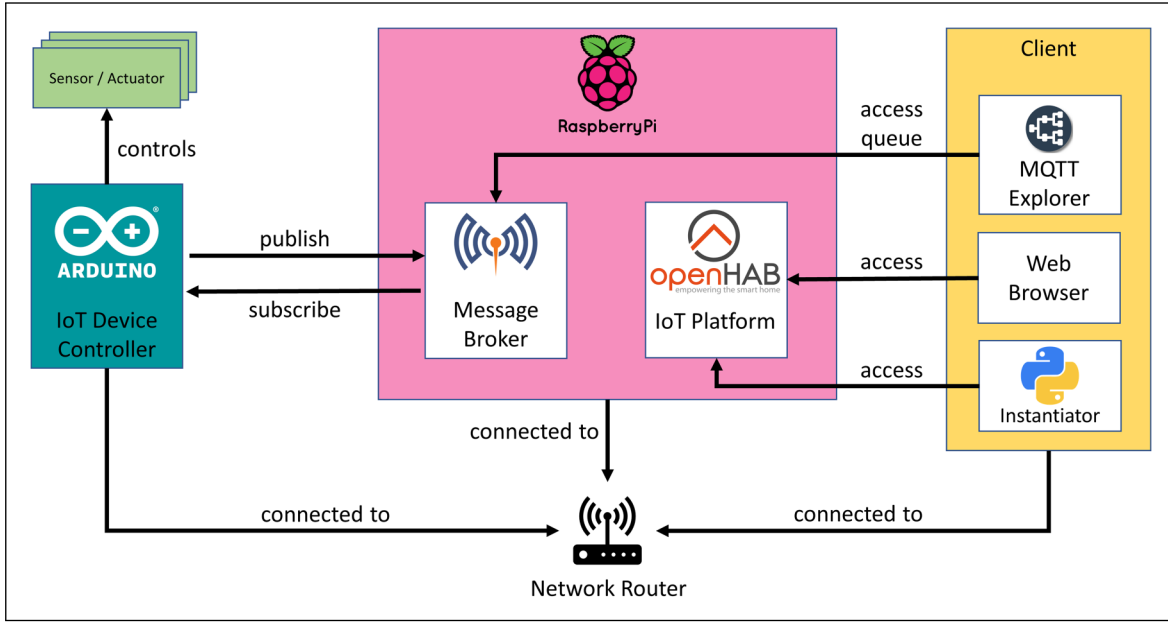
Figure 4. Overview of the Experiment Environment Architecture.

Graph does not contain the correct concept to align with, the resulting match will be inaccurate, potentially leading to inadequate device configurations.

The *IoT Platform Controller* serves as an interface between the *Instantiator Pipeline* and the IoT platform. It is responsible for creating and configuring IoT device representations through platform APIs, as well as removing inactive devices. The item creation process involves generating virtual counterparts of recognised devices by using the enriched fingerprints to establish necessary components such as data channels and control rules. This ensures that the IoT platform configuration automatically mimics the physical environment, maintaining a real-time representation of the IoT deployment. Such synchronisation is crucial for ensuring that the platform accurately reflects the current state of the physical environment, enabling reliable monitoring and control. The item deletion process is responsible for deregistering inactive or disconnected devices from the IoT platform, thus maintaining synchronisation with the physical environment. This is relevant as it prevents the platform from accumulating outdated device representations, which could lead to inefficiencies or errors in system operations. When a device is deleted, all associated data, including historical data, is removed from the platform. While the deletion of historical data may be a limitation in some cases, it is necessary to maintain synchronisation and avoid clutter in dynamic IoT environments.

4.2   A Smart House Configuration Case Based on the Instantiator Pipeline

The following configuration case that applies the *Instantiator Pipeline* is based on a simplified Smart House scenario that was described in Völz and Vaidian (2024) within an educational context. In short, the scenario revolving around pool safety required the Smart House to monitor and process various aspects of the environment to eventually trigger user-defined scenario rules (e.g., activating a servo motor based on RFID tag recognition). The setup of the corresponding experiment environment is listed below before the case-specific utilisation of the *Instantiator Pipeline* is presented.

4.2.1   Experiment Environment Setup. The physical experiment environment employed for the Smart House case incorporates an instance of the IoT platform OpenHAB, 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. This initial setup of the experiment environment is visualised in Figure 4.

OpenHAB. OpenHAB[6] stands for Open Home Automation Bus, an open-source home automation platform that 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 customisable framework. It was selected as an IoT platform because it adheres to the free and open-source principles (Babun et al., 2021; Javed et al., 2020) while
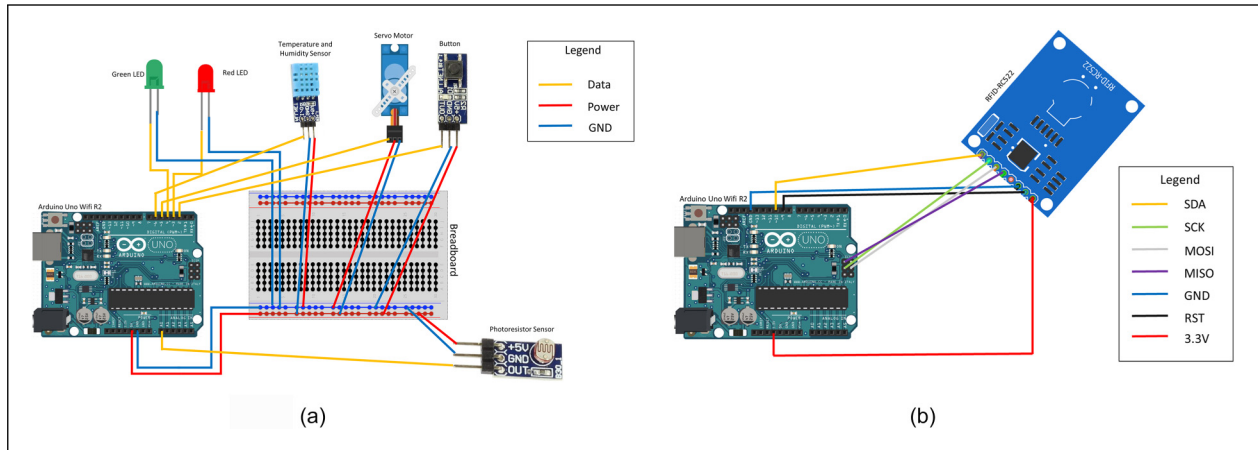
Figure 5. Physical Connections Between the Arduino UNO WIFI and the IoT Devices used in the Smart House Scenario. (a) IoT Device Pinout; (b) RFID Reader Pinout.

also supporting heterogeneous IoT devices (Parocha & Macabebe, 2019), thus satisfying the *Openness* and *Scalability* requirements.

Mosquitto MQTT Broker. The crucial intermediary of the experiment environment is the Mosquitto[7] MQTT message broker. Mosquitto utilises the MQTT protocol to ensure efficient and reliable communication between the various IoT devices and the OpenHAB IoT platform instance. Through the transmission of corresponding messages, it enables data exchange and control functionalities within the Smart House environment.

IoT Devices. Devices that are deployed within the Smart House environment include a variety of sensors and actuators controlled by an Arduino UNO WIFI microcontroller. These devices include a red and green LED light, a pressable button, a humidity and temperature sensor (one device for both), a photoresistor sensor, a servo motor and an RFID reader. The physical connections between the Arduino UNO WIFI and the IoT devices are displayed in Figure 5a, with the more complex pinout of the RFID reader being represented separately in Figure 5b. The proposed approach relies on the assumption that each sensor publishes data to a dedicated topic, allowing subscribers to receive real-time updates. This assumption aligns with common practices in IoT device communication, where publish-subscribe mechanisms are widely adopted for efficient data dissemination. Similarly, actuators, which typically do not generate data, are assumed to publish messages indicating their readiness and a list of available commands. Subscribers can send corresponding control commands to the actuators via the specified topic for remote operations. It is, therefore, important to emphasise that these assumptions are fundamental to the approach. Without dedicated topic-based communication for sensors and actuators, the entire system would fail to function as intended.

4.2.2 Case-Specific Utilisation of the Instantiator Pipeline.
In the following, the utilisation of the *Instantiator Pipeline* is showcased within the experiment environment setup of the Smart House scenario. The focus lies on highlighting the complete process of identifying an IoT device and enriching it with platform-specific semantics to automate the necessary device configuration steps.

Based on the experiment setup, it is assumed that the listed IoT devices send MQTT packets within an established network. As mentioned above, the *Network Sniffer* loop (cf. Figure 3) filters network traffic to discover the source, destination, topic and content of IoT device packets. An example of a gathered packet is displayed in Listing 1, which the system can process to identify the address 192.168.0.100:1883 as an MQTT broker that receives publish messages from two different IoT devices (the system ignores MQTT packets with a destination port commonly used as dynamic or ephemeral ports, which range from 49152 to 65535). The *Network Sniffer* recognises each device and stores them in a map object, with the topic of the data serving as the key (i.e. 'IoTDevice/Lamp/Green' and 'IoTDevice/Photoresistor', respectively). The resulting data structure, as displayed in Listing 2, captures all relevant information gathered by the system through packet sniffing and serves as IoT device fingerprints. The 'messages' field stores the last 10 messages sent by each device. The 'last_activity' field is utilised 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 labelled as inactive and the 'active' field is set to 'False'.

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

Listing 1: Example packets received by the Network Sniffer

```
{
    '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 2: Example output generated by the Network Sniffer based on two IoT devices

```
'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 3: Example output generated by the NeSy-Transformer for the sensor IoT device 'Photoresistor'

In the next step, the IoT device fingerprints are processed by the *NeSy-Transformer* that utilises a neuro-symbolic approach for the platform-specific semantic alignment procedure (cf. Figure 2). This procedure requires a sentence transformer model and an IoT platform Knowledge Graph. At the core of the Knowledge Graph lies the OpenHAB-specific ontology which defines the device concept belonging to the type openhab:Item. An *Item* represents a 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. Additionally, the Knowledge Graph encompasses instances of devices (sensors and actuators), each defined by a name, label, description, item type and, in the case of sensors, a channel type. By identifying the device instance most similar to the processed device fingerprint through the employed sentence transformer model, inference can be utilised to extract all device information that is required for processing by the API of the OpenHAB platform controller. Additionally, the datatype of the received messages is compared with the datatype defined for the counterpart in the Knowledge Graph to ensure accuracy. The enriched fingerprint output generated by the *NeSy-Transformer* for the example of the 'IoTDevice / Photoresistor' is displayed in Listing 3 (for actuators, the output would slightly differ since they do not require a channel but possible states and their changes). This output includes vital details such as device datatype, broker address, MQTT topic and channel type, all of which are fundamental for configuring an IoT device within OpenHAB.
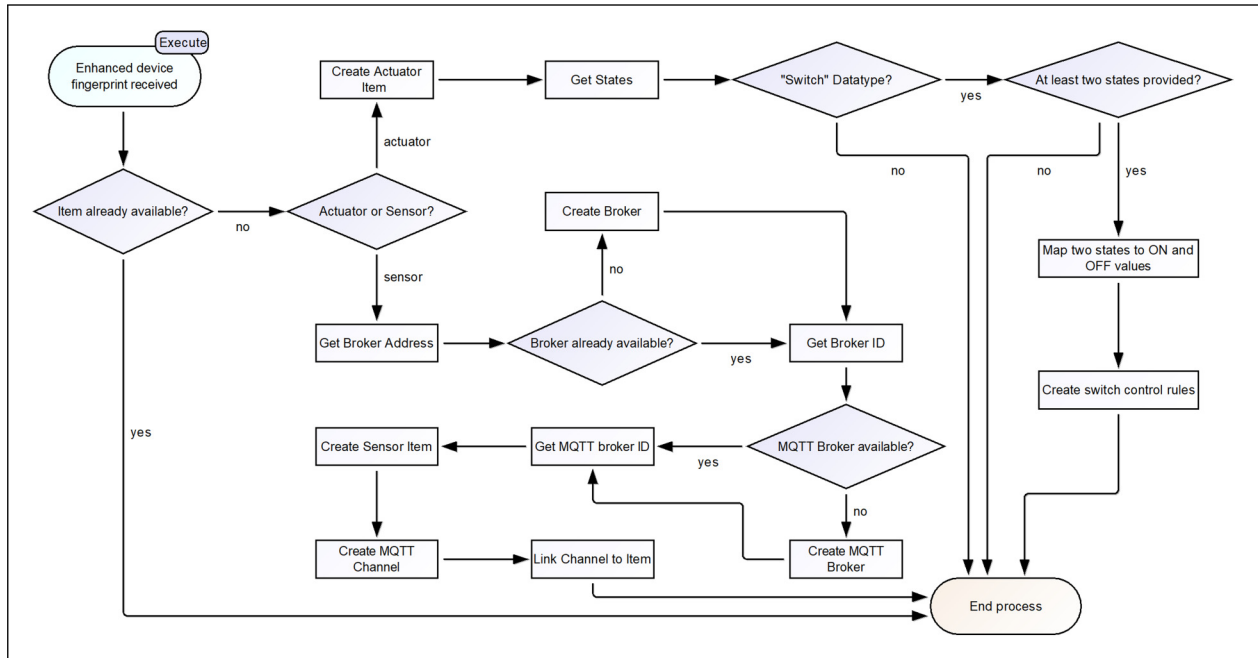
Figure 6. Item Creation Pipeline of the OpenHAB Controller.
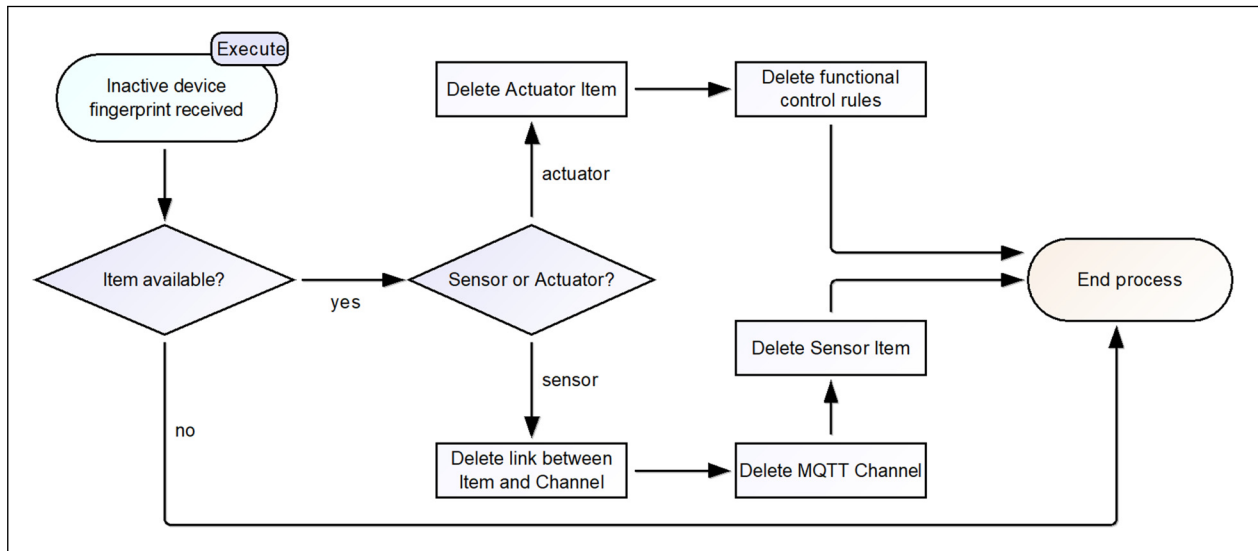
Figure 7. Item Deletion Pipeline of the OpenHAB Controller.

Finally, the API of the OpenHAB controller is utilised to process the semantically enriched fingerprints of IoT devices to either create new items or delete such that are inactive. Deletion of inactive items is important to ensure that the OpenHAB instance remains synchronised with the physical IoT devices. Each of the resulting processes that utilise the item-specific information of enriched device fingerprints is represented as flowcharts in Figures 6 and 7, respectively. In both processes, the sequence of events varies depending on whether it involves a sensor or actuator. This distinction ensures that the creation and deletion processes are tailored to the specific characteristics and requirements of each device type that are captured within the IoT platform Knowledge Graph. These steps conclude the automated IoT platform configuration process of the *Instantiator Pipeline* implementation.
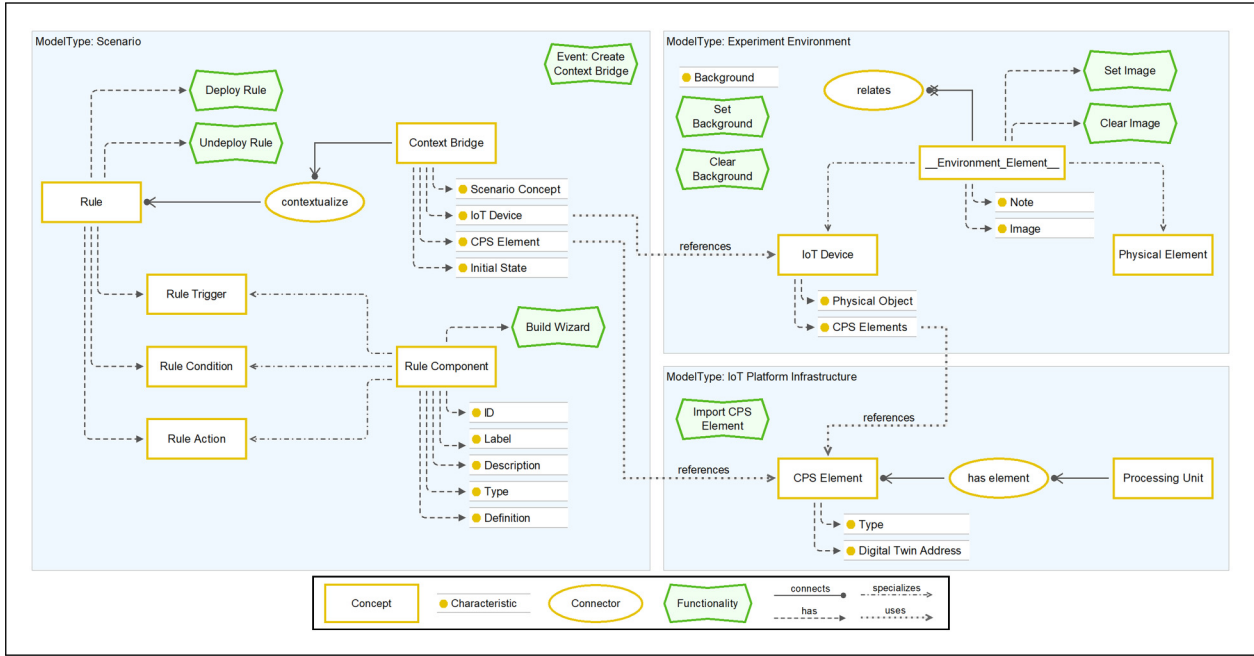
Figure 8. IoT2Model Metamodel (Created Using CoChaCo Tool).

4.3 Scenario-Based Configuration of IoT Platforms Through Conceptual Modelling: The IoT2Model Method

The automated IoT platform configuration process, enabled by the *Instantiator Pipeline*, satisfies five of the six identified requirements (cf. Table 1). The last requirement, R6 (Model-based Scenario Specification), is based on the aim of this research to support users in defining scenario-specific rules for a given IoT environment. This requirement is motivated by the notion that the primary advantage of IoT platforms demands scenario-based automation through the definition of specific rules (Domínguez-Bolaño et al., 2022; Fahmideh & Zowghi, 2020). To serve this purpose of configuring scenarios within IoT environments, the ADOxx-based, open-source modelling method *IoT2Model*[8] was created based on the GMMF and AMME (cf. Section 2.3), which allows users to design scenarios by modelling abstract rules. Moreover, it enables the bridging between the concepts required for a certain rule and the physical environment represented through the IoT platform of choice. Considering these capabilities, *IoT2Model* covers three different model types, each responsible for modelling IoT scenarios on a different abstraction level. The resulting metamodel, as implemented in ADOxx, is visualised in Figure 8 using the CoChaCo tool (Karagiannis et al., 2019). The following subsections detail the process of creating scenario-specific rules using the *IoT2Model* tool in the context of the Smat House case, as displayed in Figure 9.

4.3.1 IoT Infrastructure Model Type.
The *IoT Infrastructure* model allows users to design a model of the IoT devices integrated with a given IoT platform. This is achieved by linking the model with a corresponding platform, which is OpenHab in this case. To start the process of creating scenario-specific rules, all items created and configured by the *Instantiator Pipeline* are automatically imported through dedicated mechanisms and algorithms. These devices, represented as 'CPS Element' concept, can be semantically enriched with information such as corresponding computational units, which are relevant for creating accurate models of physical environments. Items represented in the *IoT Infrastructure* model are referenced in the *Environment* model, which is used to bridge the scenario rules and the physical environment.

4.3.2 Environment Model Type.
The *Environment* model has the purpose of capturing the environment by modelling the physical components and IoT devices available in it. Devices are represented as 'IoT Device' concept within this model type, which embodies physical objects with IoT capabilities from the environment. On the other hand, the 'Physical Element' concept represents non-IoT objects. Additionally, several of the modelled IoT devices can be combined into semantically rich modules, which is often the case in physical IoT environments (e.g., air conditioning units can contain temperature and humidity sensors, as well as cooling fans within the same device). This is achieved by creating references to the 'CPS Element' concepts available in corresponding *IoT Infrastructure* models.
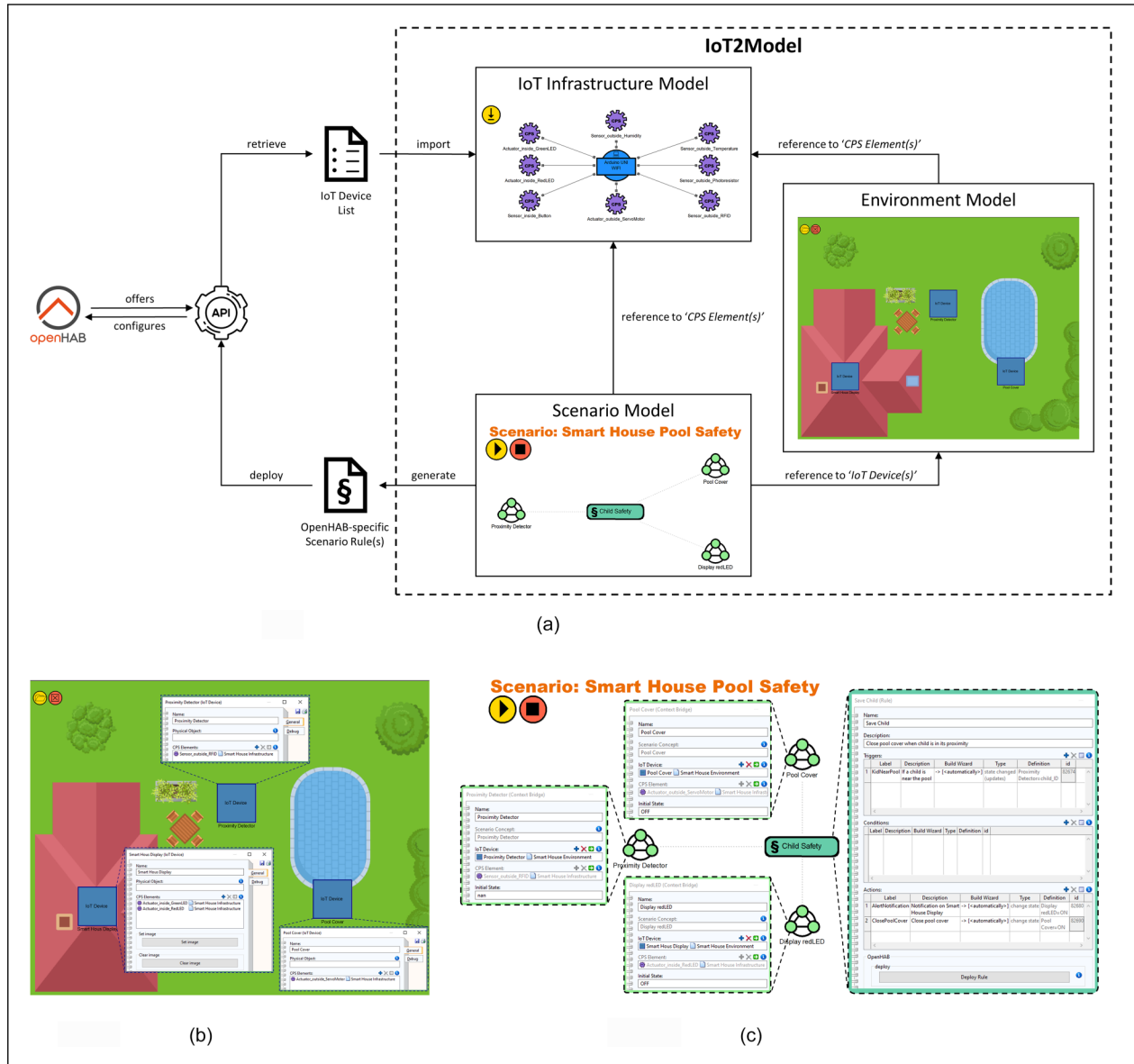
Figure 9. Process View and Detailed Model View of Utilising the IoT2Model Tool for the Pool Safety Scenario. (a) Process of Creating Scenario-Specific Rules Using the IoT2Model Tool; (b) Environment Model (detailed view); (c) Scenario Model (Detailed View).

4.3.3 Scenario Model Type. Lastly, the *Scenario* model type offers concepts for modelling a given scenario based on rules, usually consisting of triggers, conditions and actions. Triggers are states of the environment that specify the application scope of rules. If multiple triggers exist, each of the corresponding states that are met will start the rule execution process. The conditions of a rule comprise logical statements that have to be met before a rule can be executed. Actions correspond to operations that change the state of environment elements as a result of executing a defined rule. The resulting operations vary from simple state changes (turning an LED on or off) to more complex behaviours defined as executable scripts. Combining the different components accordingly, a rule like the following can be created:

> Every two seconds [Trigger], if no child is detected [Condition], turn (or keep) the green LED on [Action].

By using the specialisations of the 'Rule Component' concept from the *Scenario* model type, users can define their desired scenario rules through natural language. The specified rule components are then bridged to the fitting 'IoT Device' in the *Environment* model and the corresponding 'CPS Element' within the *IoT Infrastructure* model. For this bridging

process, the 'Context Bridge' concept is utilised, which is automatically added through an event-based functionality for each created 'Rule Component'. Afterward, the respective references to the related modelling concepts can be specified within the 'Context Bridge' concept. In addition, the *Scenario* model provides functionalities that enable users to directly deploy as well as undeploy defined rules, assuming a link to the API of the IoT platform of choice has been created beforehand.

4.3.4 Pool Safety Configuration Using IoT2Model. The process that is visualised in Figure 9a assumes that an instance of the OpenHAB IoT platform has already been configured using the previously presented environment setup (cf. Section 4.2.1) and the *Instantiator Pipeline*. Consequently, the process starts off by automatically importing the IoT devices that are active within the Smart House environment as 'CPS Element' objects into the *IoT Infrastructure* model. To accurately represent the environment setup, all devices are linked to the Arduino UNO WIFI as central 'Processing Unit'.

In the next step, the Smart House-based *Environment* model is created, in which the different 'IoT Device' objects are modelled and linked to the respective CPS elements available within the *IoT Infrastructure* model. In Figure 9b, a detailed view of the modelled IoT devices and their references is displayed. In this simplified environment, three IoT devices are present: a 'Proximity Detector' referencing the RFID sensor, a 'Pool Cover' referencing the servo motor and a 'Smart House display' referencing both the green and red LED lights.

After modelling the environment, scenario-specific rules are defined within the *Scenario* model. This step requires the specification of relevant triggers, conditions and actions, as displayed within Figure 9c. First, a rule named 'Child Safety' is modelled, which specifies the needed rule components in an OpenHAB-specific manner with the help of a setup assistant (i.e. functionality 'Build Wizard' in Figure 8). The trigger of the 'Child Safety' rule is defined by providing a label and description before utilising the setup assistant to select the respective type and definition. In this example, the rule execution process is started when the state of the concept 'Proximity Detector' is equal to 'child_ID'. Subsequently, an event-based functionality of *IoT2Model* automatically adds a corresponding 'Context Bridge' concept within the *Scenario* model, which is named based on the state-changing concept. Following the same approach, conditions and actions can be defined. Within Figure 9c, two actions are specified, which change the state of the concepts 'Display redLED' and 'Pool Cover' to 'ON' while creating the required 'Context Bridge' objects automatically. To complete the specification of scenario-specific rules, references to the associated *IoT Infrastructure* and *Environment* models have to be established. For example, the context bridge 'Proximity Detector' references the IoT device with the same name from the *Environment* model, which itself references the CPS element 'Sensor_outside_RFID' from the *IoT Infrastructure* model. If several CPS elements are available for a given IoT device, the user has to select the suitable one manually. Finally, the details of each rule are used for their platform-specific deployment. *IoT2Model* provides functionalities to deploy and delete either individual or all modelled rules.

To conclude, the *IoT2Model* tool provides the opportunity to model scenario-specific rules on different abstraction levels in a user-friendly manner by reducing the complexity of the required steps, therefore satisfying the *model-based scenario specification* requirement (R6). It has also been emphasised that this extension to the *Instantiator Pipeline* constitutes a significant part of holistic IoT platform configuration.

5 Discussion

The discussion highlights the contributions of our approach, particularly the integration of the *Instantiator Pipeline* with *IoT2Model*, and explores future directions for its application. *IoT2Model* was developed as an extension of the *Instantiator Pipeline* to address the need for the intuitive design of scenario-specific rules in IoT environments. While the *Instantiator Pipeline* effectively handles device discovery and basic configuration, it does not inherently support the creation of rules that govern how devices interact within a given environment. This limitation led to the introduction of the requirement R6 (model-based scenario specification), which emerged as a critical component for enabling user-defined, scenario-specific automation in IoT deployments.

The *IoT2Model* method bridges this gap by allowing users to create models that represent scenario-specific rules, thereby enabling more granular control and customisation of IoT deployments. This extension exemplifies the broader integration of neuro-symbolic AI with conceptual modelling techniques, showcasing the objective of semantic-driven systems engineering (cf. Buchmann et al., 2024). By combining the pattern recognition and learning strengths of neuro-symbolic AI with the simplification of complex configuration processes through conceptual modelling, our approach not only simplifies IoT platform configuration but also ensures that the resulting system effectively addresses the identified requirements (cf. Table 1). Future works will aim to test the interoperability of the *Instantiator Pipeline* and *IoT2Model* with other open-source IoT platforms like the one presented in Javed et al. (2020).

However, it is important to acknowledge certain limitations of the proposed approach. A key limitation is the dependency on pre-existing Knowledge Graphs for IoT platforms. The effectiveness of the NeSy-Transformer module depends on the availability and quality of these graphs, which are not always available or well-defined. In case no suitable concept is present in the respective Knowledge Graph, the transformer will still match the most similar one, which can cause incorrect device configurations. Additionally, creating and maintaining Knowledge Graphs requires domain expertise and continuous effort, which may be challenging for users with limited resources. The approach may also face scalability issues in highly heterogeneous IoT environments, as correctly aligning diverse device fingerprints with a unified Knowledge Graph could become complex and resource-intensive.

Moreover, the relevance of this work extends beyond the technical benefits, particularly in its application within educational settings. For example, the *IoT2Model* method was applied during this year's edition of the annual NEMO Summer School series (cf. Völz & Vaidian, 2024). The motivation for this application context is the importance of developing a comprehensive Digital Leader skill profile, which we have highlighted in previous works (Voelz et al., 2023). This skill profile encompasses a set of competencies required to operate effectively in the interdisciplinary and technologically advanced environments that characterise modern digital ecosystems (Karagiannis et al., 2022). However, it is evident that this profile still requires adoption within higher education institutions (HEIs) to meet the evolving demands of the digital era (Karagiannis et al., 2020), especially regarding the issues of conceptual modelling education (Buchmann et al., 2019). The *IoT2Model* method we have introduced plays a crucial role in this educational context by enabling the setup of experimental prototypes for hands-on evaluation and learning purposes. It not only reinforces the value of conceptual modelling but also facilitates the practical application of these models in IoT environments, thereby enhancing the learning experience. Consequently, the design artifacts presented in this contribution are intended for integration into HEI curricula through lecturers and presenters. By incorporating the *Instantiator Pipeline* and the *IoT2Model* method into such curricula, students can engage with a real-world example of how neuro-symbolic AI and conceptual modelling can be applied to enhance the configuration and customisation of IoT platforms. The ultimate goal of this contribution is that these tools will empower students and educators to develop the critical skills necessary for leadership in digital transformation initiatives, thereby ensuring that they are prepared to contribute to the ongoing evolution of digital ecosystems.

6 Conclusion

The rapid expansion and complexity of IoT environments present significant challenges in device discovery, platform configuration and management. As the number of interconnected devices grows, existing solutions often fall short of addressing the seamless integration and dynamic customisation required to realise the full potential of IoT ecosystems. To tackle these challenges, the core goal of this contribution was to develop a comprehensive design artifact that automates and enhances IoT platform configuration by integrating neuro-symbolic AI with conceptual modelling techniques. Our aim was to create a flexible, scalable and user-centric system capable of adapting to the evolving needs of IoT environments. For this purpose, we introduced the *Instantiator Pipeline*, an innovative solution that automates key processes such as IoT device discovery and corresponding platform configurations. The extension of the system with the *IoT2Model* method further enables the creation of scenario-specific rules, allowing for more granular control and customisation of IoT deployments. Beyond the technical improvements of this approach, we have discussed the educational opportunities it has to offer, particularly in fostering the development of critical Digital Leader skills within interdisciplinary and complex environments. The NEMO Summer School series was utilised as a testing ground for the application of the *IoT2Model* method in an educational setting, highlighting its potential for demonstrating the value of conceptual modelling in managing the complexity of intelligent information systems. Future work will focus on expanding the capabilities of the *IoT2Model* method, exploring new ways for enhancing the adaptability and intelligence of IoT systems and further integrating these advancements into educational contexts to foster the next generation of Digital Leaders.

Declaration of Conflicting Interests

The authors declared no potential conflicts of interest with respect to the research, authorship and/or publication of this article.

ORCID iDs

Danial M Amlashi [iD] https://orcid.org/0000-0003-3628-5725
Alexander Voelz [iD] https://orcid.org/0000-0003-4580-1445

Notes

1. The term *Digital Twin* is used in the referenced work. However, we chose not to engage in a detailed discussion of the terminology within the scope of this contribution, focusing instead on the core aspects of IoT platform configuration and scenario-based modelling.
2. https://adoxx.org
3. Full implementation can be found on the OMiLAB GitLab instance: https://code.omilab.org/danialm98/instantiator-pipeline
4. https://github.com/KimiNewt/pyshark
5. https://huggingface.co/sentence-transformers/paraphrase-MiniLM-L6-v2
6. https://openhab.org/
7. https://mosquitto.org/
8. Full implementation can be found on the OMiLAB Website: https://www.omilab.org/activities/projects/details/?id=373

References

Aksoy, A., & Gunes, M. H. (2019). Automated IoT device identification using network traffic. In *2019 IEEE International Conference on Communications (ICC)* (pp. 1–7). https://doi.org/10.1109/ICC.2019.8761559

Aneja, S., Aneja, N., & Islam, M. S. (2018). IoT Device Fingerprint using Deep Learning. In *2018 IEEE International Conference on Internet of Things and Intelligence System (IOTAIS)* (pp. 174–179). IEEE. https://doi.org/10.1109/IOTAIS.2018.8600824

Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, *34*(6), 26–38. https://doi.org/10.1109/MSP.2017.2743240

Ashton, K. (2009). That 'internet of things' thing. *RFID Journal*, *22*(7), 97–114.

Atzori, L., Iera, A., & Morabito, G. (2010). The internet of things: A survey. *Computer Networks*, *54*(15), 2787–2805. https://doi.org/10.1016/j.comnet.2010.05.010

Babun, L., Denney, K., Celik, Z. B., McDaniel, P., & Uluagac, A. S. (2021). A survey on IoT platforms: Communication, security, and privacy perspectives. *Computer Networks*, *192*, 108040. https://doi.org/10.1016/j.comnet.2021.108040

Buchmann, R. A., Ghiran, A.-M., Döller, V., & Karagiannis, D. (2019). Conceptual modeling education as a "design problem". *Complex Systems Informatics and Modeling Quarterly*, 21–33. https://doi.org/10.7250/csimq.2019-21.02

Buchmann, R., Eder, J., Fill, H.-G., Frank, U., Karagiannis, D., Laurenzi, E., Mylopoulos, J., Plexousakis, D., & Santos, M. Y. (2024). Large language models: Expectations for semantics-driven systems engineering. *Data & Knowledge Engineering*, *152*, 102324. https://doi.org/10.1016/j.datak.2024.102324

Corradini, F., Fedeli, A., Fornari, F., Polini, A., Re, B., & Ruschioni, L. (2023). X-IoT: A model-driven approach to support IoT application portability across IoT platforms. *Computing*, *105*(9), 1981–2005. https://doi.org/10.1007/s00607-023-01155-z

Domínguez-Bolaño, T., Campos, O., Barral, V., Escudero, C. J., & García-Naya, J. A. (2022). An overview of IoT architectures, technologies, and existing open-source projects. *Internet of Things*, *20*, 100626. https://doi.org/10.1016/j.iot.2022.100626

Ehrlinger, L., & Wöß, W. (2016). Towards a Definition of Knowledge Graphs. In M. Martin, M. Cuquet, & E. Folmer (Eds.). *Joint Proceedings of the Posters and Demos Track of the 12th International Conference on Semantic Systems - SEMANTiCS2016 and the 1st International Workshop on Semantic Change & Evolving Semantics (SuCCESS'16) co-located with the 12th International Conference on Semantic Systems (SEMANTiCS 2016)* (Vol. 1695). CEUR-WS.org. https://ceur-ws.org/Vol-1695/paper4.pdf

Fahmideh, M., & Zowghi, D. (2020). An exploration of IoT platform development. *Information Systems*, *87*, 101409. https://doi.org/10.1016/j.is.2019.06.005

Feigenbaum, E. A. (1984). Knowledge engineering: The applied side of artificial intelligence. *Annals of the New York Academy of Sciences*, *426*(1), 91–107. https://doi.org/10.1111/j.1749-6632.1984.tb16513.x

Flasiński, M. (2016). Symbolic Artificial Intelligence. In *Introduction to Artificial Intelligence* (pp. 15–22). Springer International Publishing, Cham. https://doi.org/10.1007/978-3-319-40022-8_2

González García, C., Meana Llorián, D., Pelayo G-Bustelo, C., & Cueva-Lovelle, J. M. (2017). A review about smart objects, sensors, and actuators. *International Journal of Interactive Multimedia and Artificial Intelligence*, *4*(3), 7. https://doi.org/10.9781/ijimai.2017.431

Hilario, M. (1997). An Overview of Strategies for Neurosymbolic Integration. In *Connectionist-Symbolic Integration: From Unified to Hybrid Approaches* (pp. 13–35). Psychology Press. https://doi.org/10.4324/9780203763667

Hunkeler, U., Truong, H. L., & Stanford-Clark, A. (2008). 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)* (pp. 791–798). IEEE, Bangalore, India. https://doi.org/10.1109/COMSWA.2008.4554519

Javed, A., Malhi, A., Kinnunen, T., & Främling, K. (2020). Scalable IoT platform for heterogeneous devices in smart environments. *IEEE access*, *8*, 211973. https://doi.org/10.1109/ACCESS.2020.3039368

Jordan, M. I., & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. *Science (New York, N.Y.)*, *349*(6245), 255–260. https://doi.org/10.1126/science.aaa8415

Karagiannis, D. (2015). Agile Modeling Method Engineering. In N. Karanikolas, D. Akoumianakis, M. Nikolaidou, D. Vergados, & M. Xeno, (Eds.). *Proceedings of the 19th Panhellenic Conference on Informatics*(pp. 5–10). Association for Computing Machinery. https://doi.org/10.1145/2801948.2802040

Karagiannis, D. (2024). AdoScript: A DSL for developing conceptual modeling methods. In S. Strecker, & J. Jung, (Eds.). *Informing Possible Future Worlds. Essays in Honour of Ulrich Frank* (pp. 287–308). Logos, Berlin.

Karagiannis, D., Buchmann, R. A., Boucher, X., Cavalieri, S., Florea, A., Kiritsis, D., & Lee, M. (2020). OMiLAB: A Smart Innovation Environment for Digital Engineers. In, L.M. Camarinha-Matos, H. Afsarmanesh, & A. Ortiz (Eds.). *Boosting Collaborative Networks 4.0* (pp. 273–282). Springer International Publishing, Cham. https://doi.org/10.1016/10.1007/978-3-030-62412-5_23

Karagiannis, D., Buchmann, R. A., Burzynski, P., Reimer, U., & Walch, M. (2016). Fundamental Conceptual Modeling Languages in OMiLAB. In D. Karagiannis, H.C. Mayr, & J. Mylopoulos (Eds.). *Domain-Specific Conceptual Modeling: Concepts, Methods and Tools* (pp. 3–30). Springer International Publishing, Cham. https://doi.org/10.1007/978-3-319-39417-6_1

Karagiannis, D., Buchmann, R. A., & Utz, W. (2022). The OMiLAB digital innovation environment: Agile conceptual models to bridge business value with digital and physical twins for product-service systems development. *Computers in Industry*, *138*, 103631. https://doi.org/10.1016/j.compind.2022.103631

Karagiannis, D., Burzynski, P., Utz, W., & Buchmann, R. A. (2019). A Metamodeling Approach to Support the Engineering of Modeling Method Requirements. In *2019 IEEE 27th International Requirements Engineering Conference (RE)* (pp. 199–210). https://doi.org/10.1109/RE.2019.00030

Karagiannis, D., & Kühn, H. (2002). Metamodelling Platforms. In K. Bauknecht, A. M. Tjoa & G. Quirchmayr, (Eds.). *E-Commerce and Web Technologies* (p. 182). Springer, Berlin Heidelberg. https://doi.org/10.1007/3-540-45705-4_19

Karagiannis, D., Lee, M., Hinkelmann, K., & Utz, W. (Eds.). (2022). *Domain-specific conceptual modeling: Concepts, methods and ADOxx tools* (1st ed.). Springer International Publishing. https://doi.org/10.1007/978-3-030-93547-4

Karagiannis, D., Mayr, H. C., & Mylopoulos, J. (Eds.). (2016). *Domain-specific conceptual modeling: Concepts, methods and tools* (1st edn.). Springer International Publishing. https://doi.org/10.1007/978-3-319-39417-6

Kiritsis, D. (2011). Closed-loop PLM for intelligent products in the era of the internet of things. *Computer-Aided Design*, *43*(5), 479–501. https://doi.org/10.1016/j.cad.2010.03.002

Li, S., Xu, L. D., & Zhao, S. (2015). The internet of things: A survey. *Information Systems Frontiers*, *17*(2), 243–259. https://doi.org/10.1007/s10796-014-9492-7

Mayer, S., Verborgh, R., Kovatsch, M., & Mattern, F. (2016). smart configuration of Smart environments. *IEEE Transactions on Automation Science and Engineering*, *13*(3), 1247–1255. https://doi.org/10.1109/TASE.2016.2533321

Mayr, H. C., & Thalheim, B. (2021). The triptych of conceptual modeling. *Software and Systems Modeling*, *20*, 7–24. https://doi.org/10.1007/s10270-020-00836-z

Newell, A., & Simon, H. A. (2007). Computer Science as Empirical Inquiry: Symbols and Search. In *ACM Turing Award Lectures* (pp. 113–126). Association for Computing Machinery. https://doi.org/10.1145/1283920.1283930

Parocha, R. C., & Macabebe, E. Q. B. (2019). 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)* (pp. 67–73). https://doi.org/10.1109/IoTaIS47347.2019.8980370

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems*, *24*(3), 45–77. https://doi.org/10.2753/MIS0742-1222240302

Pêgo, P. R. J., & Nunes, L. (2017). Automatic discovery and classifications of IoT devices. In *2017 12th Iberian Conference on Information Systems and Technologies (CISTI)* (pp. 1–10). https://doi.org/10.23919/CISTI.2017.7975691

Ray, S. (2019). A Quick Review of Machine Learning Algorithms. In *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)* (pp. 35–39). IEEE. https://doi.org/10.1109/COMITCon.2019.8862451

Razzaque, M. A., Milojevic-Jevric, M., Palade, A., & Clarke, S. (2016). Middleware for internet of things: A survey. *IEEE Internet of Things Journal*, *3*(1), 70–95. https://doi.org/10.1109/JIOT.2015.2498900

Sanders, C. (2011). *Practical packet analysis: Using wireshark to solve real-world proble* (2nd edn). No Starch Press.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, *61*, 85–117. https://doi.org/10.1016/j.neunet.2014.09.003

Shahid, M. R., Blanc, G., Zhang, Z., & Debar, H. (2018). IoT Devices Recognition Through Network Traffic Analysis. In *2018 IEEE International Conference on Big Data (Big Data)* (pp. 5187–5192). https://doi.org/10.1109/BigData.2018.8622243

Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G. (2009). Actuators and Sensors. In *Robotics: Modelling, Planning and Control*(pp. 191–231). Springer. https://doi.org/10.1007/978-1-84628-642-1_5

Singh, M., Fuenmayor, E., Hinchy, E., Qiao, Y., Murray, N., & Devine, D. (2021). Digital twin: Origin to future. *Applied System Innovation*, *4*(2), 36. https://doi.org/10.3390/asi4020036

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

Tang, S., Shelden, D. R., Eastman, C. M., Pishdad-Bozorgi, P., & Gao, X. (2019). 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. https://doi.org/10.1016/j.autcon.2019.01.020

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L.u., & Polosukhin, I. (2017). Attention is All you Need. In I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.). *Advances in Neural Information Processing Systems* (Vol. 30, pp. 5998–6008) Curran Associates, Inc.

Voelz, A., Amlashi, D. M., & Lee, M. (2023). Semantic Matching Through Knowledge Graphs: A Smart City Case. In M. Ruiz & P. Soffer, (Eds.). *Advanced Information Systems Engineering Workshops* (pp. 92–104). Springer International Publishing. https://doi.org/10.1007/978-3-031-34985-0_10

Voelz, A., Muck, C., Amlashi, D. M., & Karagiannis, D. (2023). Bridging haptic design thinking and cyber-physical environments through Digital Twins using conceptual modeling. In *Joint Proceedings of the BIR 2023 Workshops and Doctoral Consortium co-located with 22nd International Conference on Perspectives in Business Informatics Research* (pp. 195–208). https://ceur-ws.org/Vol-3514/paper93.pdf

Völz, A., Amlashi, D. M., Burzynski, P., & Utz, W. (2024). ADOxx: Eine Low-Code-Plattform für die Entwicklung von Modellierungswerkzeugen. *HMD Praxis der Wirtschaftsinformatik*, *61*(5), 1295–1316. 10.1365/s40702-024-01096-x

Völz, A., & Vaidian, I. (2024). Digital Transformation through Conceptual Modeling: The NEMO Summer School Use Case. In *Modellierung 2024*, Gesellschaft für Informatik e.V., Bonn, (pp. 139–156). https://doi.org/10.18420/modellierung2024_014

Wieringa, R. J. (2014). *Design Science Methodology for Information Systems and Software Engineering*. Springer.

Zikria, Y. B., Ali, R., Afzal, M. K., & Kim, S. W. (2021). Next-generation internet of things (ioT): Opportunities, challenges, and solutions. *Sensors*, *21*(4). https://doi.org/10.3390/s21041174