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

Danial M. Amlashi [a], Alexander Voelz [b,*] and Dimtris Karagiannis [a]
[a] *Department of Knowledge Engineering, Faculty of Computer Science University of Vienna, Austria*
E-mails: danial.mohammadi.amlashi@univie.ac.at, dk@univie.ac.at
[b] *Doctoral School Computer Science, University of Vienna, Austria*
E-mail: alexander.voelz@univie.ac.at

Abstract.
Complex 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 AI within a neuro-symbolic system. We propose a comprehensive approach to IoT platform configuration that integrates neuro-symbolic reasoning and conceptual modeling 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, minimizing manual intervention and ensuring seamless interoperability. During the initial iteration of the applied research methodology, it was recognized 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 behavior of their IoT deployments to meet specific requirements. By utilizing 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.

Keywords: Neuro-symbolic AI, IoT platform configuration, conceptual modeling, domain-specific modeling methods

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 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

*Corresponding author. E-mail: alexander.voelz@univie.ac.at.

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 modeling 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 modeling 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 summarizing key findings and insights.

2. Theoretical Foundations

2.1. Internet of Things and Internet of Things Platforms

First proposed by Kevin Ashton in 1999 [4], the Internet of Things (IoT) is a rapidly evolving paradigm in smart environments. IoT involves various objects such as sensors and actuators, which are uniquely identifiable and capable of interacting with each other [5]. Such interactions involve the sending and receiving of messages of both physical and virtual objects with uniques IDs, thus forming a network environment of integrated things [27]. Essentially, IoT combines network communication and computation, allowing interconnected devices to gather, exchange, process, and consume data in a given environment [28].

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 categorized 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 [14, 41]. Additionally, complex devices like single-board computers (e.g., Raspberry Pi, Arduino) can also be considered IoT devices in a more holistic view [44]. 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 [16, 43].

In the complex IoT landscape, managing diverse devices and data streams requires a centralized infrastructure often referred to as IoT Platform. An IoT Platform, or IoT middleware, acts as an integration layer [36], creating a Digital Twin of physical devices to mimic and control them in real-time [42]. This central management enables efficient device handling, monitoring, analysis, and optimization of IoT deployments through HTTP-based REST APIs [11]. Integrating a device with an IoT platform involves establishing its representation within the platform, facilitating 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 [9]. The primary advantage of IoT platforms lies in automating the respective IoT environment through self-defined rules that trigger specific actions autonomously, thereby enabling users to tailor the behavior of their IoT deployments to suit specific requirements [9, 11].

2.2. Symbolic, Connectionist, and Neuro-symbolic Artificial Intelligence

Artificial Intelligence (AI) can be categorized 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 [31], emphasizes the role of symbolic representation in achieving intelligent behavior. 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 [13]. 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 [10]. 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, such as expert systems [12].

Connectionist AI, also known as sub-symbolic AI, has the goal of establishing correlations between input data and output variables by optimizing parameters of a transformation function. This optimization process is labeled 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 [18]. The common learning approaches are supervised, unsupervised, and semi-supervised, while new developments often rely on reinforcement learning [3]. Among many established learning algorithms, [35], deep learning algorithms have become increasingly popular due to their application for Artificial Neural Networks (ANNs) [38] and, most recently, for the transformer model architecture that provides the basis of Large Language Models (LLMs) [45]. 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. 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 [15]. 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 Modeling and Domain-Specific Modeling Methods

In the context of this research, conceptual modeling, and specifically the development of Domain-Specific Modeling Methods (DSMMs), becomes increasingly relevant as we address the ex-post requirement $R6^{+}$ later in this contribution (cf. Section 3.2). Conceptual modeling offers powerful means of simplifying complex systems by abstracting them into understandable representations [30]. To achieve such simplifications through the development of a DSMM, two foundational frameworks from the conceptual modeling domain are considered: the Generic Modeling Method Framework (GMMF) and Agile Modeling Method Engineering (AMME).

DSMMs are tailored to capture the unique characteristics of specific application domains [22, 26]. The GMMF provides a structured approach for designing modeling methods, according to which each method consists of a modeling language, a modeling procedure, and mechanisms and algorithms [20]. The modeling language covers the syntax, notation, and semantics of the respective modeling method, which are specified as a machine-processable metamodel. The modeling procedure contains the intended process of applying the method. Finally, mechanisms and algorithms can be implemented to provide modeling functionalities that go beyond mere representation purposes, such as model transformation or code generation capabilities [23]. The development process of a DSMM requires

a metamodeling platform like ADOxx[1], which supports the GMMF's components. Furthermore, the five phases of the AMME lifecycle offer a systematic process model for the development and iterative refinement of DSMMs [19].

To conclude, the GMMF provides a structured approach for designing modeling methods, while AMME provides a comprehensive lifecycle 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. [33]. 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 [49]. 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 [33]. 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:

- *Lack of Support for Device Discovery*: Automated IoT device identification comprises an important issue for monitoring the devices connected to a certain network [1]. A variety of approaches to device discovery exist for both established as well as newly developed IoT platforms (cf. [17]). Nevertheless, identifying and also integrating devices into the ecosystem of an IoT platform is a complex task due to the diverse nature of IoT devices, each with its unique communication protocols and connectivity methods.
- *Complex Platform Configuration and Deployment*: Building upon the previous challenge, the integration of identified IoT devices is a labor-intensive process if no platform-specific solution is provided. Consequently, the manual configuration and deployment of IoT platforms to ensure interoperability between devices can become a complex task, which is why platform-independent solutions for dynamic configurations of smart environments have already been proposed [29].
- *Open-Source Principle*: It has been noted that differences exist between the pricing of consumer-oriented and enterprise IoT platform solutions [6]. In fact, the majority of available options have a specific pricing model and do not follow the open-source principle [6, 17]. Adherence to the free and open-source principles can thus pose considerable challenges for systems promoting adaptability, interoperability, and widespread adoption.
- *Expected Growth in the Number of Devices*: The number of IoT devices is projected to increase significantly in the near future [50]. This anticipated growth poses a scalability challenge for current IoT platforms and established methods for device discovery [17, 39]. Ensuring that the system can handle a larger number of devices without compromising performance or reliability is crucial for future-oriented IoT deployments.

By addressing the highlighted challenges, our research aims to develop a comprehensive approach for automated IoT platform configuration, enhancing the discovery, integration, and management of IoT devices in a scalable and efficient manner. Under these considerations regarding IoT platform configuration and deployment, we formulate a preliminary design problem under the DSRM template [49], based on which specific requirements for the respective design science artifact to be created are derived subsequently:

> Improve the efficiency of IoT platform configuration (**problem context**)
> ... by developing a neuro-symbolic AI-based pipeline (**artifact**)
> ... that addresses existing challenges from the field (**requirements**)
> ... to provide a holistic system for IoT environment configuration. (**goal**)

[1] https://adoxx.org

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 recognize a wide range of device types automatically, reducing the need for manual configuration.
R2	Platform Configuration	The system must automate the cumbersome platform configuration and deployment process, which includes the set up of communication channels, the integration of discovered devices, and ensuring interoperability between them. The goal is to streamline the deployment process and reduce setup time.
R3	Platform Independence	The system should support platform independence, allowing for IoT platforms to be changed without significant disruption. This includes having adaptable components that enable switching between different platforms with minimal effort to adapt to evolving needs and technologies.
R4	Openness	The system must follow the open-source principle to be freely available and extensible. This includes supporting open standards and utilizing free APIs for developers to extend the system functionality.
R5	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.
$R6^+$	Model-based Scenario Specification	The system should support extensions based on the ongoing development of the IoT2Model method, which enables users to model scenario-specific rules for a given IoT environment. This will allow user-friendly adaptions of IoT deployments through self-defined rules triggering self-defined actions.

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 [33], related works addressing similar issues are contrasted for further considerations.

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 integrates device discovery, platform configuration, and scalability has not been proposed yet. Subsequently, we summarize 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 [2]. Similarly, systems for automated classification of IoT devices based on their network traffic have been developed. Utilizing different machine learning techniques, these systems achieve high accuracy in identifying device types from single network traffic packets [1, 39]. Moreover, solutions that automate the integration process of new IoT devices by classifying them based on their communication properties have been proposed [34]. This method reduces 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 [29], leveraging a service composition system that combines semantic metadata and visual modeling tools. The resulting system offers flexibility by adapting to dynamic environments and user goals, independent of a specific IoT platform. Following the same requirement of flexibility, a home automation system focused on the diversity of IoT devices and communication protocols was developed using a specific IoT platform, namely OpenHAB [32].

More holistically, the work by Javed et al. [17] 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 challenges outlined in the problem statement and the shortcomings identified in the related works, our approach requires a robust pipeline that addresses seamless device discovery, automated platform configuration, openness, scalability, and platform interchangeability. The requirements R1-5 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 $R6^+$ was only added after the first process iteration of the DSRM as a system-independent extension and is thus detailed separately within the next section (cf. Section 4.3).

In summary, the ultimate goal of this contribution is captured through the formulation of the complete design problem by following the DSRM template [49]:

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

4. Automating the Configuration of IoT Platforms

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 Fig. 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 Fig. 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 Fig. 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 remaining two requirements of *Openness* and *Scalability* (R4, R5) apply to the complete system displayed in Fig. 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 operationalizes 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.

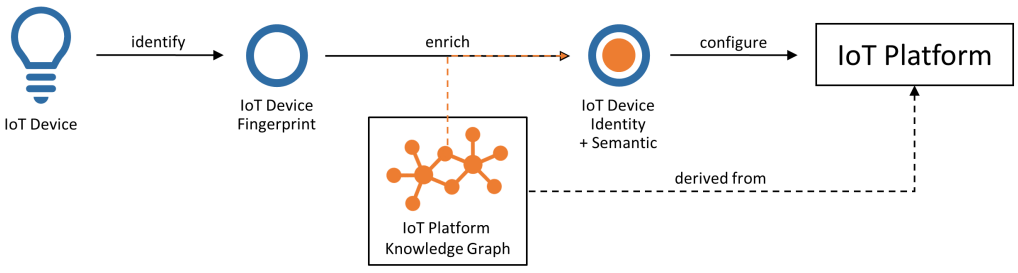


Fig. 1. Conceptual architecture of the IoT device discovery and IoT platform configuration approach.

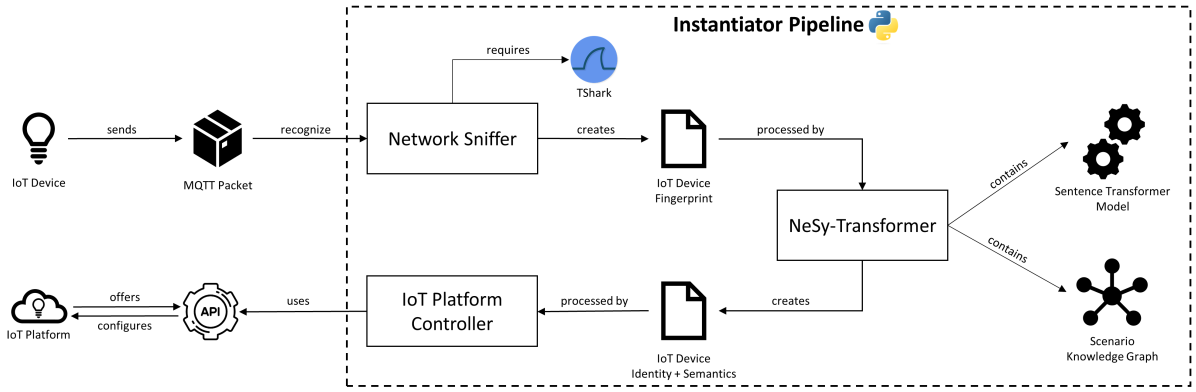


Fig. 2. Neuro-symbolic implementation for IoT device discovery and IoT platform configuration: The Instantiator Pipeline.

4.1. *Configuration of IoT Platforms through a Neuro-symbolic Approach: The Instantiator Pipeline*

The *Instantiator Pipeline*[2] implements an instance of the presented conceptual architecture for automating IoT platform configuration (cf. Fig. 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). Fig. 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
- *NeSy-Transformer*: enhances gathered information with platform-specific semantics using a neuro-symbolic algorithm
- *IoT Platform Controller*: provides the configuration interface to the respective IoT platform

The *Network Sniffer* captures and analyzes network traffic within the given IoT environment. Using the Pyshark library[3], a Python wrapper for TShark, the sniffer intercepts network packets. TShark itself is a terminal-oriented version of Wireshark, a powerful network packet analyzer tool [37]. 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 Fig. 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* has the purpose of semantically enriching the digital fingerprints produced by the *Network Sniffer* in a platform-specific way using a neuro-symbolic approach. It employs a sentence transformer model that is based on recent advances regarding LLMs to align device fingerprints with concepts in a predefined Knowledge Graph. This graph defines the concept of devices and various device types, along with their corresponding counterparts within the respective IoT platform. For the involved matching process, the paraphrase-MiniLM-L6-v2 sentence transformer model[4] is employed. By aligning the digital fingerprint with a concept in the Knowledge Graph, the system applies reasoning techniques to extract the necessary information for creating Digital Twins of the devices. Semantic alignment techniques have been proven suitable for such problems, as has been shown in our previous work [46]. 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.

[2]Full implementation can be found on the OMiLAB Gitlab instance: https://code.omilab.org/danialm98/instantiator-pipeline
[3]https://github.com/KimiNewt/pyshark
[4]https://huggingface.co/sentence-transformers/paraphrase-MiniLM-L6-v2

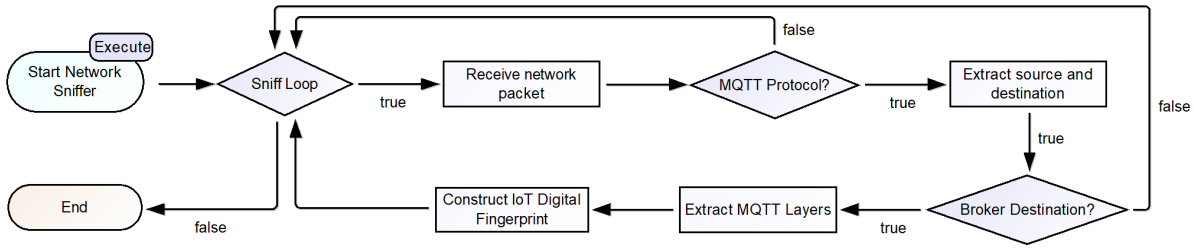


Fig. 3. Process of the Network Sniffer loop visualized as flowchart model

The *IoT Platform Controller* serves as an interface between the *Instantiator Pipeline* and the IoT platform. It is responsible for creating and configuring Digital Twins of IoT devices through the API offered by IoT platforms, as well as removing inactive devices. The item creation process involves generating virtual counterparts of recognized devices by using the enriched fingerprints to establish necessary components such as data channels and control rules. The item deletion process ensures that inactive or removed devices are properly deregistered from the IoT platform, maintaining synchronization with the physical environment.

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 Home scenario that was described in [48] within an educational context. In short, the scenario revolving around pool safety required the Smart Home 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 utilization 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 OpenHAB IoT platform, 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 visualized in Fig. 4.

OpenHAB OpenHAB[5] 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 customizable framework. It was selected as an IoT platform because it adheres to the free and open-source principles [6, 17] while also supporting heterogeneous IoT devices [32], thus satisfying the *Openness* and *Scalability* requirements.

Mosquitto MQTT Broker The crucial intermediary of the experiment environment is the Mosquitto[6] MQTT message broker. Mosquitto utilizes 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 Fig. 5a, with the more complex pinout of the RFID reader being represented separately in Fig. 5b. Each of the sensors publishes data to a dedicated topic, allowing subscribers to receive real-time updates. Since actuators typically do not 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 for remote operations.

[5] https://openhab.org/
[6] https://mosquitto.org/

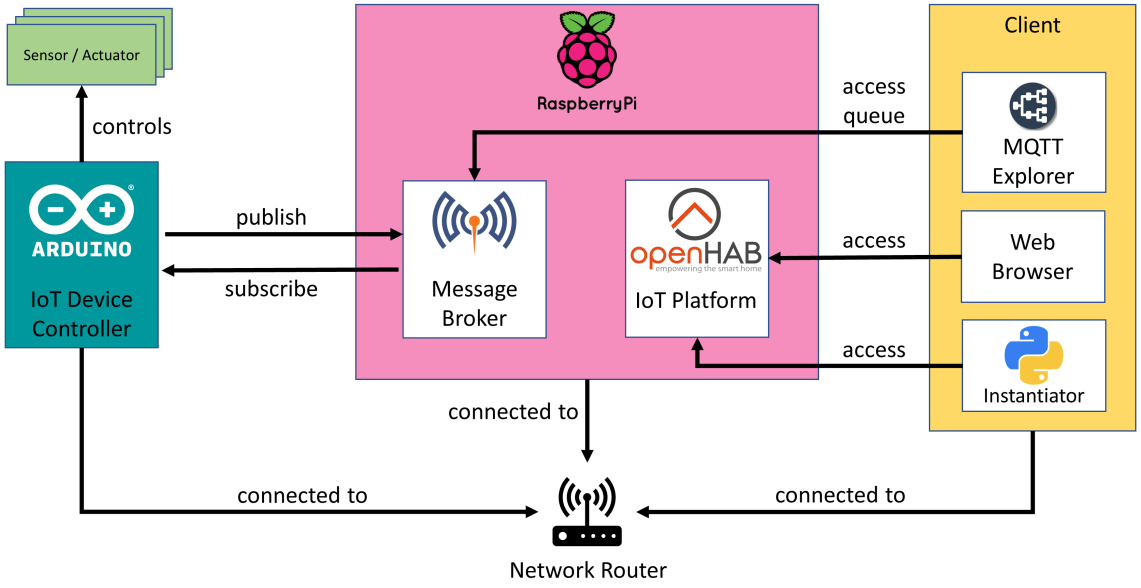
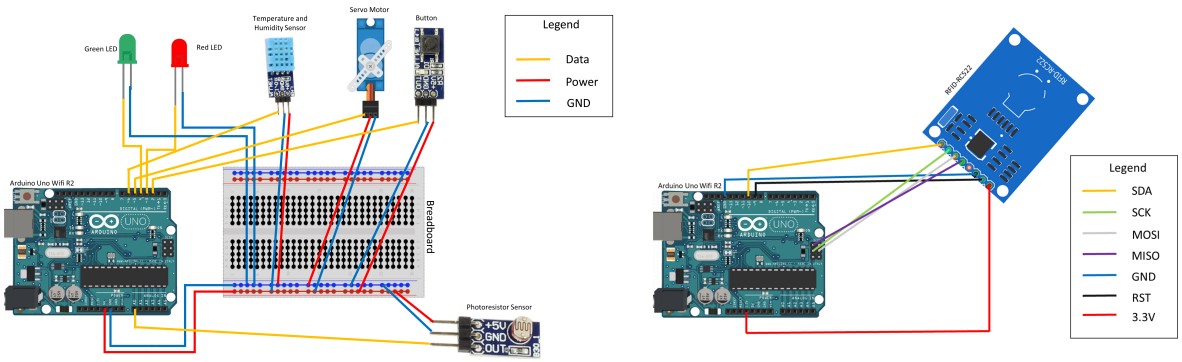


Fig. 4. Overview of the experiment environment architecture.



(a) IoT device pinout (b) RFID reader pinout

Fig. 5. Physical connections between the Arduino UNO WIFI and the IoT devices used in the Smart House scenario.

4.2.2. *Case-specific Utilization of the Instantiator Pipeline*

In the following, the utilization 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. Fig. 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* recognizes 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 utilized by a scheduler within the

```
{
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

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'.

In the next step, the IoT device fingerprints are processed by the *NeSy-Transformer* that utilizes a neuro-symbolic approach for the platform-specific semantic alignment procedure (cf. Fig. 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 utilized 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.

Finally, the API of the OpenHAB controller is utilized 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 synchronized with the physical IoT devices. Each of the resulting processes that utilize the item-specific information of enriched device fingerprints is represented as flowcharts in Fig. 6 and Fig. 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.

```
'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'

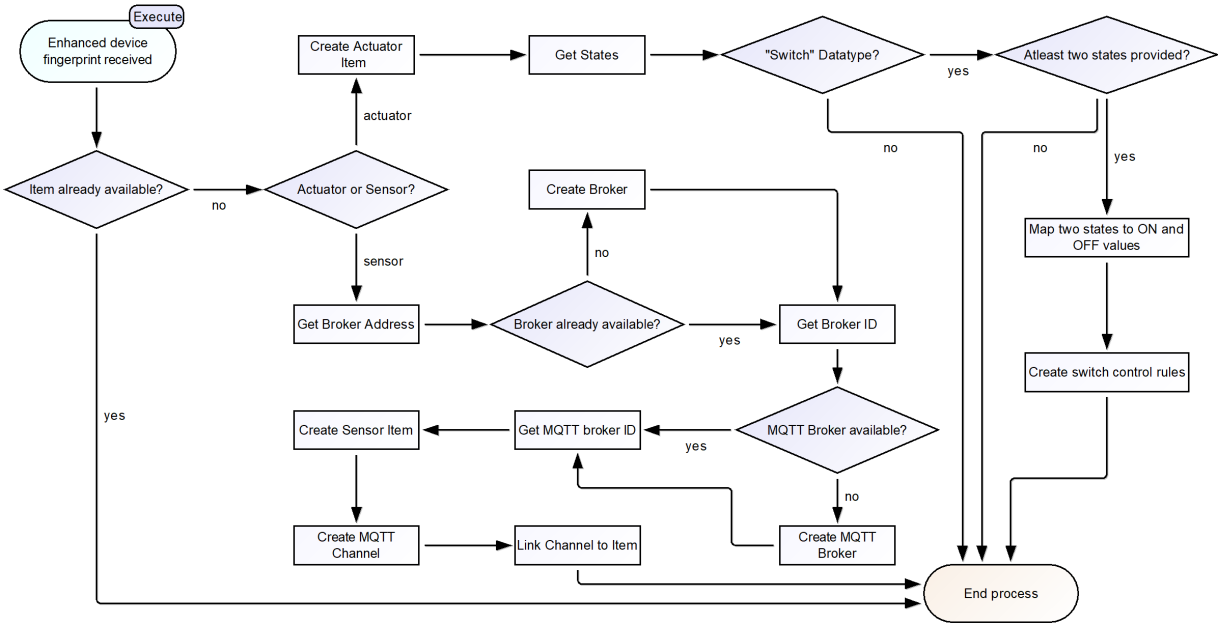


Fig. 6. Item creation pipeline of the OpenHAB controller.

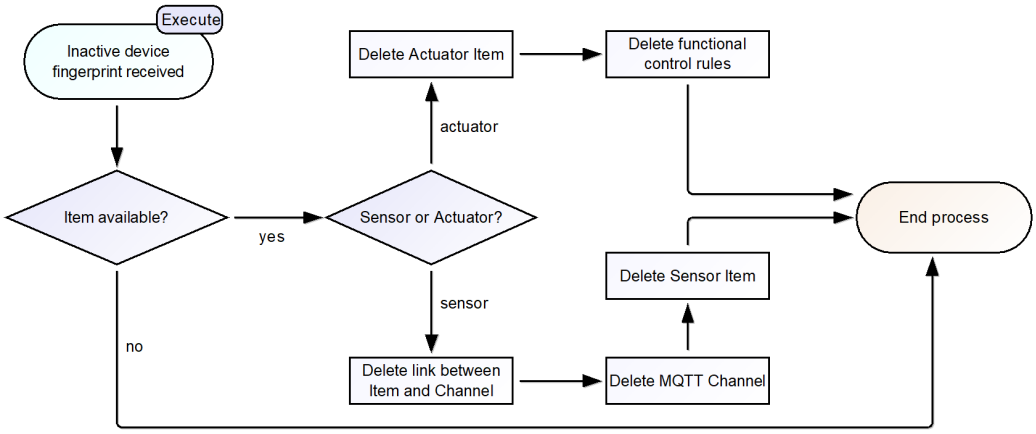


Fig. 7. Item deletion pipeline of the OpenHAB controller.

4.3. Scenario-based Configuration of IoT Platforms through Conceptual Modeling: 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), was added at a later point of the research and aims 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 [9, 11]. To serve this purpose of configuring scenarios within IoT environments, the ADOxx-based modeling method *IoT2Model* was created based on the GMMF and AMME (cf. Section 2.3), which allows users to design and evaluate scenarios through the modeling of abstract rules. It enables the bridging between the concepts required for a certain rule and the physical environment represented through the IoT platform of choice. The *IoT2Model* tool covers three different modeltypes, each responsible for modeling a scenario on a different abstraction level. The resulting metamodel, as implemented in ADOxx, is visualized in Fig. 8 using the CoChaCo tool [24]. 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 Fig. 9.

4.3.1. IoT Infrastructure Modeltype

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 Modeltype

The *Environment* model has the purpose of capturing the environment by modeling the physical components and IoT devices available in it. Devices are represented as 'IoT Device' concept within this modeltype, 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 modeled 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.

4.3.3. Scenario Modeltype

Lastly, the *Scenario* modeltype offers concepts for modeling 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 behaviors 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 specializations of the 'Rule Component' concept from the *Scenario* modeltype, users can define their desired scenario rules in 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 utilized, which is automatically added for each 'Rule Component' that is created through an event-based functionality. Afterwards, the respective references to the related modeling 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.

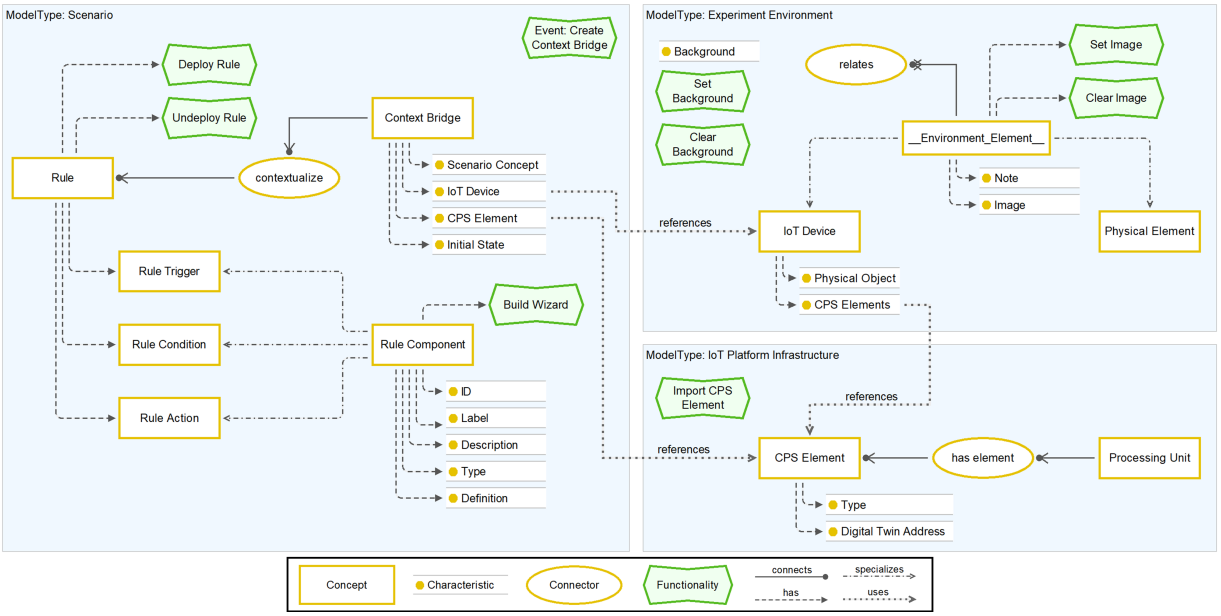


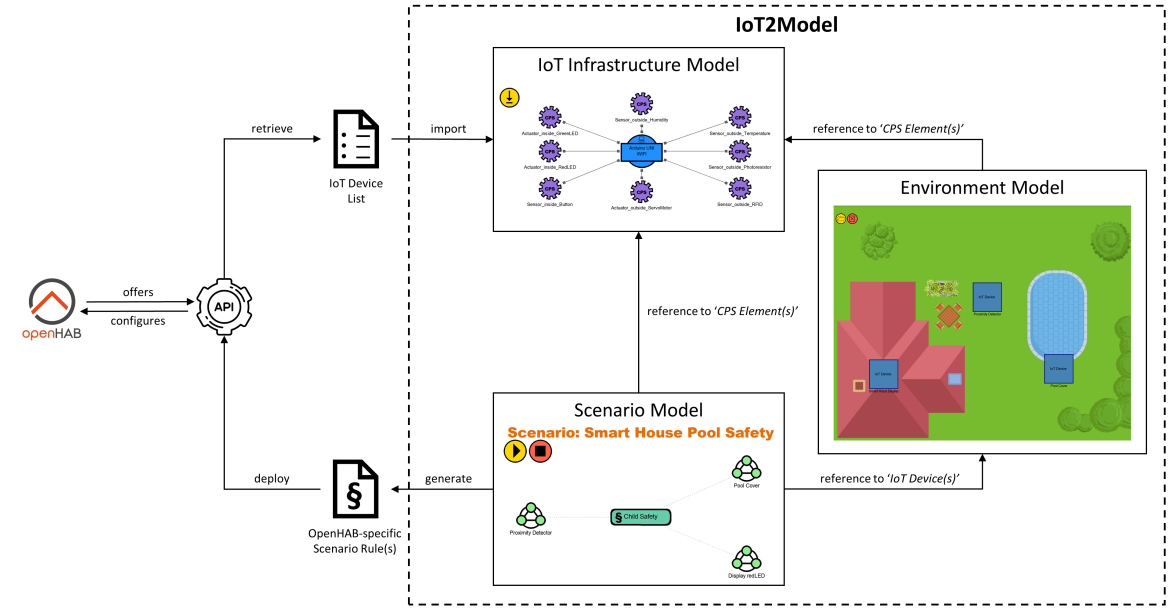
Fig. 8. IoT2Model Metamodel (created using CoChaCo tool)

4.3.4. *Pool Safety Configuration using IoT2Model*

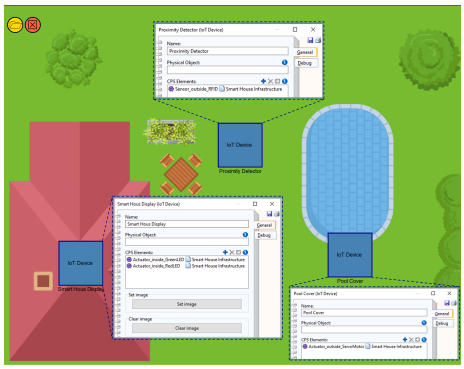
The process that is visualized in Fig. 9a assumes that an instance of the OpenHAB IoT platform has already been configured using the previously presented environment setup (cf. Section 4.2.1) and the *Instantiator Pipeline*. Consequently, the process starts off by automatically importing the IoT devices 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 modeled and linked to the respective CPS elements available within the *IoT Infrastructure* model. In Fig 9b, a detailed view of the modeled 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.

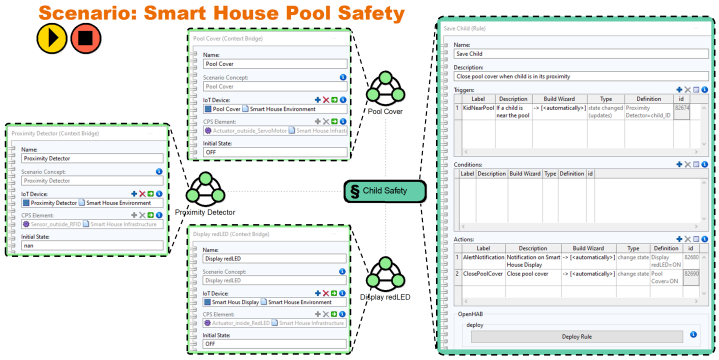
After modeling the environment, scenario-specific rules are defined within the *Scenario* model. This final step requires the specification of the relevant triggers, conditions, and actions, as displayed within Fig. 9c. First, a rule named 'Child Safety' is modeled, which is used to specify the needed rule components in an OpenHAB-specific manner with the help of a setup assistant (i.e. functionality 'Build Wizard' in Fig. 8). The trigger of the 'Child Safety' rule is defined by providing a meaningful label and description, before utilizing 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 the *IoT2Model* tool 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 Fig. 9c, two actions are specified, which change the state of the concepts 'Display redLED' and 'Pool Cover' to 'ON' while also creating the required 'Context Bridge' concepts 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 *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 or delete rules individually, or in case of several modeled rules, to deploy or delete all of them.



(a) Process of creating scenario-specific rules using the IoT2Model tool



(b) Environment Model (detailed view)



(c) Scenario Model (detailed view)

Fig. 9. Process view and detailed model view of utilizing the IoT2Model tool for the pool safety scenario

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 already been emphasized that this extension to the *Instantiator Pipeline* constitutes a significant part of holistic IoT platform configuration.

5. Discussion

The discussion aims to both motivate the extension of our approach through *IoT2Model* and highlight future work regarding the benefits of its utilization within educational settings.

The motivation for extending our approach with the *IoT2Model* method emerged from the realization that holistic IoT platform configuration extends beyond the registration of devices. While the initial steps of the *Instantiator Pipeline* effectively handle device discovery and basic configuration, they do not address the need for scenario-specific rules that govern how these devices interact within their environment. IoT environments are inherently dynamic, requiring adaptable systems that can not only integrate devices but also tailor their operations to meet specific user requirements or contextual conditions. The *IoT2Model* extension addresses this gap by enabling the

creation of user-defined models that represent these scenario-specific rules, thus allowing for more granular control and customization of IoT deployments in a low-code fashion.

This approach also exemplifies the broader integration of neuro-symbolic AI with conceptual modeling techniques, a synergy that showcases the objectives of semantic-driven systems engineering [8]. In that context, the prospects of combining LLMs and conceptual modeling to enhance the semantics and reasoning capabilities of engineering systems are discussed. Similarly, our work brings together the pattern recognition and learning strengths of neuro-symbolic AI with the simplification of complex configuration processes through conceptual modeling. This integration not only simplifies the configuration of IoT platforms but also ensures that the resulting system is effectively addressing 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 [17].

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 [48]. The motivation for this application context is the importance of developing a comprehensive Digital Leader skill profile, which we have highlighted in previous works [47]. This skill profile encompasses a set of competencies required to operate effectively in the interdisciplinary and technologically advanced environments that characterize modern digital ecosystems [21]. However, it is evident that this profile still requires adoption within Higher Education Institutions (HEIs) to meet the evolving demands of the digital era [25], especially regarding the issues of conceptual modeling education [7]. 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 modeling 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 by lecturers and presenters. By incorporating the *Instantiator Pipeline* and the *IoT2Model* method into such curricula, users can engage with a real-world example of how neuro-symbolic AI and conceptual modeling can be applied to enhance the configuration and customization 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 in addressing the seamless integration and dynamic customization required to fully realize the 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 modeling 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 customization 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 utilized as a testing ground for the application of the *IoT2Model* method in an educational setting, highlighting its potential for demonstrating the value of conceptual modeling 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.

References

[1] A. Aksoy and M.H. Gunes, Automated IoT Device Identification using Network Traffic, in: *2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–7. https://doi.org/10.1109/ICC.2019.8761559.

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

[3] K. Arulkumaran, M.P. Deisenroth, M. Brundage and A.A. Bharath, Deep Reinforcement Learning: A Brief Survey, *IEEE Signal Processing Magazine* **34**(6) (2017), 26–38. https://doi.org/10.1109/MSP.2017.2743240.

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

[5] L. Atzori, A. Iera and G. Morabito, The Internet of Things: A survey, *Computer Networks* **54**(15) (2010), 2787–2805. https://doi.org/10.1016/j.comnet.2010.05.010.

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

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

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

[9] 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** (2022), 100626. https://doi.org/10.1016/j.iot.2022.100626.

[10] L. Ehrlinger and W. Wöß, Towards a Definition of Knowledge Graphs, in: *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, M. Martin, M. Cuquet and E. Folmer, eds, CEUR-WS.org, 2016. https://ceur-ws.org/Vol-1695/paper4.pdf.

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

[12] E.A. Feigenbaum, Knowledge Engineering: The Applied Side of Artificial Intelligence, *Annals of the New York Academy of Sciences* **426**(1) (1984), 91–107. https://doi.org/10.1111/j.1749-6632.1984.tb16513.x.

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

[14] 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) (2017), 7. https://doi.org/10.9781/ijimai.2017.431.

[15] M. Hilario, *An Overview of Strategies for Neurosymbolic Integration*, in: *Connectionist-Symbolic Integration: From Unified to Hybrid Approaches*, Psychology Press, New York, 1997, pp. 13–35. https://doi.org/10.4324/9780203763667.

[16] 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)*, IEEE, Bangalore, India, 2008, pp. 791–798. https://doi.org/10.1109/COMSWA.2008.4554519.

[17] A. Javed, A. Malhi, T. Kinnunen and K. Främling, Scalable IoT Platform for Heterogeneous Devices in Smart Environments, *IEEE Access* **8** (2020), 211973–211985. https://doi.org/10.1109/ACCESS.2020.3039368.

[18] M.I. Jordan and T.M. Mitchell, Machine learning: Trends, perspectives, and prospects, *Science* **349**(6245) (2015), 255–260. https://doi.org/10.1126/science.aaa8415.

[19] D. Karagiannis, Agile Modeling Method Engineering, in: *Proceedings of the 19th Panhellenic Conference on Informatics*, N. Karanikolas, D. Akoumianakis, M. Nikolaidou, D. Vergados and M. Xeno, eds, Association for Computing Machinery, New York, NY, USA, 2015, pp. 5—10. https://doi.org/10.1145/2801948.2802040.

[20] D. Karagiannis and H. Kühn, Metamodelling Platforms, in: *E-Commerce and Web Technologies*, K. Bauknecht, A.M. Tjoa and G. Quirchmayr, eds, Springer, Berlin Heidelberg, 2002, p. 182. https://doi.org/10.1007/3-540-45705-4_19.

[21] D. Karagiannis, R.A. Buchmann and W. Utz, 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** (2022), 103631. https://doi.org/10.1016/j.compind.2022.103631.

[22] D. Karagiannis, H.C. Mayr and J. Mylopoulos (eds), *Domain-Specific Conceptual Modeling: Concepts, Methods and Tools*, 1st edn, Springer International Publishing, Cham, 2016. https://doi.org/10.1007/978-3-319-39417-6.

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

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

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

[26] D. Karagiannis, M. Lee, K. Hinkelmann and W. Utz (eds), *Domain-Specific Conceptual Modeling: Concepts, Methods and ADOxx Tools*, 1st edn, Springer International Publishing, Cham, 2022. https://doi.org/10.1007/978-3-030-93547-4.
[27] D. Kiritsis, Closed-loop PLM for intelligent products in the era of the Internet of things, *Computer-Aided Design* **43**(5) (2011), 479–501. https://doi.org/10.1016/j.cad.2010.03.002.
[28] S. Li, L.D. Xu and S. Zhao, The internet of things: a survey, *Information Systems Frontiers* **17**(2) (2015), 243–259. https://doi.org/10.1007/s10796-014-9492-7.
[29] S. Mayer, R. Verborgh, M. Kovatsch and F. Mattern, Smart Configuration of Smart Environments, *IEEE Transactions on Automation Science and Engineering* **13**(3) (2016), 1247–1255. https://doi.org/10.1109/TASE.2016.2533321.
[30] H.C. Mayr and B. Thalheim, The triptych of conceptual modeling, *Software and Systems Modeling* **20** (2021), 7–24. https://doi.org/10.1007/s10270-020-00836-z.
[31] A. Newell and H.A. Simon, *Computer Science as Empirical Inquiry: Symbols and Search*, in: *ACM Turing Award Lectures*, Association for Computing Machinery, New York, NY, USA, 2007, pp. 113–126. https://doi.org/10.1145/1283920.1283930.
[32] 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)*, 2019, pp. 67–73. https://doi.org/10.1109/IoTaIS47347.2019.8980370.
[33] K. Peffers, T. Tuunanen, M.A. Rothenberger and S. Chatterjee, A Design Science Research Methodology for Information Systems Research, *Journal of Management Information Systems* **24**(3) (2007), 45–77. https://doi.org/10.2753/MIS0742-1222240302.
[34] 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)*, 2017, pp. 1–10. https://doi.org/10.23919/CISTI.2017.7975691.
[35] S. Ray, A Quick Review of Machine Learning Algorithms, in: *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, IEEE, New York, 2019, pp. 35–39. https://doi.org/10.1109/COMITCon.2019.8862451.
[36] M.A. Razzaque, M. Milojevic-Jevric, A. Palade and S. Clarke, Middleware for Internet of Things: A Survey, *IEEE Internet of Things Journal* **3**(1) (2016), 70–95. https://doi.org/10.1109/JIOT.2015.2498900.
[37] C. Sanders, *Practical Packet Analysis: Using Wireshark to Solve Real-World Network Proble*, 2nd edn, No Starch Press, Burlingame, CA, 2011.
[38] J. Schmidhuber, Deep learning in neural networks: An overview, *Neural networks* **61** (2015), 85–117. https://doi.org/10.1016/j.neunet.2014.09.003.
[39] M.R. Shahid, G. Blanc, Z. Zhang and H. Debar, IoT Devices Recognition Through Network Traffic Analysis, in: *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 5187–5192. https://doi.org/10.1109/BigData.2018.8622243.
[40] M. Shanahan, Talking about Large Language Models, *Communication of the ACM* **67**(2) (2024), 68—79. https://doi.org/10.1145/3624724.
[41] B. Siciliano, L. Sciavicco, L. Villani and G. Oriolo, *Actuators and Sensors*, in: *Robotics: Modelling, Planning and Control*, Springer London, 2009, pp. 191–231. https://doi.org/10.1007/978-1-84628-642-1_5.
[42] M. Singh, E. Fuenmayor, E. Hinchy, Y. Qiao, N. Murray and D. Devine, Digital Twin: Origin to Future, *Applied System Innovation* **4**(2) (2021), 36. https://doi.org/10.3390/asi4020036.
[43] 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.
[44] 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** (2019), 127–139. https://doi.org/10.1016/j.autcon.2019.01.020.
[45] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L.u. Kaiser and I. Polosukhin, Attention is All you Need, in: *Advances in Neural Information Processing Systems*, Vol. 30, I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett, eds, Curran Associates, Inc., 2017, pp. 5998–6008.
[46] A. Voelz, D.M. Amlashi and M. Lee, Semantic Matching Through Knowledge Graphs: A Smart City Case, in: *Advanced Information Systems Engineering Workshops*, M. Ruiz and P. Soffer, eds, Springer International Publishing, Cham, 2023, pp. 92–104. https://doi.org/10.1007/978-3-031-34985-0_10.
[47] A. Voelz, C. Muck, D.M. Amlashi and D. Karagiannis, 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*, 2023, pp. 195–208. https://ceur-ws.org/Vol-3514/paper93.pdf.
[48] A. Völz and I. Vaidian, Digital Transformation through Conceptual Modeling: The NEMO Summer School Use Case, in: *Modellierung 2024*, Gesellschaft für Informatik e.V., Bonn, 2024, pp. 139–156. https://doi.org/10.18420/modellierung2024_014.
[49] R.J. Wieringa, *Design science methodology for information systems and software engineering*, Springer, 2014.
[50] Y.B. Zikria, R. Ali, M.K. Afzal and S.W. Kim, Next-Generation Internet of Things (IoT): Opportunities, Challenges, and Solutions, *Sensors* **21**(4) (2021). https://doi.org/10.3390/s21041174.