Dominik Scheiber, Bsc

IoT Middleware Platform for Populating Semantic Data of Buildings

Master's Thesis

to achieve the university degree of

Master of Science

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Wotawa, Franz, Univ.-Prof. Dipl.-Ing. Dr.techn.

Institute of Software Technology
Head: Wotawa, Franz, Univ.-Prof. Dipl.-Ing. Dr.techn.

Graz, July 2023

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Dissertation identisch.

Datum

Unterschrift

Abstract

Middleware platforms are an essential technology in Internet of Things (IoT) systems. They provide a unified and standardized way of managing interactions between devices, networks and applications. Modern buildings use middleware platforms as a key component of their IoT architecture. This allows for proactive energy management strategies that can result in significant benefits such as improved energy efficiency and comfort which results in reduced maintenance and operating costs. Due to the large number of components, this type of architecture may have varying communication protocols, data formats or contextual meanings, leading to semantic heterogeneity of data. This thesis aims to address the challenges of IoT heterogeneity and introduces a middleware application capable of capturing data from a building's IoT architecture. This work's research focuses on the use of common data models and population of ontology-based semantic metadata schemas based on the captured data. The use of an ontology-based metadata schema can provide a range of benefits, such as reusability, extensibility and consistency. Furthermore, the middleware application was tested on Raspberry PI hardware to access the building automation and control network (BACnet) of a non-residential building. The captured data of the buildings BACnet was used to populate an ontology-based metadata schema called Brick. The application was expanded to create a virtual environment, enabling the visualization and analysis of the extracted data. This virtual environment effectively simulates the behaviour and interactions of the sensor network. The middleware application in combination with the use of a simulated virtual environment provides a platform for testing and validating different scenarios. It can also be used to facilitate the training and evaluation of machine learning algorithms to optimize the performance of machine learning based services that analyze and optimize the energy efficiency of buildings.

Kurzfassung

Middleware-Plattformen sind ein wesentlicher Bestandteil in Systemen basierend auf dem Internet der Dinge (IoT). Sie bieten eine einheitliche und standardisierte Möglichkeit zur Verwaltung von Interaktionen zwischen Geräten, Netzwerken und Anwendungen. Moderne Gebäude verwenden Middleware-Plattformen als Schlüsselkomponente ihrer IoT-Architektur. Durch die Anwendung solcher Plattformen können proaktive Energiemanagementstrategien in die Realität umgesetzt werden. Die dadurch resultierenden Vorteile umfassen verbesserte Energieeffizienz und erhöhten Komfort, was wiederum zu einer Senkung der Wartungs- und Betriebskosten führt. Die hohe Anzahl an unterschiedlichen Komponenten, Datenformaten, kontextuellen Bedeutungen und der Einsatz von verschiedensten Kommunikationsprotokollen führt zur semantischen Heterogenität der Daten. Diese Arbeit befasst sich mit der Problemstellung von heterogenen IoT-Architekturen und stellt dabei eine Middleware-Applikation vor, welche auf Ebene von Gebäude-IoT-Netzwerken Daten sammelt und verarbeitet. Diese Arbeit konzentriert sich auf die Verwendung vereinheitlichter Datenmodelle und die Erstellung von ontologiebasierten, semantischen Metadatenschemen aus den gesammelten Daten. Durch die Verwendung von ontologiebasierten Datenmodellen sollen Wiederverwendbarkeit, Erweiterbarkeit und Beständigkeit der Daten sicher gestellt werden. Im praktischen Teil dieser Arbeit wird beschrieben, wie die Applikation auf Raspberry PI Hardware getestet wurde und der Zugriff auf das Gebäudeautomatisierungs- und Steuernetzwerk (BACnet) eines Testgebäudes realisiert wurde. Die erfassten Daten dienen als Basis um eine Brick Ontologie zu erstellen. Anschließend wurde die Anwendung erweitert, um auf BACnet basierende Geräte in einer virtuellen Umgebung zu simulieren. Ziel dabei ist es das Verhalten und die Interaktionen von BACnet Sensornetzwerken zu simulieren, um so die Visualisierung und Analyse eines IoT-Netzwerks zu ermöglichen. Die Kombination der Middleware-Applikation und der virtuellen Simulationsumgebung schafft eine Platfform zum Testen und Validieren verschiedener Szenarien. In weiterer Folge soll das Training und die Bewertung von Machine-Learning-Algorithmen, zur Analyse und Optimierung der Energieeffizienz in Gebäuden, unterstützt und erleichtert werden.

Contents

List of Figures

List of Tables

1 Introduction

Buildings are at the core of our daily lives, including our residences, workplaces, and recreational spaces. They play a vital role in our routines, demanding a significant portion of our time. Unfortunately, buildings consisting of residential, commercial, educational, and public structures, make a substantial contribution to the EU's energy consumption and carbon dioxide emissions (European Environment Agency, 2022). In fact, they are the largest energy consumers and a primary source of greenhouse gas emissions. The built environment alone accounts for 40% of energy consumption and 36% of greenhouse gas emissions in the EU, resulting from construction, usage, renovation, and demolition activities (Carlin, 2022). Therefore, prioritizing energy efficiency in buildings is essential to align with the objectives of the European Green Deal (Fetting, 2020) and achieve the goal of carbon neutrality by 2050. Currently, the energy efficiency of around 75% of buildings in the EU is inefficient in terms of energy, leading to substantial energy wastage (European Commission, 2020). To address this issue, it is crucial to focus on improving existing buildings and incorporating smart solutions and energy-efficient materials during the construction of new houses (Architecture 2030, 2022). The use of energy-efficient technologies and systems, such as efficient heating, ventilation, and air conditioning (HVAC) systems, lighting, and smart appliances, can further improve energy efficiency. By prioritizing these aspects, the goal of achieving Zero Energy Buildings can be effectively pursued (Cao et al., 2016).

1.1 Problem Statement

The observations about the energy consumption of buildings display a significant challenge in achieving sustainability objectives. The advancements in the Internet of Things (IoT) (Dorsemaine et al., 2015) and industry standards for device connectivity like BACnet (ASHRAE, 1995), result in vast amounts of data generated by these systems in buildings. Additionally, the generated data is typically characterized by heterogeneity (Da Cruz et al., 2018). Consequently, the data collections remain largely untapped for driving energy efficiency improvements. The lack of a standardized and coherent approach

for gathering and describing buildings' information limits the integration of diverse data sources and restricts the development of smart solutions.

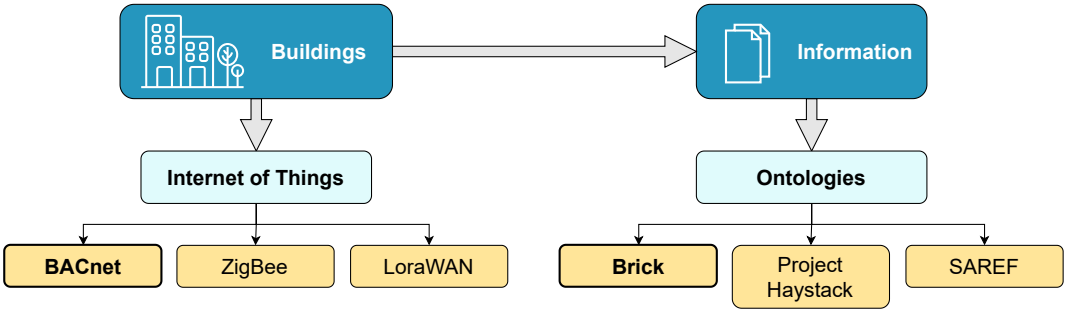

Figure 1.1: Gathering buildings information from IoT communication standards to populate ontology-driven metadata schemas

There is a pressing need for a solution that can automatically gather building information, specifically from the widely adopted and trending communication standard BACnet (Cimetrics Inc., 2019), in response to the industry's current demands for IoT development. The major challenge is to transform the gathered information into a semantic metadata schema like an ontology, which is illustrated in Figure 1.1. Ontologies can enable the standardization and harmonization of building data, facilitating interoperability and intelligent data-driven decision-making processes for energy efficiency improvements in buildings (Fensel, 2001; Manyika et al., 2015). The lack of a comprehensive semantic metadata schema constrains the ability to fully utilize the data resources available. Without a structured framework to organize and describe building information, the effective development and implementation of smart solutions becomes challenging.

1.2 Research Objectives

This thesis aims to address the problem at hand by proposing a solution that leverages an ontology-driven approach to automatically gather and describe building information, specifically focusing on utilizing the BACnet communication standard. By automatically creating standardized semantic metadata schemas, the proposed solution seeks to enable the integration and interoperability of data from various sources. This approach will support the development and implementation of smart solutions aimed at improving energy efficiency in buildings. In order to practically demonstrate the feasibility and effectiveness of the proposed solution, two distinct components will be developed and implemented. Firstly, an IoT middleware software script will be created to run on a Raspberry Pi within a real non-residential

building setting. This software script will serve as a practical example and proof of concept, showcasing the application of the proposed solution in a realistic manner. Secondly, the software script will enhance simulation capabilities, enabling the emulation of IoT devices and the creation of a simulated environment for software and network testing. This feature will provide a controlled setting for interacting with IoT devices and evaluating the functionality of a local IoT network.

The primary objective of this research is to bridge the gap between the availability of building data and its practical application in energy efficiency measures. By adopting an ontology-based approach and utilizing simulation capabilities of IoT devices and networks, the study aims to facilitate the efficient application of BACnet-generated IoT data for driving energy efficiency improvements in buildings. This research strives to contribute to the advancement of smart solutions in buildings.

1.3 Scope and Limitations

The primary focus of this research revolves around the selection of the BACnet communication standard and the Brick semantic metadata schema. The main goal of this research is to exploit the capabilities of BACnet for IoT device communication while leveraging the Brick ontology to improve interoperability and enable intelligent data-driven decision-making within the realm of building energy efficiency. These choices were made after careful consideration of a literature review, where BACnet has emerged as the most suitable and trending communication standard for seamless integration and interoperability of IoT devices. Similarly, after evaluating various metadata schemas, the Brick ontology metadata schema was deemed the most appropriate for representing building data in a standardized and coherent manner. It is important to highlight that the scope of this study is focused on the utilization of BACnet communication and the application of the Brick ontology, specifically within the context of building energy efficiency. This research does not encompass the evaluation or implementation of alternative communication protocols or ontologies. The comprehensive literature research and comparative studies conducted, are presented in Section 2.6 Related Work, providing valuable insights for the advancement of smart solutions in the building sector.

2 Background and Related Work

In this section, we will delve into the background and related literature, as well as explore the technologies employed in the study. Understanding the existing body of knowledge and the technological landscape is crucial for contextualizing the study and identifying the gaps that this research aims to address. By examining the relevant literature and exploring the technologies involved in the research process, a comprehensive understanding of the subject matter and the foundation for the study's objectives and contributions should be gained.

2.1 Industry 4.0

In 2011 the German federal government first mentioned the term Industry 4.0 as one of their key concepts about new high-tech strategies, which is also known as the fourth industrial revolution (Kagermann et al., 2013). Since then a great number of research articles and conferences have focused on this topic (Hermann et al., 2015; Wichmann et al., 2019). Despite the existence of a universal definition, Mohamed (2018) has collected and presented a collection of definitions of Industry 4.0. The most common term according to these definitions is Industry 4.0 being built upon a Cyber Physical System (CPS). CPS refers to a new way of engineered systems, that expand the physical world by enabling real-time monitoring and control of physical processes through the integration of computational and communication technologies (Baheti & Gill, 2011). Therefore IoT and CPS are often mentioned as the state-of-the-art technologies to form an Industry 4.0 standard (Devesh et al., 2020). According to Zhou et al. (2015) there are four key technologies to build this next stage of industrial development:

- Cyber Physical Systems (CPS)
- Mobile Internet and Internet of Things (IoT)
- Cloud Computing Technologies
- Big Data and Advanced Analysis Techniques

2.2 Smart Buildings

Since the 1980s definitions for Intelligent Buildings started to emerge. In a review of Intelligent Building research, Wong et al. (2005) shows that most early definitions describe an Intelligent Building as a self-controlling environment supported by numerous systems. Furthermore, the technical control of the building should be done by a computer system, with minimised human interaction (Powell, 1990). The term Intelligent Building evolved over the years and became the more frequent term Smart Building. Buckman et al. (2014) describes a Smart Building as a building system consisting of intelligence, enterprise and control, implemented in an adaptable manner. The system as a whole should be aware of context or time changes and should fulfil the requirements for building progression: energy and efficiency, longevity, comfort and satisfaction(Buckman et al., 2014). According to Lê et al. (2012) smart homes are characterized by five basic features:

- Automation: the ability to handle or perform automated tasks
- Multi-functionality: the ability to handle or produce multiple tasks or results
- Adaptability: the ability to meet a user's need through adjustments
- Interactivity: the ability to allow for interaction
- Efficiency: the ability to perform tasks time and cost saving

Achieving these features needs to digitize environmental conditions. Advanced sensor technology makes it possible to collect various physical information such as temperature, humidity, motion, light and sound in a smart building environment. Data capturing is an essential part to build a sensing and signal understanding infrastructure that is capable of detecting and monitoring contextual factors like environmental conditions or human behaviour. This infrastructure proposes a utilization for a wide range of purposes including energy saving and improved efficiency, enhanced comfort and improved security (Essa, 2000). To achieve all this, the idea of a Smart Building is directly related to the term Internet of Things (IoT) (Dorsemaine et al., 2015). The concept of IoT refers to a network in which the physical world is interconnected with the digital, creating a global ecosystem of sensors, actuators and data communication technologies (Brad & Murar, 2014).

2.3 Internet of Things

The emerging trend of internet-connected devices based on their physical environment is often referred to as the Internet of Things (IoT) (Dorsemaine

et al., 2015). IoT is associated with a wide range of technologies, systems and design principles. The main idea is to connect sensors and other devices to information and communication technologies through wireless or wired networks. It aims for real-world objects to connect, communicate and interact with their physical and digital environment, the same way humans interact with the web (Holler et al., 2014). Such IoT networks are core principles of a Smart Building (Brad & Murar, 2014). Currently, there exists a broad width of different network types and protocols on how an IoT network for Smart Buildings can be realized.

2.3.1 Wired Solutions

One of the key network types used in smart buildings is the so-called Building Automation and Control Network (BACnet) (ASHRAE, 1995) which is developed by the American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE[1]). It is a communication standard for building automation and control networks which provides a vendor-independent network protocol. The objective is to facilitate interoperability across different types of equipment, sensors and control devices by defining a networking solution based on communication messages, commands and rules for exchanging data and status information (ASHRAE, 1995).

At the same time, the widespread adoption of the internet revolutionized the way people communicate. Ethernet and the Internet Protocol (IP) became standards in communication, as we all know today. BACnet was considered to be a lightweight data protocol and most of the devices were not capable of computing the much larger Internet Protocol (IP) packets (Bushby & Newman, 2002). Increased computational power and decreasing costs for hardware also enhanced the development of lightweight communication. The ability to talk BACnet over the internet was one of the first expansions of the BACnet protocol stack. BACnet/IP is one of the new standards for BACnet and made it possible for BACnet devices to exchange messages via IP-based networks which also closes the gap to wireless solutions (Bushby & Newman, 2002).

Another communication protocol widely used in the industry is the Modbus protocol (Modbus Organization, 1979). It's based on a client-server architecture which indicates the whole network is built upon one centralized unit. Mostly it is used for connecting measurement and control systems and its more popular to be used in industry.

[1]https://www.ashrae.org/ (accessed 13 April 2023)

2.3.2 Wireless Solutions

Pretty similar to BACnet exists another networking specification, specialized in wireless networks. The ZigBee specification (Connectivity Standards Alliance, 2004) comes with benefits such as small data volumes and low energy consumption. It is ideal for low-bandwidth devices like lighting or temperature sensors. ZigBee is also considered to be one of the standards to build wireless IoT solutions (Kocakulak & Butun, 2017). As soon as it comes to more data-intensive devices like security cameras or access control systems, environments have to switch to e.g. WI-FI protocol (WI-FI Alliance, 1999).

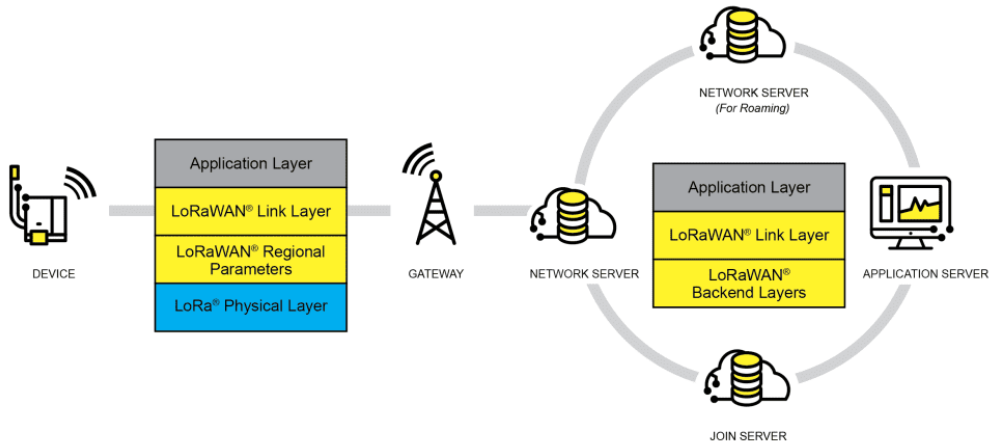

Figure 2.1: LoRaWAN network architecture (taken from LoRa Alliance, 2015; accessed 14 April 2023)

In 2015, the LoRa Alliance published a new network specification, based on the LoRa chipset from Semtech[2]. The network specification is called LoRaWAN (LoRa Alliance, 2015) which is the short form for Long Range Wide Area Network. A central networking server is connected to one or many gateways by a standardized IP connection. The complete network architecture can be seen as a star-of-stars topology where all the gateways connect end devices to the central network server. By using a specialised physical network layer, wireless communication benefits from single-hop connections between devices and gateways. Long range, low power consumption and bi-directional communication can be seen as the main advantages of this specification. Additionally the AES algorithm (National Institute of Standards and Technology, 2001) supplies all network packets with authenticity and integrity and also end-to-end encryption. Figure 2.1 gives a rough overview of the architecture of LoRaWAN environments (LoRa Alliance,

[2]https://www.semtech.com/lora (accessed 14 April 2023)

2015). Previously, LoRaWAN has been investigated as a potential solution for various types of smart city applications (Basford et al., 2020; Loriot et al., 2017; Rizzi et al., 2017).

2.4 Ontologies

Effective communication among people, organizations and software systems is crucial (Uschold & Gruninger, 1996). However, since they come from different backgrounds, and have different viewpoints or needs, there may be significant differences in perspectives and assumptions about the same subject matter. Each of them may use different terminology, and have divergent or overlapping concepts, structures and methodologies, leading to confusion or misunderstanding. When building an IT system, the differences in interpretation can result in challenges in identifying requirements and defining system specifications (Guarino, 1997). In addition, this heterogeneity significantly limits interoperability, reduces the potential for reusability and often results in redundant development. To address such problems, one of the most common approaches is to minimize or eliminate conceptual and terminological confusion by aiming to achieve a shared understanding in the form of a unifying framework (Uschold & Gruninger, 1996).

The term *Ontology* is used to describe such a shared understanding, which can function as a unified framework of a particular domain of interest (Uschold & Gruninger, 1996). In essence, an ontology incorporates or represents a specific worldview about a knowledge domain. This point of view is typically presented as a collection of concepts, including entities, attributes, definition and their interrelationships. This form of knowledge representation is also called conceptualisation, which forms a major constituent of ontologies (Guarino, 1997).

Within the scope of information and computer science, ontology refers to a set of building blocks used to represent a specific domain of knowledge. The fundamental elements are typically made of sets or classes, properties or attributes and relationships or connections among members of the classes. The definitions of these fundamental elements serve to clarify their intended meanings and specify limitations on how they can be logically used to ensure coherence and consistency (Gruber, 2008). Despite variations that may exist among ontologies, there is a broad agreement on numerous aspects. Some of the points of agreement within ontologies are that each object is part of the world and has properties or attributes which can be assigned values. These objects may be connected by relations. Properties and relations of objects can change over time and participate in processes that occur over

time. At distinct time points, events can happen and can also cause new events to be triggered. The objects and the world itself can be in varying states. Summarizing these points, a domain-dependent vocabulary can be established, providing a clear structure and conceptualisation of knowledge about that domain and forming its Ontology (Chandrasekaran et al., 1999).

Furthermore, Chandrasekaran et al. (1999) defined two dimensions of ontology specifications in knowledge systems:

- *Domain factual knowledge*: describes objective realities in a domain (objects, relations, ...)
- *Problem-solving knowledge*: describes how to achieve goals (problem-solving methods)

Information systems and AI commonly utilize these dimensions to acquire factual knowledge about their domains and to select from various reasoning or decision-making approaches. Information retrieval systems, digital libraries and internet search engines have to deal with heterogeneous information sources, where ontologies are used to organize information and search processes. Each object-oriented design of software represents and depends on an appropriate domain ontology. Additionally, AI can benefit from ontologies, as they assist in modelling extensive knowledge domains (Chandrasekaran et al., 1999). By using ontologies, AI systems can organize and structure information in a way that allows them to make more accurate predictions and decisions. As AI becomes more advanced and is applied in various industry scenarios, the importance of ontologies in this field will continue to grow (Chandrasekaran, 1994; Chandrasekaran et al., 1999).

2.4.1 Ontology Example

As an ontology example Figure 2.2 shows a visualized ontology about a small IoT network. Each object of the represented world is identified by a unique four-digit number. Each object has a specific role in the network and is in specific relationships with other objects or attributes. The corresponding extensional relational structure could then look as follows:

$$\begin{aligned} W &= \{1001, 1077, 1121\} \\ R &= \{Controller, Sensor, Setpoint, manages, reads, monitors, hasUnit\} \end{aligned} \tag{2.1}$$

A relational extension structure is a mathematical concept used to define and analyze sets based on a binary relation. Equation 2.1 shows each object of the network as part of the whole world W. Controller, Sensor and Setpoint

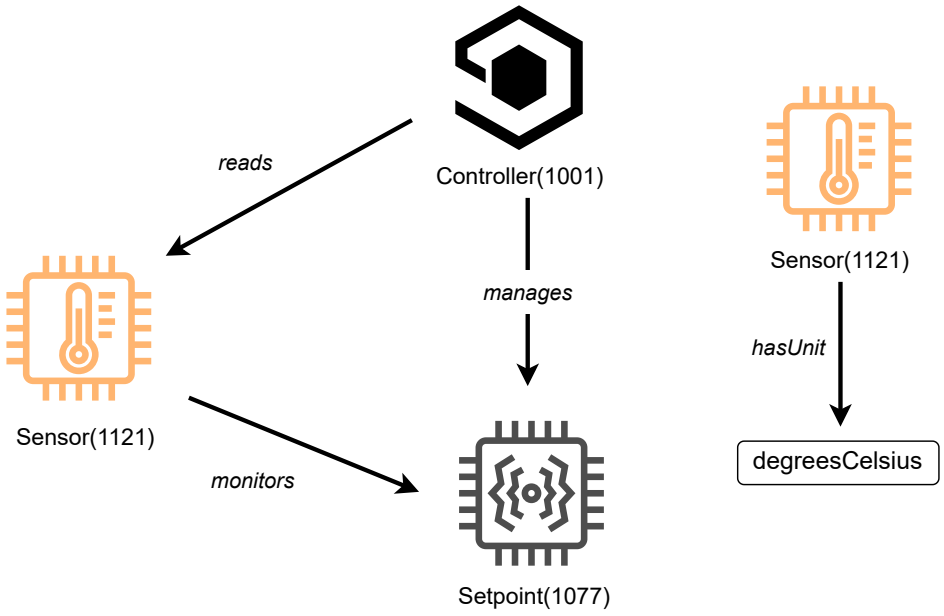

Figure 2.2: Visualized ontology example of a small IoT network including controllers, sensors, setpoints and their relations

are strict subsets of W whereas binary relations like *manages*, *reads*, *monitors* and *hasUnit* are sets of tuples which describe hierarchical relationships and collaborations. R represents the set of conceptual relations on the domain space. 2.2 shows each object with its corresponding role in the network. Additionally, it visualizes relationships, like the Controller *1001* reading the Sensor *1121* information and managing the Setpoint *1077*.

Furthermore, Sensor *1121* also monitors the Setpoint *1077*. In the final part, the relations to attributes are also shown as an example. Sensor *1121* is linked to the corresponding *hasUnit* attribute, which relates to the type of engineering unit the sensor is measuring (Staab & Studer, 2009).

$$\begin{aligned} Controller &= \{1001\} \\ Sensor &= \{1121\} \\ Setpoint &= \{1077\} \\ manages &= \{(1001, 1077)\} \\ reads &= \{(1001, 1121)\} \\ monitors &= \{(1121, 1077)\} \\ hasUnit &= \{(1121, \text{'degreesCelsius'})\} \end{aligned} \tag{2.2}$$

2.4.2 Ontologies for the Building Sector

The knowledge representation and reasoning by ontologies, as described in the previous section, have become crucial tools in various domains. Because the world in all its detail richness can not simply be covered by one ontological representation, there exist ontology formats for specific domains such as buildings. The building domain has some special systems and equipment that need to be treated by the ontology. The official documentation of the Brick Ontology (2023) has listed a spectrum of equipment and sensors that most commonly exist in buildings:

- *Heating, Ventilating and Air Conditioning (HVAC) Systems*: responsible for thermal comfort and indoor air quality
- *Lighting Systems*: providing adequate lighting levels
- *Electrical Systems*: power and energy management
- *Spatial Information*: information about the physical layout of a building (locations, room sizes, ...)
- *Sensor Systems*: monitoring of various aspects of a building's performance (temperature, humidity, occupancy, ...)
- *Control Relationships*: relationships between different systems, such as the HVAC system and the lighting system
- *Operational Relationships*: relationships between different operations, such as energy consumption and maintenance
- *Formal Definitions*: precise and unambiguous definitions of the concepts and relationships in a building ontology

Providing a common vocabulary and a shared understanding of the building domain can facilitate communication and interoperability. Most of the use cases deal with energy audits, fault diagnosis and detection, and optimal control. To achieve these goals, semantic metadata schemes are widely used and evaluated. Three of the most common state-of-the-art ontologies are described in the following sections (Mahdavi & Taheri, 2017; Qiang et al., 2023).

Brick Schema Ontology

Brick (Brick Ontology, 2023) is an open-source project that aims to standardize the description of physical, logical, and virtual assets in buildings and their relationships. It consists of a flexible data model, an extensible dictionary of terms and concepts related to buildings, and a set of relationships that link and compose these concepts. By using semantic web technology, Brick can describe characteristic features and subsystems found in buildings in a consistent manner. Adopting Brick as the standard description of a

building can help lower costs for deploying analytics, energy efficiency measures, and intelligent controls across buildings. It can present an integrated, cross-vendor representation of building subsystems, which simplifies the development of smart analytics and control applications, whereas it also reduces the reliance on non-standard labels used in building management systems. Brick is free and open-source under the BSD 3-Clause licences and its source, website and related tools are available on GitHub[3]. Overall, Brick offers a comprehensive, extensible, and consistent approach for standardizing semantic descriptions (Brick Ontology, 2023).

Project Haystack

Project Haystack (Project Haystack, 2023) is a suite of open-source technologies designed for modelling IoT data in a consistent and interoperable way. The source code for the ontology's definition can be found on GitHub[4]. The stack consists of several components, including a fixed set of general-purpose data types known as *kinds*, text formats for encoding and exchanging data and a protocol for exchanging data over HTTP. It consists of a standard ontology for modelling common building concepts such as equipment, sensors, and a way to define and extend the ontology. The primary collection type in Haystack is the *dict* (short for dictionary), which models entities using name-value pairs called tags. These tags inform about what type of entity is being modelled and facts about that entity. Altogether, Project Haystack aims to foster the interoperable exchange of data by providing an extensible and consistent set of technologies for modelling IoT data (Project Haystack, 2023).

Smart Applications REFerence (SAREF)

SAREF (SAREF, 2023), or the Smart Applications REFerence ontology, is a shared model of consensus that aims to facilitate the matching of existing assets in the smart applications domain. It provides building blocks that allow the separation and recombination of different parts of the ontology depending on specific needs. The ontology explicitly specifies recurring core concepts in the smart applications domain, the main relationships between these concepts, and axioms to constrain their usage. The principles of SAREF are centred on reusing and aligning existing concepts and relationships, allowing for modularity to separate and recombine different parts of the ontology. It promotes extensibility to support future growth, while also

[3]https://github.com/BrickSchema (accessed 03 May 2023)
[4]https://github.com/Project-Haystack/ (accessed 04 May 2023)

prioritizing maintainability to assist in identifying and correcting defects, accommodating new requirements, and adapting to changes in the ontology (SAREF, 2023).

2.5 Technologies

The following section provides a comprehensive overview of the technologies that were utilized in the context of this study. Hardware, software and various tools and techniques have been used throughout the research process. The major task was to develop software that is capable of collecting, analyzing and processing data, whereas each of the described technologies comes with its own benefits and impacts on the outcome of the study.

2.5.1 Python

Python (Python, 2023) is a high-level, interpreted programming language that is widely used in various fields. It has gained popularity due to its simplicity, readability and ease of use, making it accessible for both beginner and experienced programmers. Python's syntax is straightforward and intuitive, allowing for faster and more efficient coding. In the annual Stackoverflow developers survey (Stackoverflow, 2022a), Python is mentioned as one of the most loved and wanted programming languages, which highlights its enduring appeal and strong demand among developers. The availability of various Python libraries such as BACpypes, BAC0, brickschema, pyModis or pySHACL, offers extensive functionalities that are particularly useful for data analysis, visualization and machine learning for the building automation sector. Additionally, its open-source nature and a great community of developers ensure that it is constantly updated and improved. The practical experiment of this thesis also requires the software to run on lightweight hardware like the Raspberry PI. Python itself is known as a lightweight scripting language which makes it an ideal choice for hardware with limited processing power, memory and storage.

2.5.2 Raspberry PI

The Raspberry Pi (Raspberry Pi Ltd, 2023) is a powerful, lightweight single-board computer that is widely used for IoT applications, due to its versatility and affordability. For the purpose of this thesis, lightweight hardware had to be chosen, to deploy and test the final software script. Finally, the Raspberry

PI 4 Model B[5] has been selected for the practical experiment and can be seen in Figure 2.3. The Raspberry PI 4 Model B is one of the latest models in the Raspberry PI series and comes with significant improvements in processing power, memory, and connectivity compared to its predecessors. It is equipped with a Broadcom BCM2711 quad core cortex-A72 (ARM v8), 64-bit processing unit, which runs at 1.8GHz. This offers a significant increase in performance compared to previous models. It also features up to 8GB of RAM, allowing it to handle multiple applications simultaneously. The networking modules of the PI 4 consist of a dual-band 802.11ac wireless networking unit, Bluetooth 5.0 and Gigabit Ethernet connectivity, making it a perfect choice for connecting with networks or other devices (Raspberry Pi Ltd, 2023).

One of the main reasons why the Raspberry PI 4 Model B is a popular choice for IoT deployment, is because of its General Purpose Input Output (GPIO) pins, which allow it to interact with a wide range of sensors and other hardware components. This makes it ideal for projects that involve collecting and analyzing data from sensors such as temperature, humidity, motion and light. Another benefit of using Raspberry PI for IoT deployment is the availability and capability of a wide range of software and programming languages. This includes the most popular languages such as Python, Java, and C++, which makes it easy for developers to build and deploy applications. In conclusion, the Raspberry PI 4 Model B is an excellent choice for deploying IoT software, due to its affordable price, processing power, memory, network connectivity and support of programming languages (Raspberry Pi Ltd, 2023).

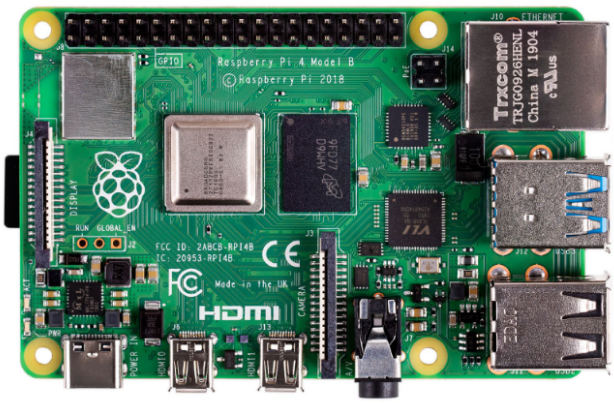

Figure 2.3: Raspberry PI 4 Model B from a top-down view (taken from Raspberry Pi Ltd, 2023; accessed 12 May 2023)

[5]https://www.raspberrypi.com/products/raspberry-pi-4-model-b/ (accessed 12 May 2023)

2.5.3 Docker

Docker (Docker Inc., 2022) is a popular containerization platform that allows developers to package and deploy applications in a lightweight, portable, and isolated environment. According to the annual Stackoverflow survey (Stackoverflow, 2022b), Docker has seen a significant rise in popularity and has come close to becoming one of the most widely used tools among developers. It provides a way to package an application with all its dependencies, libraries, and configuration files in a single container. Each container then can be run on any machine with Docker installed. As Figure 2.4 shows, Docker is built upon a traditional operating system, such as Linux or Windows. It leverages the host's kernel to run containers. Unlike a virtual machine, which emulates an entire operating system and requires significant system resources, Docker containers share the same kernel as the host operating system. This makes them lightweight and portable, while still providing a level of isolation and security. To achieve this, Docker utilizes a layered filesystem and copies only the necessary dependencies and libraries into each container, reducing the overall footprint of applications (Docker Inc., 2022).

To configure a Docker container, a so-called Dockerfile serves as a blueprint for defining the environment of a container. The Dockerfile is a text document that specifies the base image and defines dependencies and runtime commands. Typically a simple syntax is used, which makes the files portable and helps define runtime environments for applications. On the other hand, a Compose file collaborates with Docker Compose (Docker Inc., 2023), a tool designed for defining and executing multi-container Docker applications based on their appropriate Dockerfiles. Written in YAML format (YAML, 2021), the Compose file describes the services, networks and volumes required by an application. It defines relationships between different containers, specifies environment variables, manages shared resources and sets up network configurations. This helps in managing and deploying application stacks, with multiple connected containers.

One of the key features of Docker is its virtual network, which allows containers to communicate with each other and with external networks. Docker provides a default bridge network that allows containers to communicate with each other on the same host. It's also possible to create custom networks to isolate and control the flow of traffic between containers. By default, each container is assigned its own IP address, and Docker provides tools for managing and monitoring the network traffic between containers. This virtual network allows developers to create complex multi-container applications that can communicate and exchange data, without the need for complex network configurations or infrastructure (Docker Inc., 2022).

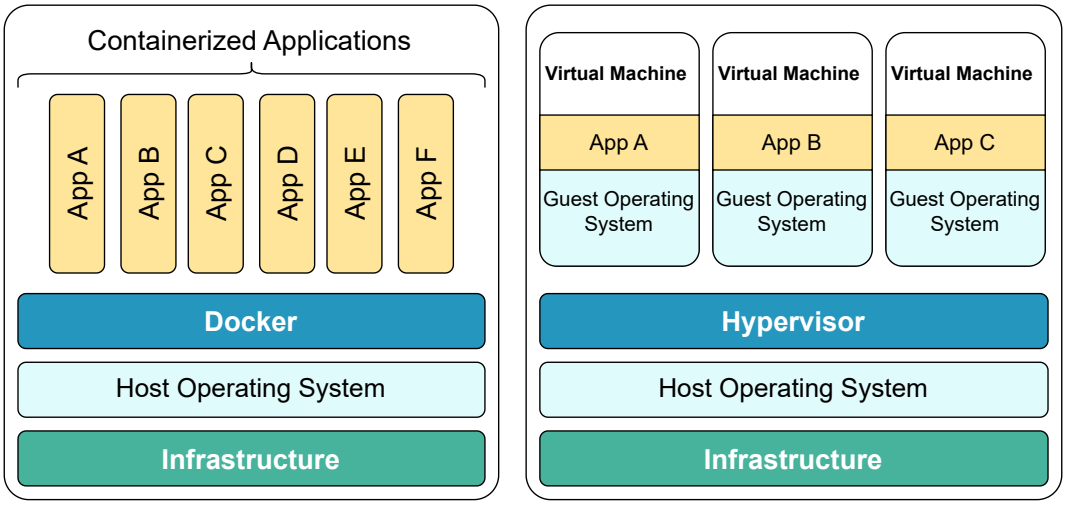

Figure 2.4: Comparison of resource allocation and isolation between Docker and Virtual Machines (Docker Inc., 2022)

The primary focus of this research is to collect and analyze IoT data, with an additional goal of exploring the potential of simulating IoT devices. Aiming to accurately replicate the behaviour and characteristics of real-world IoT devices, which use the BACnet/IP communication protocol, Docker was used as the main environment for this simulation task. Each simulated device is defined by a configuration file, which passes to the dockerized simulation environment. Finally, each device is represented by a container running on the same host, making it possible to simulate multiple devices on one single host machine. This results in perfect isolation for each of the simulated devices, which use the virtualized Docker network to communicate with each other. The summarized result is a virtual network of IoT devices, where each device is represented by a container. Each of them is assigned a unique IP address, to ensure the ability to communicate properly over the virtual Docker network by using the BACnet/IP network protocol.

2.5.4 Wireshark

The Wireshark tool (Wireshark Foundation, 2023) is a popular open-source network protocol analyzer, developed by the Wireshark Foundation. The source code for the protocol analyzer can be found on GitHub[6]. Nowadays it has become a standard for network administrators, security professionals, and developers. It is capable of capturing and analyzing network traffic in real time, providing detailed information about the network packets and their contents. This is useful to gain a deep understanding of how network

[6]https://github.com/wireshark (accessed 12 May 2023)

traffic flows in a system and how devices interact with each other. The tool comes with a user-friendly graphical interface and a broad variety of filter options, which makes it an easy and versatile tool to analyze network traffic. Wireshark supports a wide range of network protocols, including TCP/IP, UDP, SSH and many more. Furthermore, Wireshark is an effective tool for network performance analysis. It offers comprehensive statistics on network metrics such as throughput, packet loss, and latency, which can assist in detecting performance issues and identifying bottlenecks (Wireshark Foundation, 2023). As this research aims to gain a deeper understanding of how BACnet/IP is used in various building automation systems, Wireshark has emerged as a valuable tool. The tool has built-in support for the BACnet/IP network protocol and therefore can assist in monitoring and troubleshooting BACnet-based systems. The final practical example used Wireshark to monitor the communication between the Raspberry PI and the BACnet devices. The existence of a Wireshark Docker image additionally makes it possible to capture network traffic of a virtualized Docker network, which was used to monitor the simulated devices and their communication.

2.5.5 Agile Software Development Methods

Agile software development methods (Schwaber & Beedle, 2002) have gained widespread popularity in the technology industry, due to their flexible and iterative approach to project management. Agile methodologies prioritize collaboration, adaptability and customer satisfaction, emphasizing the delivery of working software in short, incremental cycles. Some popular agile methods include Scrum, Extreme Programming (XP) and Lean Development (Abrahamsson et al., 2017). As an agile software development enthusiast, I believe in its effectiveness for team collaboration. However, in the case of this project, some agile principles couldn't be fully implemented because it was a solo work. Despite these limitations, I still strived to maintain an agile mindset by being able to adapt to changes and iterate on the project as needed. Having previously worked as a mobile app developer, I brought my knowledge of some of the core principles of agile methodologies to this project. Two key principles used throughout this research project were the implementation of a Kanban board and regular code reviews. The Kanban board provided a visual representation of the project's workflow, allowing for better task management, transparency and tracking of progress (Ahmad et al., 2013). Code reviews, on the other hand, ensured that the quality of the code remained high by catching any potential issues or requirement changes early in the development process (Baker Jr, 1997; Fagan, 2002). The decision to employ agile methods in this project was driven by the fact that the requirements and especially the environment were not initially

clear in detail. Agile methodologies are well suited for situations where requirements may evolve or change over time.

2.6 Related Work

The Internet of Things (IoT) (Dorsemaine et al., 2015) can be described as a pervasive and intricate network consisting of diverse and interconnected entities or *things*, that are capable of being networked (Bertin et al., 2013). The IoT market is flooded with numerous vendors offering a wide range of IoT devices, many of which have their own unique firmware incorporating exclusive semantic models and communication protocols. Consequently, the very nature of IoT systems, which involves connecting millions of individual devices, inherently leads to a high degree of heterogeneity (Da Cruz et al., 2018). The role of middleware in this context is to serve as an intermediary layer between devices and applications, as illustrated in Figure 2.5. Middleware applications should propose the seamless integration of data from the physical environment into IoT-connected devices, networks and services. The integration of middleware platforms encompasses essential operations like data storage, analysis, and processing to enhance connectivity among diverse devices and programs that were not initially designed to support such functionality (Agarwal & Alam, 2020; Razzaque et al., 2016).

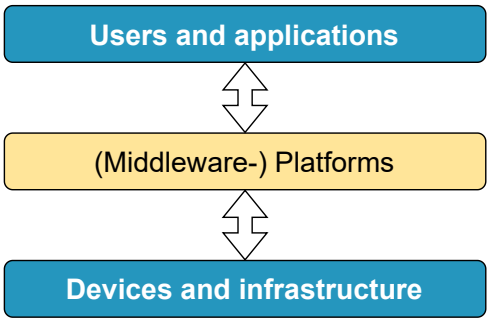

Figure 2.5: Simple representation of a layered IoT architecture

Ngu et al. (2016) identified and examined middleware platforms of different architectural types and categories such as ambient data collection and real-time reactive applications. The first type is an application that typically gathers sensor data which gets processed with no need for real-time interactions. The data is then processed offline to extract meaningful insights such as developing predictive models. Furthermore Ngu et al. (2016) addresses four major challenges when it comes to IoT middleware development:

- Developing a lightweight platform that can operate efficiently on power-constrained devices

- Creating an application-independent composition engine
- Establishing a security mechanism that is compatible with environments with limited resources
- Implementing a semantic-based approach for integrating IoT devices and services

Not only the development of middleware platforms comes with challenges, but Chaqfeh and Mohamed (2012) also mentioned challenges for provided solutions such as interoperability, scalability, security and privacy. The challenge of semantic interoperability is the main part being tackled by this thesis. Semantic interoperability issues often emerge due to ambiguities from the absence of standardized metadata schemas within a heterogeneous system. Resolving this issue is a crucial task to embrace data-driven decision-making and AI in the context of IoT (Manyika et al., 2015). This involves harmonizing existing metadata schema efforts and considering established standards such as BACnet, while also leveraging ontologies. The standardization of semantic information is essential for the cost-effective and functional delivery of data-driven smart building technologies (Bergmann et al., 2020). Despite these findings, a recent expert survey conducted by Alfalouji et al. (2022) on IoT middleware platforms revealed that over 80% of participants do not employ a unified data model. However, it is noteworthy that they still recognize the significance of developing semantic models.

Since the early 1990s, ontologies have gained significant attention as a prominent research subject in various Artificial Intelligence research communities (Fensel, 2001). These communities encompass fields such as *Knowledge Engineering*, *Natural Language Processing* and *Knowledge Representation*. By enabling the organization and classification of knowledge, ontologies can lead to efficient information retrieval and sharing (Fensel, 2001). In order for these standards to be impactful, they need to possess clear and easily expandable definitions, promote consistent usage, and seamlessly integrate with established and trending industrial standards like BACnet (Cimetrics Inc., 2019). However, a natural conflict arises between informal tag-based systems, which rely on idioms and conventions for meaning, and formal ontologies that are adaptable to automated tools (Fierro et al., 2019). An existing study (Pritoni et al., 2021) focused on five ontologies (BOT, SSN, SAREF, RealEstateCore, Brick) related to building operations, evaluating their concepts and identifying gaps and overlaps. This study emphasizes the need for semantic interoperability in building systems by three use cases: Energy Audits, Automated Fault Detection and Diagnostics (AFDD), and optimal control of HVAC systems. The findings contribute to improving semantic interoperability and energy-efficient building systems by examining a semantic metadata schema and their relevance to the specific use cases (Pritoni et al., 2021).

Brick (Balaji et al., 2016; Brick Ontology, 2023) and Project Haystack (Project Haystack, 2023) are widely recognized as two of the most popular ontologies for building systems, known for their extensive adoption and industry-wide recognition in the field (Quinn & McArthur, 2021). Project Haystack is an object-centred ontology that employs a tagging model, while Brick is a description logic ontology with a hierarchical design. In a review, both ontologies have been compared to each other in terms of completeness, expressiveness and qualitative assessments like flexibility, portability, readability and queryability, with Brick achieving higher scores in nearly all categories (Quinn & McArthur, 2021). Literature also discusses customized extended frameworks based on Brick or Haystack like *Brick+* (Fierro et al., 2019) or *Energy Flexibility Ontology (EFOnt)* (Li & Hong, 2022). Ploennigs and Schumann (2017) conducted an experiment on a campus equipped with 3,330 sensors, serving as an illustration of semantic modelling. They utilized a pre-modelled Brick ontology to facilitate tasks such as anomaly detection and diagnosis through the combination of machine learning and semantic reasoning. This approach involved mapping the tasks to semantic concepts and deducing physical relationships based on them.

3 BACnet

BACnet (ASHRAE, 1995) was initially created in 1987 with the support of the American Society of Heating Refrigerating and Air-conditioning Engineers (ASHRAE). Since 1995, it has been recognized as an ANSI standard and became an ISO standard in 2003. BACnet is a registered trademark owned by ASHRAE. Opting for BACnet as the main communication protocol retains its significance for various compelling factors. First and foremost, BACnet retains its global recognition as the widely adopted standard for building automation and control networks. Its established reputation and strong backing from the industry make it a dependable choice when it comes to ensuring seamless interoperability and safeguarding the longevity of a building automation system (ASHRAE, 1995). Furthermore, BACnet has consistently evolved to keep pace with technological advancements and industry demands. The protocol offers a rich ecosystem of tools, libraries, and resources that simplify the development, deployment, and maintenance of BACnet-based solutions. There are numerous commercial and open-source implementations available, along with a vibrant community of developers and experts. Additionally, the protocol has embraced contemporary communication technologies, such as BACnet/IP, which facilitates effortless integration with Ethernet and IP networks. This capability allows for flexible and scalable implementations, providing specific requirements of diverse projects (Bushby & Newman, 2002).

Choosing BACnet as the communication protocol for building automation and control offers a solid foundation for smart solutions and continuous improvements. According to reports, BACnet is currently the leading global protocol, experiencing growing demand in every country worldwide, while other protocols are declining (Cimetrics Inc., 2019). The provided report is titled "Market Penetration of Communication Protocols" (BSRIA, 2018) and was conducted by BSRIA[1] (Building Services Research and Information Association). It offers comprehensive insights into communication protocols, covering data from 2012 to 2018, derived from primary and secondary research. The study highlights the significance of communication protocols in enabling interoperability among building systems and devices, particularly

[1] https://www.bsria.com/ (accessed 04 May 2023)

for building automation and control applications. It confirms the prominence of BACnet as a preferred protocol for smart building solutions and emphasizes the preference for vendor-independent, open protocols with certification schemes (BSRIA, 2018).

3.1 Certification of Devices

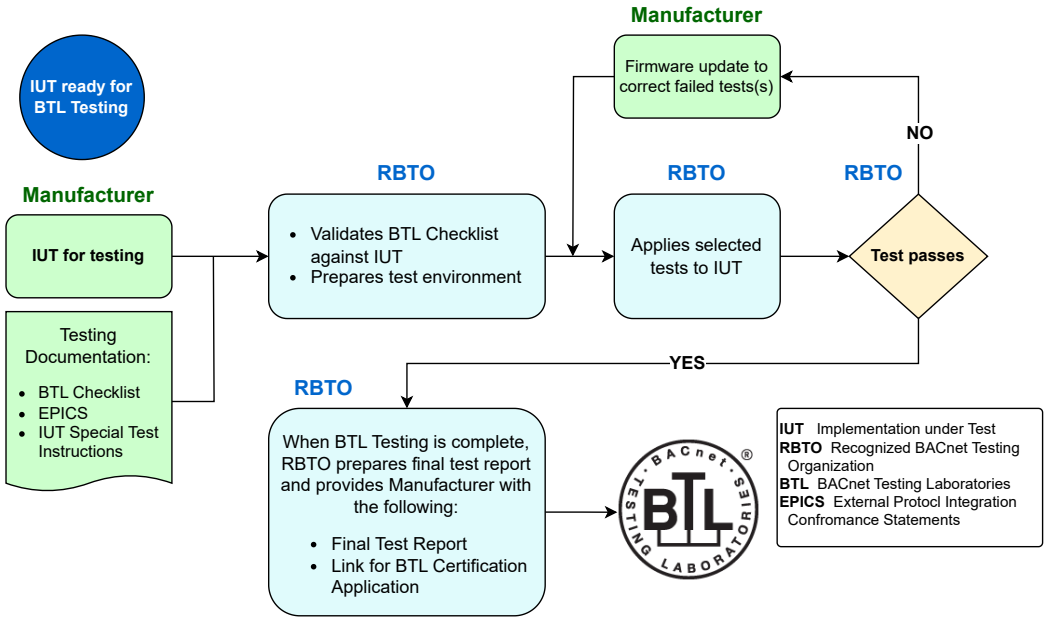

Figure 3.1: Workflow of a certification process to test an implementation against BACnet conformance (BTL Working Group, 2023)

The BACnet Testing Laboratories (BTL) Certification program (BTL, 2022) plays a crucial role in guaranteeing that these products are correctly implemented. Through independent testing for compliance and interoperability, the BTL certifies products according to the BACnet industry standard. Certified products can be identified by the BTL Mark, either displayed on the physical product or found in the online database from BTL (2022). The use of BTL Certified products offers several benefits. It reduces integration time and costs, as these products have already been tested and proven to work well together. This saves building owner or operators, controls specifiers, and system implementers valuable time and resources. Secondly, BTL Certification provides a solid foundation for future system enhancements and extensions. It ensures that new components can be seamlessly integrated into the existing BACnet system without compatibility issues (BACnet International, 2023). The certification process involves independent testing

by a recognized test organization, adhering to the testing requirements established by the BTL. The BTL Certification program is administered by BACnet International, a global collaboration platform for BACnet professionals and organizations. The BTL is responsible for a range of critical tasks and activities, including the development and maintenance of BACnet product test requirements and testing policies.

The BACnet Testing Laboratories are supported by the BTL Working Group (BTL-WG) (BTL Working Group, 2023), composed of voluntary members and the BTL Manager, established by BACnet International to uphold and enhance the BACnet conformance certification and listing program. The working group consists of members from BACnet International and BACnet Interest Group Europe[2] (BIG-EU), selected because of their expertise in the BACnet community. With a focus on improving BACnet interoperability, the BTL-WG provides oversight and guidance to the BTL and its certification program. As a technical resource, the BTL-WG maintains and expands the BACnet product test requirements and testing policies. Additionally, they developed a test package that manufacturers can utilize to pre-test their products for BACnet compliance before submitting them to a Recognized BACnet Testing Organization (RBTO). After successful evaluation of such a pre-test package, a manufacturer is able to submit their implementation to gain the BTL mark from an RBTO, as illustrated in Figure 3.1. The BTL Manager leads the organization, with support from the BTL Technical Advisor. Moreover, the BTL plays an important role in recognizing testing laboratories that are suitable for conducting certification testing. This ensures that the testing process is carried out by reliable and competent entities. The responsibilities of the BTL-WG encompass publishing the BTL Test Package, offering support to the BACnet community, evaluating test organization applicants, ensuring result equivalency, and organizing the annual BACnet International PlugFest[3], an in-person BACnet interoperability workshop. Through these efforts, the BTL-WG aims to enhance real-world interoperability and foster continuous improvement in BACnet product testing and certification (BACnet International, 2023).

3.2 Types of Devices

BACnet is a versatile data communication protocol designed for building automation and control systems. It defines standard methods that manufacturers can implement to create components and systems that can interoperate with other BACnet devices. The protocol provides a set of standardized

[2]https://www.big-eu.org/de/ (accessed 06 May 2023)
[3]https://bacnetplugfest.org/ (accessed 06 May 2023)

tools for building owners and system specifiers to specify and design interoperable systems, encompassing a wide range of automated building systems beyond HVAC applications. Devices that can communicate using BACnet include HVAC systems, such as thermostats, temperature sensors, dampers, valves, chillers, boilers, air handlers and VAV controllers. Additionally, lighting control systems, access control systems, fire and life safety systems, energy management systems and environmental sensors can also utilize BACnet for integration, centralized control, monitoring and energy optimization within buildings (Bushby & Newman, 2002).

A BACnet device typically consists of a microprocessor-based controller and software combination that understands and uses the BACnet protocol. It can serve as a controller, gateway, or user interface. Each BACnet device contains a device object that defines device information, including a unique device instance number, which must be field-configurable to ensure uniqueness across the BACnet network. The device instance, along with other information, control programs, logic, and data values, form the collection of device information and metadata within a BACnet device. BACnet addresses device interoperability by dividing it into three distinct areas: Objects (information), services (action requests) and transport systems (networking, messaging systems). The protocol defines methods and requirements for implementing each of these areas, facilitating the exchange of information over various networking systems by defined messages and performing actions to enable effective communication between devices (BACnet International, 2014). Some of the most popular device types namely are:

- Supervisors
- Routers
- Gateways
- Controllers
- Communicating Thermostats
- I/O Modules

Supervisors serve as powerful management tools, providing a centralized interface for monitoring and controlling various aspects of the building automation system. Routers enable seamless communication between different BACnet networks, ensuring efficient message routing and data exchange. Gateways play a crucial role in integrating non-BACnet devices into BACnet systems, allowing different protocols to coexist and interact harmoniously. Controllers provide flexible and programmable solutions for automation and control, enabling customized functionality tailored to specific building requirements. Communicating thermostats offer advanced HVAC control capabilities, allowing for precise temperature regulation and energy optimization. Lastly, I/O modules expand the number of input and output

points in the field, providing a scalable solution for extending the capabilities of the building automation system. These devices collectively contribute to the seamless integration, efficient operation and enhanced functionality of buildings that utilize BACnet protocols. BACnets generalized model of device operation, information description and protocol specification contribute to enhanced interoperability and centralized management, promoting improved efficiency, comfort and operational control within buildings.

3.3 Objects, Services, Networking

The interoperable approach, which BACnet is aiming for, is being realized through information objects. These objects encompass crucial device components and data collections, enabling sharing and accessibility among BACnet devices. They can represent both physical and virtual information, covering inputs, outputs, control algorithms, applications and calculations. The BACnet standard (ASHRAE, 1995) defines 54 object types, ensuring uniform implementation and clear interpretation across devices. Non-standard or proprietary objects can also be created, although their interoperability may vary. Objects possess properties that define their attributes, including a name and value. Properties convey information about objects and can be read or written by other BACnet devices. The properties divide into required and optional ones, based on the object type.

To facilitate communication, BACnet employs services that enable formal requests between devices. These services are categorized into object access, device management, alarms, events, file transfer and virtual terminal functionalities. Object access services allow reading, writing, creating and deleting of object data, while device management services handle tasks like device discovery, time synchronization and database operations. Alarm and event services manage alarms and state changes, while file transfer services encourage the transfer of trend data and programs. Virtual terminal services enable interaction between humans and machines through prompts and menus. Each service has a defined request and corresponding reply, with the necessary parameters for effective execution. Messages exchanged between devices are encoded using a consistent numeric code language, ensuring seamless communication. BACnet supports various transport and networking systems for conveying these encoded messages, providing flexibility in choosing cost-effective methods for specific applications. Despite different transport methods, the content of the coded messages remains uniform, guaranteeing interoperability.

3.3.1 Objects and Properties

BACnet revolutionizes conventional industry practices by introducing object-oriented terminology (Bushby & Newman, 2002). Instead of the vague and manufacturer-specific term *Points* to denote various inputs and outputs, BACnet defines standardized *Objects* with corresponding *Properties* that describe the object's characteristics and status. This enables seamless communication and control between BACnet devices on the network.

Objects

BACnet objects serve as the foundation for organizing and exchanging data related to inputs, outputs, software and calculations. They can take different forms such as single points, logical groups, program logic, schedules and historical data in the form of trends. These objects encompass both physical concepts, like thermostats and HVAC systems, as well as non-physical concepts, like software-based HVAC maintenance schedules. Every object includes properties that facilitate information exchange and command execution. These properties can be visualized in a tabular format, with the property names listed in one column and their corresponding values in another. The values can be either read-only or write-enabled, allowing for effective data manipulation and control within the BACnet framework (BACnet International, 2014).

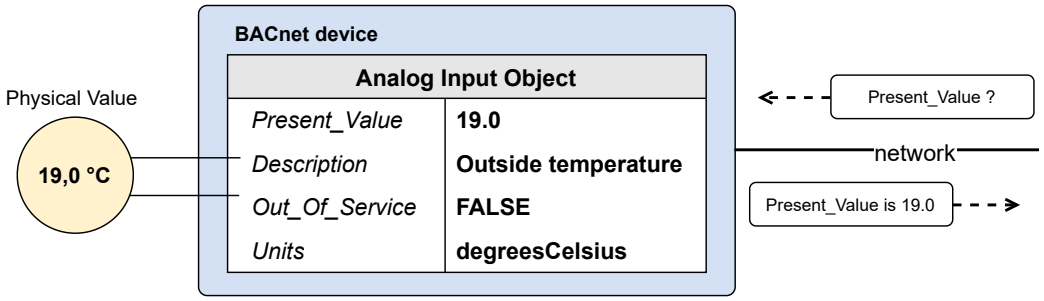

Figure 3.2: Example of a BACnet device that measures a physical value, which is further represented as a BACnet object

The Analog Input Object is a standard component in the BACnet framework, serving as a representation of an analog sensor input like a thermistor. Figure 3.2 shows a diagram displaying the Analog Input Object and its four properties when accessed through the network. Certain properties such as *Description* and *Units* are configured during installation, while others like *Present_Value* and *Out_Of_Service* provide information on the status of the sensor input associated with the Analog Input Object. Additional properties

may be defined by the equipment manufacturer. All properties can be read, and if a query about the *Present_Value* is executed, it would get the response *19.0*.

Table 3.1: Standard BACnet Objects with example use cases

Object	**Example use case**
Analog Input	Represents a sensor input, such as a temperature sensor
Analog Output	Controls an output, like adjusting the speed of a motor
Analog Value	Setpoint or analog control parameter, like target temperature for HVAC
Binary Input	Switch input, like a door open/closed
Binary Output	Relay output, like turning on/off a light
Binary Value	Binary control parameter e.g. auto/manual
Calendar	Defines a schedule with dates like holidays or events
Command	Executes a specific action across multiple devices, like switching from day mode to night mode
Device	Provides information about a device, like vendor, firmware and supported objects and services
Event Enrollment	Describes an event or alarm condition
File	Allows read and write access to data files supported
Group	Access multiple properties of multiple objects in a read single operation
Loop	Provides standardized access to a control loop
Multi-state Input	Represents the status of a multi-state process, e.g. status of a meeting room being empty, occupied, reserved
Multi-state Output	Set the desired state of a multiple-state process
Notification	List of devices to be informed if an Event Enrollment determines an event or alarm
Program	Controls the execution and status of a program
Schedule	Define a weekly schedule of operations with exceptions defined in the Calendar object

BACnet encompasses a comprehensive set of 18 standard objects, shown in Table 3.1, that represent various elements of a building control system (BACnet International, 2014). These objects range from analog inputs for sensors to schedules for scheduling and notification classes for alarms. The inclusion of specific objects in a BACnet device depends on its function and capabilities, with no requirement for all objects to be present in every scenario. For instance, a device controlling a Variable Air Volume (VAV) box will likely feature analog inputs and outputs, while a workstation providing a user interface without sensor inputs or control outputs will

not. Manufacturers can also introduce proprietary objects that are specific to their equipment, which coexist with standard BACnet objects without causing interference (Swan & Alerton Technologies Inc., 2022).

Properties

The BACnet standard (ASHRAE, 1995) defines 123 different properties for objects, with each type of object having a specific subset of these properties. Certain properties are required for each object, while others are optional. The specification outlines the behaviours of these implemented properties, particularly those related to alarm or event notifications and those affecting control values or states. Some properties are writeable according to the specification, while others may be at the manufacturer's discretion. All properties can be read over the network. Although BACnet allows vendors to include proprietary properties, these may not be understood or accessible by equipment from other manufacturers. While providing an exhaustive list of all properties and their requirement status would be beyond the scope of this section, it is worth noting that Table 3.2 shows the properties of an Analog Input Object, which was previously discussed in the example related to Figure 3.2.

Table 3.2: Overview of properties of an Analog Input Object

Property	**Required**	**Example**
Object_Identifier	Yes	analogInput_160
Object_Name	Yes	AI_160
Object_Type	Yes	anlogInput
Present_Value	Yes	19.0
Description	Optional	Outside temperature
Device_Type	Optional	Thermistor
Status_Flags	Yes	In_Alarm, Fault, Overriden, Out_Of_Service flags
Event_State	Yes	Normal
Reliability	Optional	No_Fault_Detected
Out_Of_Service	Yes	FALSE
Update_Interval	Optional	1.00 (seconds)
Units	Yes	degreesCelsius
Min_Pres_Value	Optional	-30.00, minimum reliably read value
Max_Pres_Value	Optional	+100.00, maximum reliably read value
Resolution	Optional	0.1
COV_Increment	Optional	Notify if Present_Value changes by increment: 0.5
Time_Delay	Optional	Seconds to wait before detecting out of range: 5
Notification_Class	Optional	Send Change-Of-Value (COV) notification to Notification Class Object: 2
High_Limit	Optional	+60.0, Upper normal range
Low_Limit	Optional	-20.0, Lower normal range
Deadband	Optional	0.1
Limit_Enable	Optional	High-limit-reporting, Low-limit-reporting
Event_Enable	Optional	To_Offnormal, To_Fault, To_Normal
Acked_Transitions	Optional	Flags indicating received achnowledgements for above changes
Notify_Type	Optional	Events or Alarms

3.3.2 Services

BACnet services are formal requests sent between BACnet devices to perform specific actions or exchange information. Each service request is encoded into numeric codes and transmitted as electronic messages across the network. BACnet allows flexibility by supporting multiple types of transport systems for message transmissions. These services are categorized into five

groups:

- Object access
- Device management
- Alarm and event
- File transfer
- Virtual terminal

Each service category includes confirmed and unconfirmed services, indicating whether a reply is expected or not. While BACnet devices are not required to implement all services, the *ReadProperty* service is mandatory for all devices, to access objects on a basic level. Additional services may be supported depending on the device's function and complexity. The alarm and event services handle changes in conditions and notifications of possible errors or alarms. Change-Of-Value (COV) reporting allows devices to notify subscribers about changes in properties without constant polling, reducing network workload. File access services enable the reading and manipulation of files within BACnet devices. Object access services facilitate reading, modifying and writing properties, as well as adding and deleting objects. Remote device management services provide various functions including operator control, specialized message transfer, and addressing capabilities. As an example, services such as *DeviceCommunicationControl* and *ReinitializeDevice* allow diagnostic tools to be invoked remotely, while *ConfirmedPrivateTransfer* and *UnconfirmedPrivateTransfer* enable the transmission of non-standard messages. *TimeSynchronization* ensures device clock synchronization, while *Who-Is* and *I-Am* services help obtain network addresses of BACnet devices. Similarly, *Who-Has* and *I-Have* services assist in discovering devices using object identifiers or object names.

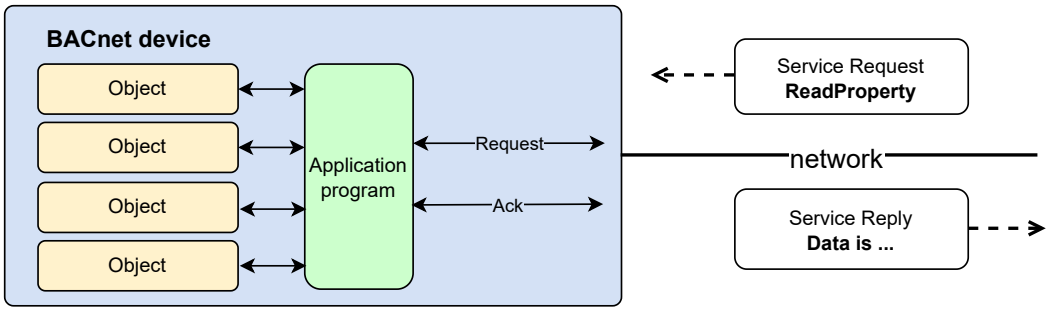

Figure 3.3: Example of a BACnet device requesting and acknowledging services over network

Services play a vital role in acquiring information, commanding actions or announcing events between BACnet devices. When a service request is issued, it becomes a message packet transferred from the sending to the

receiving device. An application program running on a BACnet device initiates service requests and processes them upon receipt, which is illustrated in Figure 3.3. For instance, an operator workstation may periodically request sensor data from target devices, while the target device processes the request and sends back the requested data. The provided Table 3.3 presents a comprehensive overview of some of the most frequently used services in BACnet. It categorizes the services into the two already mentioned types: Object Access and Device Management.

Table 3.3: Comprehensive overview of BACnet services

SERVICE	C/U	DESCRIPTION
Object Access Services		
AddListElement	C	Adds one or more items to a property that is a list.
RemoveListElement	C	Removes one or more items from a property that is a list.
ReadProperty	C	Returns a value of one property of one object.
ReadPropertyConditional	C	Returns the values of multiple properties in multiple objects.
ReadPropertyMultiple	C	Returns the values of multiple properties of multiple objects.
WriteProperty	C	Writes a value to one property of one object.
WritePropertyMultiple	C	Writes values to multiple properties of multiple objects.
Remote Device Management Services		
ConfirmedPrivateTransfer	C	Sends a vendor-proprietary message to a device.
UnconfirmedPrivateTransfer	U	Sends a vendor-proprietary message to one or more devices.
ReinitializeDevice	C	Order the receiving device to cold- or warm-boot itself.
ConfirmedTextMessage	C	Conveys a text message to another device.
UnconfirmedTextMessage	U	Sends a text message to one or more devices.
TimeSynchronization	U	Sends the current time to one or more devices.
Who-Has	U	Asks which BACnet devices contain a particular Object.
I-Have	U	Affirmative response to Who-Has, broadcast.
Who-Is	U	Asks about the presence of specified BACnet devices.
I-Am	U	Affirmative response to Who-Is, broadcast.

Note: The second column displays if it is a Confirmed or Unconfirmed service request, which indicates whether a reply is expected or not.

3.3.3 Transport and Networking Systems

Transport or networking systems use various electronic messaging standards and methods to transmit encoded messages. Despite using different transport methods, the content of the coded messages remains the same. This approach allows to select the most cost-effective transport method for particular applications. The 2012 BACnet standard defines seven network types that serve as the means of transporting BACnet messages:

- **BACnet/IP**
- BACnet MS/TP (Master-Slave/Toke Passing)
- BACnet ISO 8802-3 (Ethernet)
- BACnet over ARCNET
- BACnet Point-to-Point (EIA-232 and Telephone)
- BACnet over LonTalk
- BACnet over ZigBee

These network types encompass the physical and datalink layers of the protocol, collectively known as the Medium Access Control (MAC) layer. Regardless of the MAC layer used for transport, a BACnet message itself remains independent of it. Therefore, the commands and monitoring information conveyed through BACnet messages are the same. BACnet also offers a solution to connect multiple network types. A BACnet router, as shown in Figure 3.4 acts as a bridge between different network types. These routers can either be standalone devices or built into automation controllers, to allow BACnet messages to pass through different network types without altering their content.

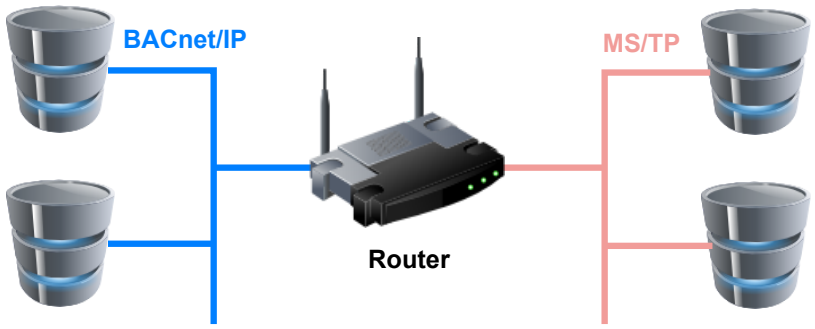

Figure 3.4: Example of a BACnet router to connect various network types

BACnet implements a collapsed architecture that corresponds to four layers of the Open Systems Interconnection (OSI) model: Application, Network, Data Link, and Physical. A detailed comparison is visualized in Figure 3.5. Unlike connection-based protocols, BACnet, being a non-connection protocol, requires less message segmentation and end-to-end error checking.

Consequently, the need for a distinct transport layer is eliminated, as its functionalities are integrated into the application layer. Additionally, the session layer is not utilized, and BACnet adopts a fixed encoding scheme while delegating security responsibilities to the application layer, which eliminates the necessity for a separate presentation layer.

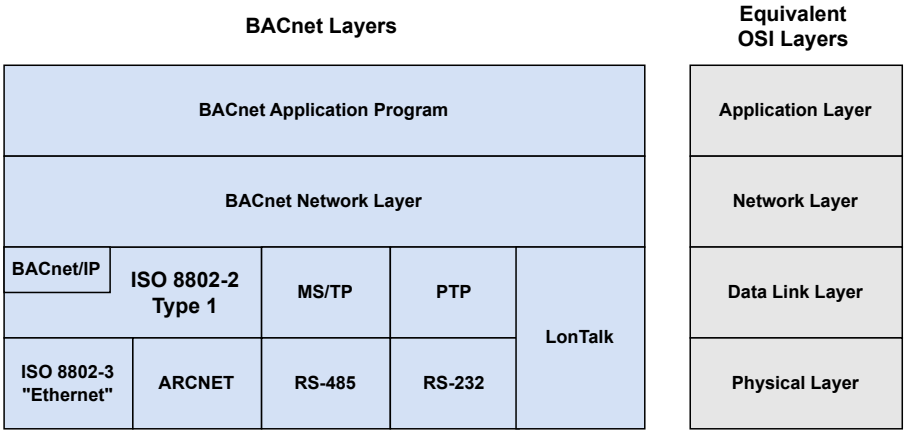

Figure 3.5: BACnet network layers compared to equivalent layers of the OSI model

BACnet over IP (Bushby & Newman, 2002) is the recommended choice for building automation and control networks due to its numerous advantages. It offers better performance and higher bandwidth compared to other protocols. With BACnet over IP, devices utilize Ethernet-based communication, which allows for fast and efficient data transfer. Most BACnet/IP devices support 100-megabit Ethernet, enabling data transmission at high speeds. Scalability is another key benefit of BACnet/IP. It can accommodate a large number of devices on the network without the need for tokens or waiting periods to initiate messages (Mills, 2019). Additionally, it allows for the seamless integration of numerous devices, even across multiple networks via the Internet. This enables the expansion and growth of building automation systems without the limitations imposed by other protocols. Security is a critical concern in any networked system, and BACnet/IP offers robust protection against potential breaches of sensitive data. This is particularly important for businesses and organizations that handle confidential information or operate in regulated industries. It ensures the integrity and privacy of data, providing peace of mind for users. Furthermore, issues and faults can be identified and resolved more efficiently through the use of IP addresses as unique identifiers. This facilitates quick and targeted troubleshooting, minimizing downtime and optimizing system performance (Erturk, 2021; Mills, 2019).

3.4 BACnet in Software Development

In the realm of BACnet communication, this section explores three significant topics that provide software developers with various options and tools. These topics include a comprehensive pre-compiled BACnet toolset, as well as BACpypes and BACo for Python development. Each of these offerings serves as a valuable resource for integrating BACnet functionality into projects efficiently through a ready-to-use toolset, a comprehensive Python library, or a lightweight and user-friendly API approach. These topics collectively contribute to the advancement and ease of implementing BACnet communication in software development projects.

3.4.1 BACnet Stack: Comprehensive Open-Source Tool Set

The described library is a comprehensive implementation of the BACnet protocol stack, which provides communication services for building automation and control networks. It is developed and maintained by Steve Karg, an experienced BACnet product developer, who also offers consulting services in BACnet product development. The BACnet protocol is an open standard for data communication in building automation and control networks. It allows devices to send and receive messages containing data that can be understood by other BACnet-compliant devices. The BACnet library provided by Steve Karg includes the application layer, network layer, and media access (MAC) layer communications services. It is specifically designed for embedded BACnet appliances and is written in C for portability. The code can be compiled with GCC as well as other compilers like Clang or IAR. The library is released under a GPL with an exception license, which means that any changes to the core code that are distributed must be shared, but the BACnet library can be linked to proprietary code without the proprietary code becoming GPL. Some of the source files are provided as skeleton, example, or template files and are not copyrighted as GPL. The BACnet protocol is an ASHRAE/ANSI/ISO standard, and this library adheres to that standard. It allows software developers to develop BACnet-compliant devices and integrate them into building automation and control networks. The library is freely available, and is hosted on Sourceforge[4] and Github[5].

[4]https://sourceforge.net/projects/bacnet/ (accessed 10 January 2023)
[5]https://github.com/bacnet-stack/bacnet-stack (accessed 09 January 2023)

Docker Multi-Stage Build for BACnet Stack

Setting up a Docker container is a valuable approach when dealing with cross-compatibility and library dependency issues in software development. Docker provides a containerization platform that allows applications to be bundled with their dependencies and run consistently across different environments. This solves the challenge of ensuring that an application functions correctly regardless of variations in underlying libraries or dependencies. In the Dockerfile provided in Listing 3.1, a Docker container is being created using a multi-stage build approach, which further enhances the portability and efficiency of the containerization process.

The Dockerfile starts by utilizing a Debian Bullseye base image to create an image named `compile-image`. It installs necessary build tools, downloads the BACnet stack source code from a specific Github URL, and extracts it into a directory called `bacnet-stack`. Then, it moves into the `bacnet-stack` directory and compiles the BACnet stack using the `make` command. After successful compilation, unnecessary files in the `bin` directory are removed. This stage serves as an intermediate image solely for building purposes. In the subsequent phase, a new image named `build-image` is created based on Python 3.11. The previously compiled BACnet stack binaries are copied from the `compile-image` stage into the `/usr/local/bin` directory. Additionally, the Dockerfile copies the application's source code from the local `./src` directory into `/src` within the image. As one intermediary step, the compiled binaries are also copied into the `/compiled_tools` directory. In combination with the docker-compose file of Listing 3.2, the tools will be available also outside of the container, making it an efficient and dependency-independent compilation image.

```
1  FROM debian:bullseye AS compile-image
2
3  RUN apt-get update && apt-get -y install build-essential
4
5  ADD https://github.com/bacnet-stack/bacnet-stack/archive/
       refs/tags/bacnet-stack-1.0.0.tar.gz .
6
7  RUN  tar zxf bacnet-stack-1.0.0.tar.gz  \
8       && mv bacnet-stack-bacnet-stack-1.0.0 bacnet-stack
9
10 RUN cd bacnet-stack \
11          && make \
12          && rm -f bin/*.txt bin/*.bat
13
14
15 FROM python:3.11 AS build-image
16
```

```
17 COPY --from=compile-image /bacnet-stack/bin/* /usr/local/bin
     /
18 COPY --from=compile-image /bacnet-stack/bin/* /
     compiled_tools
19 COPY ./src /src
20
21 WORKDIR /app
22
23 COPY bacnet-wrapper .
24
25 CMD ["/bacnet-wrapper"]
```

Listing 3.1: Multi stage Docker build to compile the BACnet stack tools of S. Karg

The Dockerfile example utilizes a multi-stage build technique. This approach allows the construction of a container image in multiple stages, each with a distinct purpose. In this case, the first stage, `compile-image`, is responsible for building the BACnet stack and generating the necessary binaries. These binaries are then extracted and transferred to the second stage, `build-image`, which creates the final image for running the application. By separating the build and runtime environments, unnecessary dependencies and artefacts from the build stage are excluded, resulting in a smaller and more efficient final image. This approach enhances the portability, security, and performance of the Docker container.

Docker and Docker Compose work together to simplify the deployment and management of containerized applications. Docker provides a platform for creating and running containers, which are isolated environments that encapsulate an application and its dependencies. Docker Compose, on the other hand, is a tool that allows you to define and manage multi-container applications using a YAML file. It provides a way to orchestrate the deployment of multiple services and configure their interactions, networks, volumes, and other aspects (Docker Inc., 2023).

```
1  version: '3'
2  services:
3
4    server:
5      build:
6        context: .
7      volumes:
8        - ./compiled_tools:/compiled_tools
9      command: ["bacserv", "200001"]
10     network_mode: bridge
```

Listing 3.2: Docker Compose file for orchestrating the compilation service and map a container directory to a local one

The purpose of the provided Docker Compose file is to create an environment for generating the compiled binaries of the library. The file defines a service named `server` based on the corresponding Dockerfile of Listing 3.1. The service is responsible for building the necessary image and executing the required commands. Within the Docker Compose file, the `server` service is configured to build the image of the compilation process using the specified Dockerfile located in the current directory and start one of the tools called `bacserv`, which starts a BACnet server. To facilitate access to the compiled files, the volumes section is included in the Docker Compose file. It maps the `./compiled_tools` directory from the host machine to the `/compiled_tools` directory within the container. This enables the compiled binaries to be available both within the container and in the file directory of the host machine. This volume mapping ensures that the compiled files can be easily accessed and shared between the container and the local machine's file system.

Summarized, the Docker Compose file in combination with the Dockerfile creates an environment in which the library's binaries can be compiled. The resulting compiled files are made accessible within the container and also conveniently available in the host machine's file directory through volume mapping.

Drawbacks of BACnet Stack

Pre-compiled tools in software development often lack transparency and control, as developers have limited visibility into their internal workings and less control over their behaviour. This can impede troubleshooting efforts and necessitate reliance on third-party support or updates from the tool provider. Compatibility limitations are another concern with pre-compiled tools. When aiming for cross-platform compatibility, these tools may not seamlessly work across different operating systems or architectures. Ensuring compatibility might require additional efforts, such as platform-specific configurations or workarounds. However, when contained within a Dockerized environment as shown in the previous section, pre-compiled tools can provide cross-compatibility within the containerized ecosystem. This means that the tools can function consistently and predictably within the container environment, offering a stable and reproducible development environment.

To mitigate these limitations, developers should consider alternative approaches[6], such as building tools from source code or utilizing software development libraries that provide access to parameters and return values.

[6] https://bacnet.org/developer-aids/ (accessed 03 June 2023)

This opens up opportunities to delve deeper into the inner workings of the tool or library, enabling us to understand its functionality and make modifications as necessary. This level of transparency provides insights into the tool's behaviour, facilitating easier troubleshooting and bug fixing. As a developer, having access to software development libraries that provide comprehensive parameter and return value access allows one to fine-tune the behaviour of the tool and seamlessly integrate it into projects. This level of control fosters greater efficiency and adaptability in the development process.

3.4.2 BACpypes and BAC0: Libraries for Python

Python libraries for BACnet communication are widely used by developers to facilitate seamless integration and interaction with BACnet-based systems. Among these libraries, *BACpypes* and *BACo* stand out as popular choices for implementing BACnet functionality in Python applications. They provide powerful tools and frameworks to implement BACnet functionality in Python applications, enabling efficient control and monitoring of building automation and control networks. With a wide range of features and functionalities, these libraries can facilitate the streamlining of the development process and can help build robust BACnet applications with ease. Whether it's for commercial or industrial purposes, Python libraries for BACnet communication offer a reliable foundation for achieving interoperability and maximizing the potential of BACnet technology.

BACpypes by Joel Bender

BACpypes is a widely-used open-source library designed to enable BACnet functionality within Python applications developed by Joel Bender. It offers developers a comprehensive toolkit with various utilities and tools, simplifying communication with BACnet-based systems. BACpypes is highly regarded for its versatility, user-friendly nature, and extensive support for BACnet protocols and services, whereas its source code can be found on Github[7]. One of the standout features of BACpypes is its object-oriented approach to BACnet communication. It provides a rich collection of classes representing BACnet objects, including devices, objects, properties, and services. This allows developers to leverage these classes to create BACnet devices, simulate BACnet networks, and interact with BACnet objects using familiar Python programming concepts. BACpypes supports multiple transport options for BACnet communication, such as Ethernet, IP, and serial

[7]https://github.com/JoelBender/bacpypes (accessed 12 December 2022)

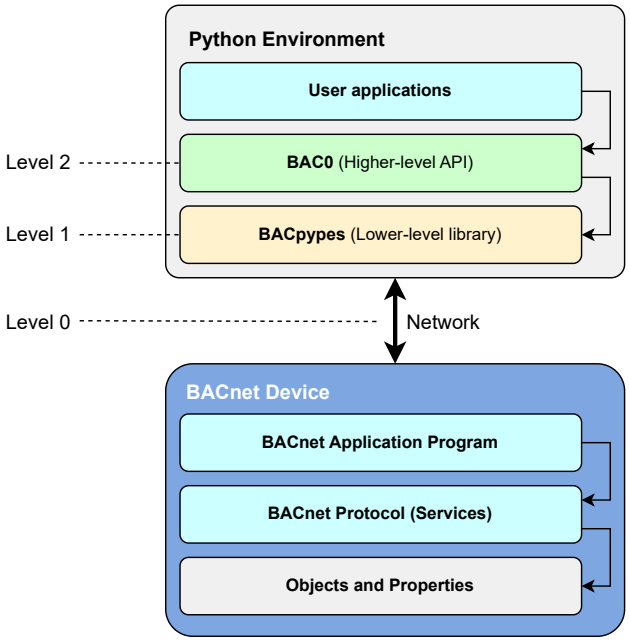

Figure 3.6: From BACnet Objects and Properties to Python-based user applications. Abstraction levels of common Python-based BACnet libraries

connections. This versatility enables developers to establish communication with BACnet devices across diverse networks. Additionally, BACpypes simplifies the handling of BACnet messages by offering convenient features for encoding, decoding, and exchanging BACnet packets. However, due to its rich feature set and object-oriented approach, BACpypes can be more complex to use and may require a deeper understanding of BACnet protocols and concepts.

BAC0 by Christian Tremblay

In response to this complexity, BAC0 was built upon the foundation of BACpypes. BAC0 is developed by Christian Tremblay and its source code is also available on Github[8]. The project gets good reputations and also the main developer of the project is very responsive in answering issues. With its lightweight and flexible approach, BAC0 offers simplicity and ease of use, making it an ideal choice for developers seeking a more streamlined BACnet solution. Its minimalistic design provides a straightforward API that abstracts away the complexities of the BACpypes communication, allowing developers to efficiently interact with BACnet objects and services. Supporting various BACnet transports, such as Ethernet and IP, BAC0

[8]https://github.com/ChristianTremblay/BAC0/ (accessed 04 December 2022)

enables communication with BACnet devices over different networks. It simplifies tasks like device discovery, property reading and writing, and executing BACnet services. BAC0 also includes subscription-based monitoring, providing real-time updates and notifications from BACnet devices by automatically subscribing to BACnet's Change-Of-Value (COV) services.

BACpypes vs. BAC0

BAC0 and BACpypes differ in their design philosophy and complexity, with BACpypes offering a comprehensive feature set that can be more complex to use, while BAC0 focuses on simplicity and ease of use with a minimalistic API. With BAC0 being built upon the BACpypes library, it also provides a higher level of abstraction. Figure 3.6 illustrates these abstraction levels, where BACpypes form the foundation providing low-level BACnet functionality to interact with devices, objects, properties and services. BAC0, as the higher-level API, abstracts away much of the complexity and exposes simplified interfaces for interacting with BACnet devices, making it easier for developers to integrate BACnet functionality into their Python applications. In the subsequent chapter, we will delve into the selection of BAC0 as the preferred library for implementing BACnet communication, exploring its key features, advantages and practical examples to demonstrate its effectiveness in real-world scenarios.

4 Brick

Brick is an open-source project that aims to establish a unified framework for describing the physical, logical and virtual components of buildings, as well as their interconnections in the form of relationships. It consists of a flexible data model, an expandable dictionary of terms and a system of relationships to connect and combine concepts. Leveraging Semantic Web technology, Brick enables the consistent representation of diverse features, assets and subsystems across the building sector (Brick Ontology, 2023). Brick is a popular ontology choice in the building sector, surpassing other ontologies by outscoring them in comparative studies as described in the Sections 2.4.2 Ontologies for the Building Sector and 2.6 Related Work. In addition, Brick has released a comparative study on their website[1], where they discussed the modelling support of building specific aspects of Brick and four other ontology models namely: Project Haystack (Project Haystack, 2023), SAREF (SAREF, 2023), IFC (buildingSMART International, 2023) and BOT (Linked Building Data Community Group, 2021). Table 4.1 shows this comparison.

Table 4.1: Comparative overview of ontologies related to building aspects

Modeling Support	**Brick**	**Haystack**	**IFC**	**BOT**	**SAREF**
HVAC Systems	**yes**	**yes**	**yes**	no	no
Lighting Systems	**yes**	partial	**yes**	no	no
Electrical Systems	**yes**	**yes**	**yes**	no	no
Spatial Information	**yes**	no	**yes**	**yes**	no
Sensor Systems	**yes**	**yes**	generic	no	**yes**
Control Relationships	**yes**	no	generic	no	no
Operational Relationships	**yes**	no	generic	no	no
Formal Definitions	**yes**	no	**yes**	**yes**	**yes**

[1] https://docs.brickschema.org/intro.html (accessed 28 May 2023)

4.1 Core Concepts

Entities, tags, classes, and graphs are fundamental components of Brick, forming the core concepts that enable standardized semantic descriptions. The interconnections between components are visualized as relationships within the framework. Alongside the core concepts, Brick follows a set of design principles which include completeness, expressiveness, usability, consistency and extensibility. These principles ensure that the schema can handle all the necessary information for building applications, captures diverse entities and their relationships, remains user-friendly and comprehensible, promotes consistency in modelling processes and allows for seamless expansion to cover new concepts in a consistent manner (Brick Ontology, 2023).

4.1.1 Tags

In Brick, tags serve as atomic facts or attributes associated with entities, providing additional descriptive information. They can include labels like sensor, setpoint, air, water and more. Brick adopts the concept of tags from Project Haystack to maintain annotation flexibility and usability. However, the entity classification of Brick goes beyond tags alone. It incorporates a comprehensive ontology, defining a wide range of classes and relationships to accurately represent building systems. The combination of tags and ontological definitions builds a robust framework for modelling building systems. Tags that are associated with Brick classes will be inherited by any Brick entity which is an instance of that class.

4.1.2 Classes

In the context of Brick, a class refers to a named category that carries a specific definition, which serves the purpose of grouping entities based on their shared characteristics or features. These classes are hierarchically structured, forming a taxonomy[2]. Entities are considered instances of one or more classes. An entity's type is determined by the class(es) it belongs to. Additionally, classes are associated with a set of tags, which serve as informative annotations. The root classes defined by Brick Ontology (2023) are:

- *Equipment*

[2]https://brickschema.org/ontology/1.1/#Classes (accessed 19 May 2023)

- *Location*
- *Point*
- *Tag*
- *Measurable* (containing the *Substance* and *Quantity* classes)

4.1.3 Entities

An entity in Brick serves as an abstract representation of any physical, logical or virtual item present in a building. Physical entities are tangible elements like mechanical equipment, lighting systems, networked devices and spatial elements such as rooms and floors. Virtual entities are software-based representations, including sensing and status points that provide real-time information, as well as actuation points that enable software-controlled adjustments. Logical entities refer to entities or groups defined by specific rules, such as HVA and Lighting zones and also include class names and tags within their scope.

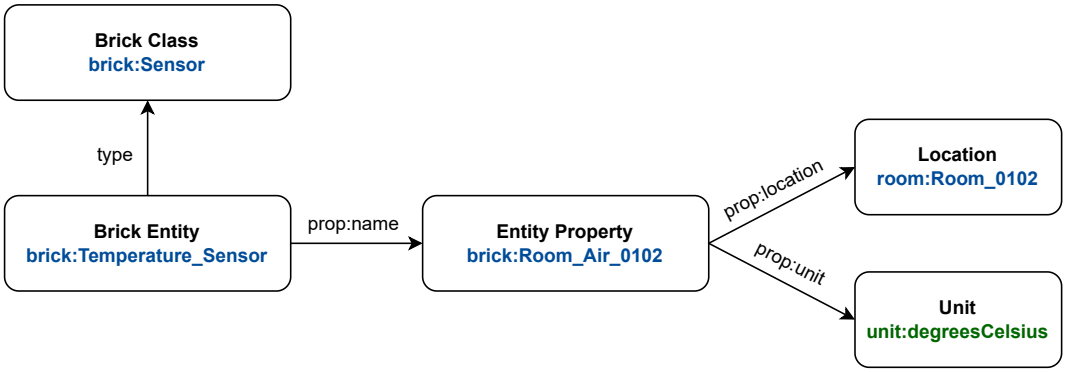

Figure 4.1: Example of a Brick temperature sensor entity, described by properties (Brick Ontology, 2023)

Entity properties within Brick represent the attributes or characteristics of entities. These properties have infrequent or static changes, while regularly changing values are better suited for modelling as Brick Points. Entity properties are useful for capturing static features like floor area, room volume, electric meter phases and more. An Entity Property consists of two components: the named relationship between an entity and the property value and the property value itself, which can e represented as a separate entity with its own set of properties. Figure 4.1 shows an example of a Brick Temperature Sensor. The type of entity is derived from a Brick class called `Sensor`. The *name* property identifies the entity whereas *location* and *unit* describe static features. The definition and specification of Entities and related Properties along with their classes and relationships are defined within the brick namespace (Brick Ontology, 2023).

4.1.4 Graphs

The representation of Brick itself is based on a directed, labelled graph shown in Figure 4.2. This is a conceptual structure that organizes data by representing entities as nodes and their relationships as edges. In Brick, the graph is specifically represented using the Resource Description Framework (W3C, 2023) data model. RDF provides a standardized way to express and link information in the form of subject-predicate-object triples. Subjects and objects refer to entities and properties represented by nodes whereas the connecting predicate describes the type of relationship between them visualized as edges, which is illustrated in Figure 4.3. Brick benefits from the RDF framework through its serialization syntaxes for storing and exchanging RDF-defined data models as known file formats such as Turtle and JSON.

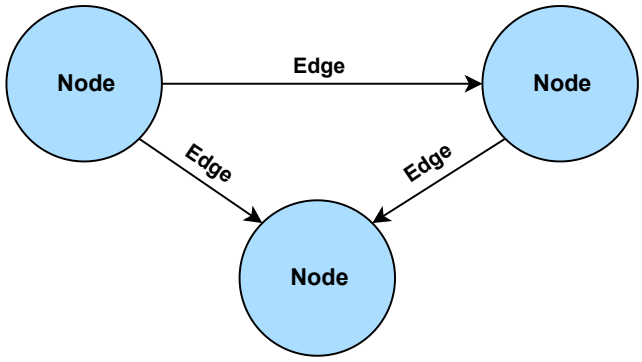

Figure 4.2: Directed labelled graph

4.1.5 Relationships

The description of a building can be approached from various perspectives and Brick defines relationships that cover several of these aspects. Firstly, there is composition, which outlines how different elements can be assembled to form others, such as equipment being composed of other equipment (e.g. a sensor being part of an air conditioning unit) or locations being composed of other locations (e.g. rooms building a floor). There exist also logical compositions, like an HVAC zone consisting of a specific set of rooms. Secondly, there is topology, which describes the connections and arrangements of elements, including how equipment is interconnected and the order in which they affect media flowing through a building, like air or water. Topological descriptions also describe relations between spaces or zones. Lastly, there is telemetry, which focuses on the data sources associated with various elements, whether logical, physical or virtual. These sources are referred to as *Points* in building management, which include sensors,

setpoints, commands, alarms and parameters that generate data for the building that is not considered static (Brick Ontology, 2023).

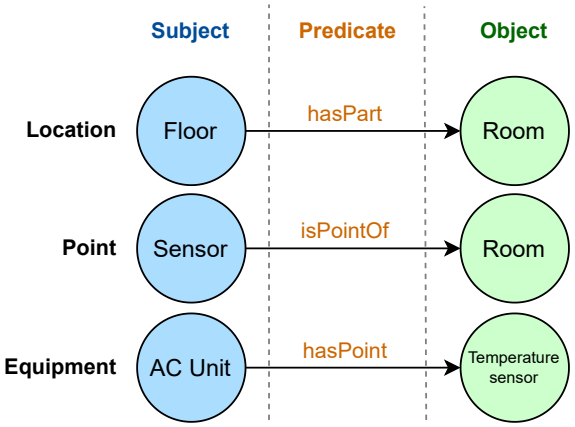

Figure 4.3: Brick relationship examples for location, point and equipment drawn as subject-predicate-object triples

When dealing with different subjects such as locations, points and equipment, there are specific relationships that can be used to describe their attributes. Locations use relationships such as *hasPart* to define components of that locations. The *isPointOf* relationship indicates relevant data of a specific point, for example, a sensor being the point of data collection for a room. In terms of equipment, relations like *hasLocation* denote physical locations of equipment or *feeds* are used to describe the downstream equipment and locations, for instance, an air handling unit feeding an HVAC zone. Figure 4.3 illustrates some of these relations as directed graphs. To increase flexibility for the modeller, Brick describes most of its relationships in a bidirectional way, by defining *inverse* relationships as shown in Table 4.2. This results in `subject-relation-object` being equivalent to `object-inverse-subject`.

Table 4.2: Examples of relationships and their respective inverse

Relationship	**Inverse**
`hasPoint`	`isPointOf`
`hasPart`	`isPartOf`
`hasLocation`	`isLocationOf`
`hasTag`	`isTagOf`
`feeds`	`isFedBy`

4.2 Data Source Models

Brick models provide a comprehensive representation of data sources and their associated characteristics. However, since Brick models are solely focused on describing the semantics of data sources, it is also necessary to have a method for representing the actual data they generate. Usually, data in the context of buildings refer to a collection of data points captured over time at regular intervals - time series. Time series data is used to represent the behaviour and performance of various building systems, such as temperature, humidity, energy consumption and other relevant metrics. This form of data is primarily used for analyzing trends, detecting anomalies, optimizing operations or making informed decisions (InfluxData Inc., 2023). Brick uses an external reference model for connecting data sources with their associated entities.

4.2.1 External References

Instances belonging to the Brick Point class serve as the origin or destination of telemetry data. Within Brick, there are several primary classes of telemetry including:

- *Sensor*: measured value from a device or instrument
- *Setpoint*: target values
- *Alarm*: notification about conditions or situations requiring corrective action
- *Command*: settings or actions influencing the behaviour of equipment
- *Parameter*: configuration settings (e.g. constraints on valid setpoint values)
- *Status*: observational data typically read-only

Each of these classes forms the root of a hierarchy consisting of more specific point types. As an example, a hierarchy of a sensor point could look like: `Point - Sensor - Temperature Sensor - Water Temperature Sensor`. Like all entities in Brick models, Points can be associated with metadata that specifies the storage location and identification of time series data. This metadata is linked to entity instances, representing data sources through the *ref:hasExternalReference* property. This property allows for a connection between a Brick entity and an external data source. More precisely for time series data exists the *ref:TimeseriesReference* object, which contains all necessary information to connect the metadata schema to an actual database. This includes information about unique identifiers or primary keys, as well as information about the database itself. Listing 4.1 shows an example of an air temperature sensor, which is linked to a time series database entry.

```
:sensor_1 a brick:Air_Temperature_Sensor ;
    brick:hasUnit unit:DEG_C ;
    ref:hasExternalReference [
        ref:TimeseriesReference
        ref:hasTimeseriesId "8F437C"
        ref:storedAt :database_3 ;
    ]

:database_3 a ref:Database ;
    brick:hasDescription "Timeseries storage"
    :connstring "http://1.2.3.4/data_3"
```

Listing 4.1: Example of a Brick entity linked with time series storage information as external reference

BACnet

One of Brick's features is the ability to model connections between Points and their counterparts in external systems. The *ref:hasExternalReference* property, is a generic property to define external references and also offers support for BACnet networks. The object-oriented approach of the BACnet communication protocol works well with the Brick metadata schema. A BACnet object is represented through the *ref:BACnetReference* object, which is defined as an external reference. By extending the example of Listing 4.1, the Brick model can link a temperature sensor to its time series storage and also model the reference to the existing BACnet network. Building such a complete semantic description makes it easier to access and analyze historical time series data, while also enabling real-time monitoring and control of building components through the BACnet object. Listing 4.2 illustrates the connection of a Brick Point and a BACnet Object through external references.

```
:sensor_1 a brick:Air_Temperature_Sensor ;
    brick:hasUnit unit:DEG_C ;
    ref:hasExternalReference [
        a ref:BACnetReference ;
        bacnet:objectIdentifier "analogInput_231" ;
        bacnet:objectName "BLDG-AI-231"
        bacnet:objectOf :device_7
    ]

:device_7 a bacnet:BACnetDevice ;
    bacnet:device-instance 7 ;
    bacnet:hasPort [ a bacnet:Port 47808]
```

Listing 4.2: Example of a Brick entity linked to a BACnetReference object

5 Study Design

After collecting knowledge about the domains of BACnet and Brick, this section will describe the design of the implementation as well as the practical experiment conducted within this study. Besides showcasing the practical part of the study, another major goal is to generate measurable results. To fulfil these requirements, the implementation followed a specific timeline to test and deploy the software script in both, a virtual and real-world environment.

5.1 Implementation Design

The implementation process was initiated with the collection of valuable insights in a simulation setup, where BACnet devices were virtually simulated. This stage involved simulating various interactions to understand the behaviour and responses of a BACnet system. Following the simulation phase, a develop and deploy setup was established in a real-world building to test the software script in practical conditions. The initial step of the script involved capturing a snapshot of the entire building's systems. This comprehensive snapshot included information about sensors, actuators, control points and other relevant data present in the infrastructure of the building. Additionally, this snapshot served as the foundation for the subsequent stages of the implementation, enabling the generation of a Brick ontology model.

By utilizing the system snapshot, the Brick ontology model was generated. The model aims to represent the building's infrastructure and their interrelationships in a standardized and structured format. By mapping the captured information to the Brick ontology, a unified and interoperable representation of the components of the building was established. The ontology model provides a powerful tool for further analysis, data integration and development of applications within the building ecosystem.

5.2 Non-Residential Building as Testbed

As a testbed for practical experiments, hardware and software is being deployed in a non-residential building, which is an elementary school in Graz. The building houses twelve classrooms and embodies a modern and innovative architectural approach, designed by an award-winning architect. The building incorporates a green roof with a photovoltaic system for power generation, storing rainwater and promoting thermal balance. Heating and cooling are provided by a ground-source heat pump connected to geothermal probes in the school garden. The system utilizes underfloor heating, ceiling radiant panels, and a central ventilation system for fresh air with heat and humidity recovery. In summer, a free cooling operation regenerates the geothermal probes, while windows can be opened for natural ventilation. The future plans are to extend the school by another building, housing another twelve classrooms. The building is controlled and automated by a BACnet-based building automation and control network. Some facts about the building in general:

- *Building floor area:* 4,256 m²
- *Heating and cooling requirements* according to the energy certificate: 12.12 kWh/m²a and 30.63 kWh/m²a, respectively
- *Photovoltaic system:* 25 kWp with 88 modules
- *Ground heat exchange field:* 9 double-pipe U-tube probes, approximately 100m in lengths
- *Heating:* 55 kW ground-source heat pump system, covering approximately 36% of the heating energy demand and about 43% of the heating and ventilation energy demand
- *Cooling:* 17.4 MWh/a of free cooling (direct cooling) with a maximum capacity of 60 kW and a continuous output of 30 kW

5.3 Experiment Design

The main goal of the experiment is to compare the simulation setup with the real-world setup. The software script was adapted to measure the time consumption associated with taking the system snapshot and generating the corresponding Brick ontology model. By tracking the duration of these processes, efficiency and performance measures of the implemented methodology have been done. To achieve this, a total of nine virtual devices were generated, each with a random number of BACnet objects. Each of the created virtual test devices got assigned a random number of different BACnet objects. For the sake of completeness, each device represents analog,

binary and multistate objects of input, output and value types. Four of the nine devices are defined as smaller devices, which could be represented as workstations. The other five devices are defined with a high number of BACnet objects, representing controllers. In comparison to the real devices, the simulated devices have an increased number of objects. A complete overview of all virtual generated devices is listed in Table 5.1. This approach ensures a representative sample of devices commonly found in building environments.

Table 5.1: Virtual test devices and their associated BACnet objects

Device	**BACnet objects**			**Sum**
	Analog	**Binary**	**Multistate**	
device_100	24	24	20	68
device_200	25	26	18	69
device_300	24	20	25	69
device_400	26	23	22	71
device_500	885	1352	1052	3289
device_600	1065	779	686	2530
device_700	1124	998	1271	3393
device_800	1068	1370	1011	3449
device_900	980	1026	976	2982
				15920

Based on the snapshot of the real-world setup, Table 5.2 shows an overview of all scanned devices from the non-residential building. This overview was taken from the system snapshot. The simulated setup consists of more BACnet objects than the real devices to exploit a more exhaustive approach when simulating.

Table 5.2: Devices of building and their associated BACnet objects

Device	**BACnet objects**			**Sum**
	Analog	**Binary**	**Multistate**	
WDXXZ_BC01	342	304	97	743
Smart_W100BC01	210	92	15	317
Smart_W100BC02	218	130	43	391
Workstation_1	0	0	0	0
Workstation_2	0	0	0	0
W100BC001	337	97	165	599
W100BC100	762	229	278	1269
W100BC200	1006	297	373	1676
W100BC300	497	147	179	823
				5818

Furthermore, to evaluate the coverage and accuracy of the Brick ontology model, a comparison was made between the number of setpoints abstracted within the ontology and the corresponding technical data sheets. This analysis provided valuable insights into the extent to which the model effectively captured and represented the real-world attributes and functionalities of the BACnet device. By undertaking this comparative analysis between the simulation setup and the real-world implementation, the experiment aims to express the strengths and limitations of the software script. Additionally simultaneously evaluating the same experiment in a virtual environment contributes to a better understanding of the simulation's reliability in capturing the complexity and intricacies of the physical building system. The evaluation of the virtual environment should show its potential as a cost-effective tool for testing and developing advanced building management solutions.

6 Implementation

In this chapter, we will discuss the development and deployment setup, the connection to an existing BACnet by using the BACo Python library, the serialization of scanned devices to a JSON file format and the generation of a Brick ontology model. The development and deployment setup is a crucial aspect, involving the configuration and integration of hardware and software components. BACo, a powerful library for BACnet communication, plays a key role in establishing connectivity with the existing BACnet network. It simplifies the interaction with BACnet devices by providing a high-level interface for reading and writing data points and controlling devices. To gain insights from the system while not being constantly connected to the BACnet network, the scanned devices are serialized to a JSON file format. This serialization process involves the connection to scanned devices and converting the device objects and property data into a structured format that can be easily stored, transferred and analyzed. By serializing the scanned devices, a snapshot of the BACnets system state at a specific point in time is captured. Examining the serialized data provides valuable insights for offline analysis and informed decision-making. This initial exploration of the serialized data serves as a foundational step towards the subsequent conversion to a Brick Ontology schema.

6.1 BACnet Simulation

This section investigates the capabilities that arise by combining the BACo library with a dockerized environment. The focus lies on simulating a locally hosted BACnet system, where the BACo library serves as a tool for emulating virtual BACnet devices, while the Docker containerization provides an isolated environment for each of the emulated devices. Additionally, Docker creates a segregated network environment that is independent of the host's original network. The provided network environment enables the handling of each device as if it were a physical hardware device, complete with its own unique IP address and network connection. The emulated devices within the simulation environment are configured using a JSON file (see Section 6.4.2) that serves as a dedicated configuration file for each device. This JSON

file comprehensively describes the objects and properties of the emulated device. Figure 6.1 gives an overview of the corresponding components for the simulation environment.

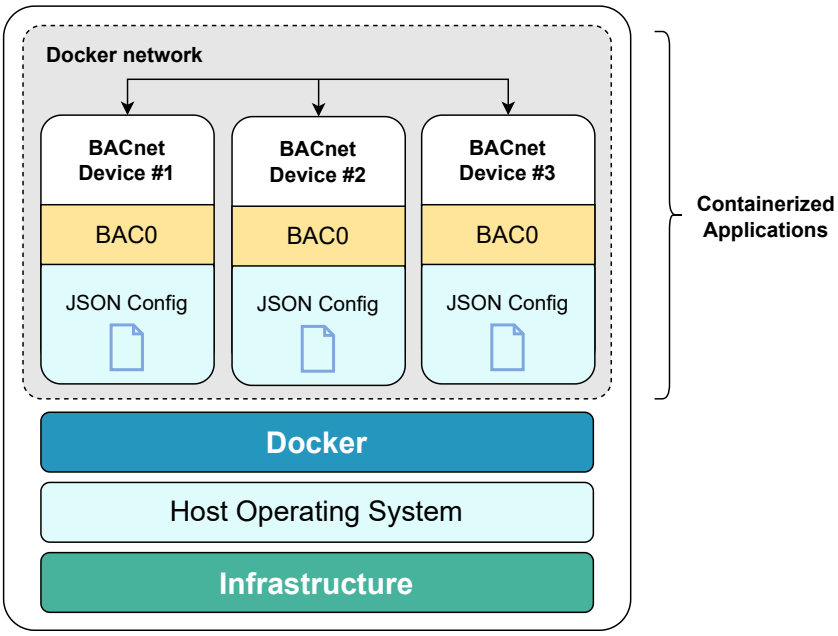

Figure 6.1: Overview of the dockerized simulation environment, to host a local BACnet emulating devices

Starting by providing a dockerized environment, Listing 6.1 shows the according Docker and Compose file. The base image is defined as Python 3.8-slim, which is a lightweight image providing basic Python functionalities. In order to make the BACo library available inside the Docker container, the requirements consisting of BACo and BACpypes need to be installed. Additionally the `netifaces` library needs to be installed too, which is used by BACo for handling IP addresses. The Compose file defines different services based on the provided Dockerfile. In this example, one device is defined as an `exampl_device` which starts the developed BACo script and passes the path to the config file, as well as the desired port number to the simulation service of the script. Each device can be defined as a service within the Compose file.

```
1 # Dockerfile
2 FROM python:3.8-slim
3
4 WORKDIR /app
5
6 COPY requirements.txt /tmp/requirements.txt
7 # Requirements are:
8 # bac0, bacpypes and netifaces
```

```
RUN pip install -r /tmp/requirements.txt

COPY /src/ /app/

ENV PYTHONPATH /app/src

# docker-compose.yml
version: '3'
services:

  example_device:
      build:
        context: .
      command: ["python", "-u", "bac0_tools.py", "simulate",
                "--device",
                "/path_to_config_file/example_device.json",
                "--port", "47808"]
      network_mode: bridge
```

Listing 6.1: Docker and Compose file for the dockerized simulation environment of a device

The script provided in Listing 6.2 starts by loading the device config file, and parsing it to a Python dictionary. It creates a BACo application for the device, to be able to use the BACnet protocol. The application object is additionally defined by the device properties, which get read from the config file. The function `add_properties_to_device` loads the analog, binary and multistate data points from the config file, and adds them as objects with appropriate properties to the simulated device. Finally a `BAC0.device` object is created, to connect to the simulated device. This final step is used to test the device creation and interact with the simulated device.

```
def simulate_bacnet_device(path_to_device_config,
                           device_port):

    device_dict = json.load(device_config_json)

    device_app = BAC0.lite(
        deviceId=device_dict[OBJECT_ID],
        localObjName=device_dict[OBJECT_NAME],
        description=device_dict[DESCRIPTION],
        vendorId=device_dict[VENDOR_ID],
        vendorName=device_dict[VENDOR_NAME],
        modelName=device_dict[MODEL_NAME])

    add_properties_to_device(device_dict, device_app)

    ip_address = device_app.localIPAddr.addrTuple[0]
    device_boid = device_app.Boid


```

```
20      created_device = BAC0.device("{}:{}"
21          .format(ip_address, BACNET_DEFAULT_PORT),
                device_boid)
```

Listing 6.2: Docker and Compose file for the dockerized simulation environment of a device

6.2 Development and Deployment Setup

In a setup aimed at enhancing communication and control within a commercial building, a Raspberry PI device running the Debian-based Linux OS, Raspbian GNU/Linux 10 Buster (Long, 2019), is utilized to establish a connection to the buildings Ethernet network, as shown in Figure 6.2. The commercial building uses BACnet/IP over the Ethernet network for communication and control of its automation systems and devices. By connecting the Raspberry PI to the Ethernet network, it becomes a central hub for bridging the gap between the BACnet devices and the network infrastructure. Raspbians's Linux foundation provides a reliable and secure platform for networking and is basically acting as a gateway to provide the necessary hardware for BACnet/IP communication.

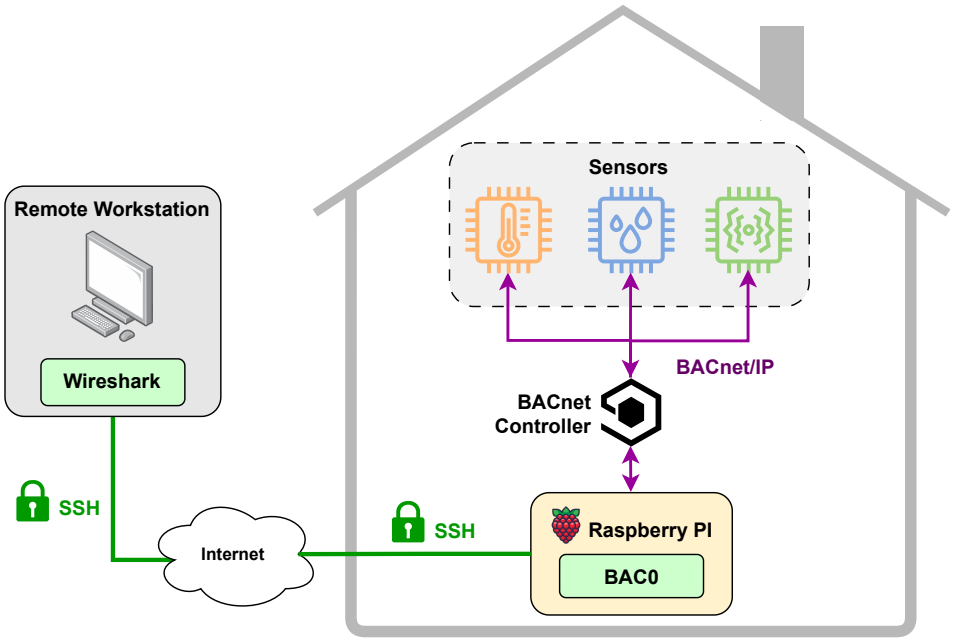

Figure 6.2: Development and deployment setup: Connecting a remote workstation over SSH to a Raspberry PI. The Raspberry is connected to the commercial buildings Ethernet where BACnet/IP is used as the protocol for the buildings automation system's communication

Due to the limitations of developing software directly at the commercial building's location, a Secure Shell (SSH) setup was established on the Raspberry Pi to enable remote access from a different location. SSH allows for secure and encrypted remote communication between two devices, enabling developers to connect to the Raspberry Pi over the network and execute commands, transfer files, and manage the device remotely. It provides a secure and convenient means of remotely accessing and controlling the Raspberry Pi's operations without physically being present at the commercial building (SSH, 2023). This involves setting up SSH key-based authentication, where a public-private key pair is generated. The public key is copied to the Raspberry Pi's `authorized_keys` file, while the private key is securely stored on the remote workstation. The SSH setup, illustrated in Figure 6.2, is used for two purposes: Firstly, to deploy software scripts onto the locally installed hardware of the Raspberry PI, allowing for remote development and updates. Secondly, to monitor on-site network traffic from a remote workstation, to gain insights about the BACnet traffic that is exchanged by devices.

6.2.1 Network Monitoring over SSH

To enable network traffic analysis in a commercial building, a setup is established involving the Raspberry PI (acting as an intermediate device) and a local PC, which is the remote workstation. The Raspberry PI, acting as the source device, requires the installation of *tcpdump* (Tcpdump Group, 2023), a powerful command line packet analyzer. It captures and displays network traffic, offering filtering options to selectively capture specific packets based on criteria such as sour or destination IP addresses, ports, protocols or packet types, allowing for precise analysis and monitoring of network data. This enables the Raspberry PI to capture and export network traffic data. On the other hand, the destination device, the remote workstation, requires the installation of graphical *Wireshark* (Wireshark Foundation, 2023), a user-friendly network protocol analyzer along with *mkfifo* (MKFIFO, 2001). The *mkfifo* command line utility, creates named pipes, providing a way to transmit data between processes, which in this example is used as an intermediate stage for storing the captured network data. This configuration enables the workstation to receive and analyze the network traffic captured by the Raspberry PI in real-time and has been already in use by Lee (2016). The goal is to execute *tcpdump* with filters over SSH on the Raspberry PI, redirect the output to a pipe file and subsequently visualize the captured network traffic using Wireshark. The following command creates a FIFO pipe file at the given location.

```
1 [Workstation] mkfifo /tmp/raspi_packet_capture
```

An intermediate step to allow a registered SSH user to execute a specific program without the need for a `sudo` password is to configure the `sudoers` file using `visudo`, adding an entry granting the user permission to run the program as root without requiring a password prompt over the SSH connection.

```
1 [Raspi] sudo visudo
2
3 # Grant sudo access to the tcpdump utility without password
     prompt by adding this line to the sudoers file
4 $USER ALL = NOPASSWD: NOPASSWD: /usr/sbin/tcpdump
```

To perform the necessary steps in Wireshark for network traffic analysis, Wireshark needs to be opened with sudo rights. In the Capturing options menu, a pipe file can be selected as part of the managed interfaces by Wireshark. In this section the local generated pipe file is added by selecting the file `/tmp/raspi_packet_capture`. This sets the local pipe file as data source of network traffic and Wireshark can start capturing. One important note here is to start capturing in Wireshark before executing *tcpdump* over SSH. The next step involves the execution of *tcpdump* over SSH on the Raspberry PI and setting appropriate filters to reduce network load. The following command is being executed at the destination device.

```
1 [Workstation] ssh raspi "sudo /usr/sbin/tcpdump -i eth0
     portrange 40000-50000" > /tmp/raspi_packet_capture
```

By specifying filters, the captured network traffic can be narrowed down to the desired scope. In the case of BACnet, a port range is added to cover common BACnet ports which were defined in a range from 40000 to 50000 based on Ethernet traffic. This filtering technique ensures that only relevant packets are captured, reducing the overall network load and providing a more efficient analysis of the BACnet communication within the commercial building. The output of the executed command is getting redirected to the local pipe file, which is already monitored by the Wireshark instance. Figure 6.3 shows a screenshot of the graphical interface of Wireshark, which visualized the data of the local pipe file.

6.3 BACnet Communication

This section explores the process of connecting to an existing BACnet system and discovering BACnet devices by utilizing the BACo API. By harnessing

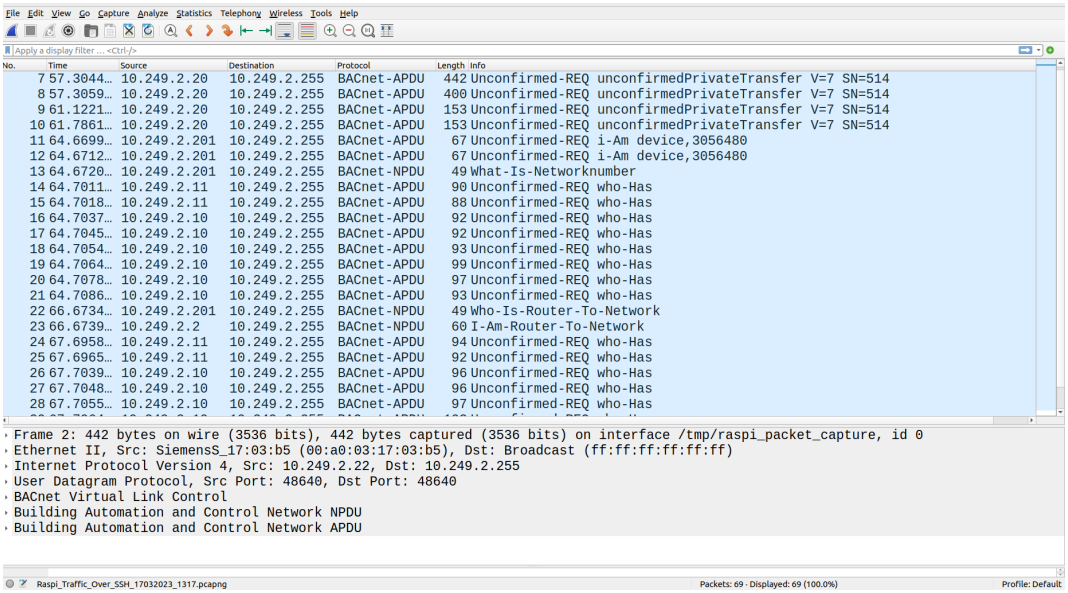

Figure 6.3: Visualizing the network traffic over SSH by using the graphical Wireshark interface

the capabilities of the BACo API, the process of connecting to an existing BACnet is simplified significantly. The necessary network calls for BACnet communication are wrapped up by the internally used BACpypes library. Therefore the API provides much simpler function calls, that trigger the desired network messages in the background. To gain a basic understanding of the core concepts, a shortened version of the script is provided in Listing 6.3. The script demonstrates the fundamental steps involved in connecting to a BACnet system and discovering BACnet devices using the BACo API.

```
1  import BAC0
2
3  def scan_for_devices():
4
5      port_number = 46000
6      bacnet_connection = BAC0.lite(port=port_number)
7
8      bacnet.whois()
9      discovered_devices = bacnet.devices
10
11     for device_local_name, device_ip, device_boid in
           discovered_devices:
12         temp_device = None
13
14         try:
15             temp_device = BAC0.device(device_ip,
16                                       device_boid,
17                                       bacnet,
18                                       poll=0)
```

```
19
20              # Do serialization and save JSON file to
                    directory ...
21          except:
22              print("Device connection could not be
                    established")
23
24          temp_device.disconnect()
```

Listing 6.3: Python script demonstrating the basic concepts of connecting and discovering to BACnet devices

At first, the script imports the BACo library to be accessible. The given code sets the log levels for the BACo API, specifying different levels of logging for different outputs. Afterwards, the specification of the BACnet connection takes place. The BACo API offers two modes for establishing BACnet connections: Lite and Complete. Lite mode is suitable for interacting with devices without utilizing the aspects of a graphical interface. It is designed for resource-constrained environments, such as small devices like a Raspberry PI. On the other hand, the Complete mode provides a graphical interface by launching a web server. The web interface shows and interacts with connected devices and features live trending features (Tremblay, 2020). In the context of this example, the utilization of the Lite version is sufficient, because of the unnecessity of a web interface and also the minimalistic hardware approach based on the SSH connection. After monitoring the network traffic, discussed in Section 6.2.1, it was evident that the devices of the building under investigation use port number 46000 to send BACnet/IP packets over the Ethernet connection. Therefore defining the port number and starting a lite connection to the building's BACnet worked out as a successful connection.

A `Who-Is` call serves the purpose of obtaining the network addresses of devices present on a network. This information is essential to enable devices to communicate with each other. When a device requires the address of another device, it sends a message that specifies either a specific Device Object Instance Number or a range of Instance Numbers. For instance, it may send a `Who-is device 3001` or `Who-is device 3000 to 3099` call. In the context of the script, it initiates a broadcast call to discover and retrieve the network addresses of BACnet devices within the network. The broadcasting character of this call makes it pretty heavy in terms of network load, especially if there are lots of devices. The acknowledgements of the present devices are stored in the `bacnet` objects and are accessed by `bacnet.devices`. This returns a list of the responding devices and their device information such as the device's address, which is mandatory to establish a connection.

As a final step, the retrieved device information is used to arrange a device

connection. Invoking the `BAC0.device(...)` call provides access to a specific BACnet device within a network. Because no live data is needed, the polling is set to 0, which disables constant polling of a device's properties. The returned `BAC0.device` object enables access to its BACnet application program which handles object types, properties and supported services. Additionally, it provides methods to manipulate the device data, such as reading or writing to property values and invoking BACnet services. This call serves as a tool for interacting with the individual BACnet devices, which is further needed to extract semantic data.

6.4 Serialization of Devices

In order to extract meaningful information from the system without the need for a constant connection to the BACnet network, the scanned devices undergo a serialization process that results in a JSON file format. Python's native support for JSON's key-value pair format simplifies data manipulation and analysis, as the JSON structure aligns with Python's dictionary object type. This compatibility is beneficial when it comes to reusing the data for further development. The familiarity with Python and the JSON file format eliminates the need for parsing or data handling routines. The serialization step involves establishing connections with the scanned devices and transforming the device objects and property data into a structured format that is easily storable, transferable and analyzable. Through this process, a snapshot of the BACnet system's state at a specific moment in time is effectively captured. Additionally, it serves as an initial exploration, laying the foundation for the subsequent conversion to a Brick Ontology schema.

6.4.1 Accessing Objects and Properties

The following code snippet, provided in Listing 6.4, is based upon the creation of the `BAC0.device` object, shown in Listing 6.3. The `BAC0.device` object is getting passed to the `serialize_to_json()` function.

```
1 def serialize_to_json(bacnet_device):
2   bacnet_properties = bacnet_device.bacnet_properties
3   property_dictionary = {
4     # Fetching device properties
5     OBJECT_TYPE: bacnet_properties[OBJECT_TYPE],
6     OBJECT_NAME: bacnet_properties[OBJECT_NAME],
7     DESCRIPTION: bacnet_properties[DESCRIPTION],
8     ...
```

```
9
10      # Looping over objects and appropriate properties
11      ANALOG_UNITS: get_analog_properties(bacnet_device),
12      BINARY_UNITS: get_binary_properties(bacnet_device),
13      MULTI_STATES:
14          get_multi_state_properties(bacnet_device)
15      }
16
17      object_name = str(bacnet_properties[OBJECT_NAME])
18          + '_' + str(bacnet_properties[OBJECT_ID][1])
19
20      return object_name, json.dumps(property_dictionary)
```

Listing 6.4: Reading device properties by invoking BACo object function calls

The purpose of this function is at first to store the properties of the `bacnet_device` into a dictionary object. The code begins by extracting the device object properties which have been discussed in Section 3.3.1. The different keys used from the dictionary, represent the required object properties. These properties need to be present on every BACnet device, ensuring that there are no access problems when trying to access the property values. The simplicity of the BACo calls abstract away the hidden complexity of composing the BACnet messages. Calls like `bacnet_device.bacnet_properies` or accesses like `bacnet_properties[DESCRIPTION]` are invoking the composition of the desired BACnet messages by sending according requests and handling the return messages. For instance, the access of the description property would form a `ReadProperty` service request to be sent to the connected device.

Since every BACnet object differs in terms of required properties, the decision was made to handle the physical descriptions for analog, binary, and multi-state data in different functions to treat them accordingly. These are the major fields describing the physical measures of a building. As an example, Listing 6.5 shows how the analog units of a BACnet device are being serialized.

```
1  def get_analog_object_properties(analog_bacnet_object):
2    bacnet_properties = analog_bacnet_object.bacnet_properties
3    object_id = bacnet_properties[OBJECT_ID][0]
4
5    return object_id, bacnet_properties, {
6      OBJECT_NAME: bacnet_properties[OBJECT_NAME],
7      ...
8      PRESENT_VALUE:
9          bacnet_properties[PRESENT_VALUE],
10     ENGINEERING_UNITS =
11         bacnet_properties[ENGINEERING_UNITS]
12   }
13
14 def get_analog_properties(bacnet_device):
```

```
15    analog_units = bacnet_device.analog_units
16    analog_properties = {}
17
18    for analog_unit in analog_units:
19      current_analog_object = bacnet_device[analog_unit]
20      obj_identifier, unit_properties =
21          get_analog_object_properties(current_analog_object)
22
23      analog_properties[obj_identifier] = unit_properties
24
25    return analog_properties
```

Listing 6.5: Reading analog object properties by invoking BACo object function calls

It is a pretty similar syntax to accessing device properties, but still, there are some specific actions about it. The `get_analog_object_properties()` function reads the `presentValue` and `engineeringUnits` properties from the device's object, by invoking the BACo API in the same manner as in Listing 6.4. Those properties are represented only by analog objects. The second part forms an overall dictionary about all analog units known to the device, which is then getting passed back to add to the device property dictionary. The same procedure is done also for binary and multi-state data, by considering their specific properties.

6.4.2 Serialization Results

As an illustration of a final result, Listing 6.6 shows the serialized data of the commercial building, captured by running the connection and serialization process on the Raspberry PI hardware.

```
1  {
2    'objectIdentifier': "device",
3    'objectType': "device",
4    'objectName': "W100XG001",
5    'description': "TRA Controller EG",
6    'location': "W100XGV01/EG",
7    'vendorIdentifier': 7,
8    'vendorName': "Vendor Building Technologies",
9    'modelName': "PXC3.E75-1",
10   'analogUnits': {
11     ...
12   },
13   'binaryStates': {
14     ...
15   },
16   'multiStates': {
17     ...
18   }
```

```
19 }
```

Listing 6.6: Serialized data of a BACnet device

The provided JSON object of Listing 6.6 represents serialized data of a BACnet controller device. It includes essential information such as the device's object identifier, type, description and vendor details. The blocks for `analogUnits`, `binaryStates` and `multiStates` are nested dictionary objects. These blocks contain information about the objects and their related properties. Listing 6.7 shows an excerpt of the serialized analog and multistate data points.

```
'analogInput_4': {
      'objectName': "W002GVS01/EG/R0_11S/SenDev/TR",
      'objectType': "analogInput",
      'description': "Room temperature",
      'statusFlags': [0, 0, 0, 0],
      'presentValue': 22.65999984741211,
      'lastTimestamp': "2023-04-18T12:28:56.655717+02:00",
      'units': "degreesCelsius"
    }, ...

'multiStateValue_22': {
      'objectName': "W002GVS01/EG/R0_09_1/RHvacCoo/RAInd",
      'objectType': "multiStateValue",
      'description': "Room air quality indication",
      'statusFlags': [0, 1, 0, 0],
      'presentValue': 1,
      'lastTimestamp': "2023-04-18T12:28:58.849615+02:00",
      'stateText': [
        "Undefined",
        "Poor",
        "Okay",
        "Good"]
    },
```

Listing 6.7: Serialized data of an analog and multistate object

6.5 Brick Ontology Generation

This section addresses the utilization of Brick in developing data-driven applications. Brick provides a collection of references that describe the interface between Brick and other APIs, databases, software libraries or digital representations. The fundamental purpose of a Brick model is to describe the data sources present in a building along with their contextual information. Data sources are represented by instances of the Brick Point

class, while the context is provided by instances of locations, equipment, systems and other entities within the building. Brick relationships establish associations between these instances and the corresponding data sources. The bidirectional approach in connecting instances grants flexibility in how software can interact and model a building and its data.

To enable data-driven applications and generate value, the creation of a Brick model is necessary. Depending on factors like the availability of metadata or digital representations of the building, Brick models can differ in the level of detail. There is no predefined methodology for building a Brick model, but common practice involves exploring data structures to represent collections of similar information, such as all rooms in a building. The main idea is to traverse these data structures and the creation of triples in a graph along the way (Fierro & Nagare, 2022).

6.5.1 BACnet-to-Brick Mapping

Understanding the relationships between BACnet objects and their representations in the Brick ontology model is important for modelling the Brick ontology graph. Therefore, an intermediate mapping process has been developed. This mapping process involves establishing a direct one-to-one correspondence, linking BACnet properties to their respective Brick entities within the ontology model. Figure 6.4 presents an illustrative flow diagram providing an overview of the mapping process. The blue marked fields are the BACnet objects of their respective device, which serve as a starting point for the mapping process. The final Brick entities are marked as yellow blocks. Some parts of the mapping process need to distinguish between the types of objects, especially if they are analog, binary or multi-state measures, which are represented in green. Each BACnet object will be defined as a unique entity within the Brick ontology. The labelling of the edges shows the corresponding relationship between the object id and the corresponding Brick attribute or entity.

Although not explicitly illustrated in Figure 6.4, distinguishing between analog sensors and setpoints is a particular case to consider. Analog measures have associated engineering units, which allow for more precise handling. Taking the engineering units into account, it is possible to achieve a higher degree of detail, by utilizing subclasses of the sensor and setpoint entities within the Brick model.

$$brickType_{Input} = \begin{cases} \texttt{Temperature\_Sensor} \; if \; unit = "degreesCelsius" \\ \texttt{Pressure\_Sensor} \; if \; unit = "pascals" \\ \dots \end{cases} \tag{6.1}$$

An approach to handling the mapping of engineering units from BACnet to their corresponding Brick entities involves the utilization of a lookup table. This lookup table acts as a reference that combines the various engineering units used in BACnet with their representations in the Brick ontology model. Equation 6.1 shows an excerpt from the lookup table and how internal decision-making is done. An analog input object in the context of BACnet is defined as a sensor responsible for measuring analog values. Additionally, when the specific engineering unit, such as *degreesCelsius* is known, a more detailed distinction can be achieved by classifying the analog input object as a "Temperature_Sensor". A similar approach has been applied to analog values, which are representing setpoints. To augment the mapping process, further information is integrated using a one-to-one relationship, achieved by introducing a BACnet reference as an external resource. This reference establishes a direct link between BACnet object properties and their corresponding Brick entities.

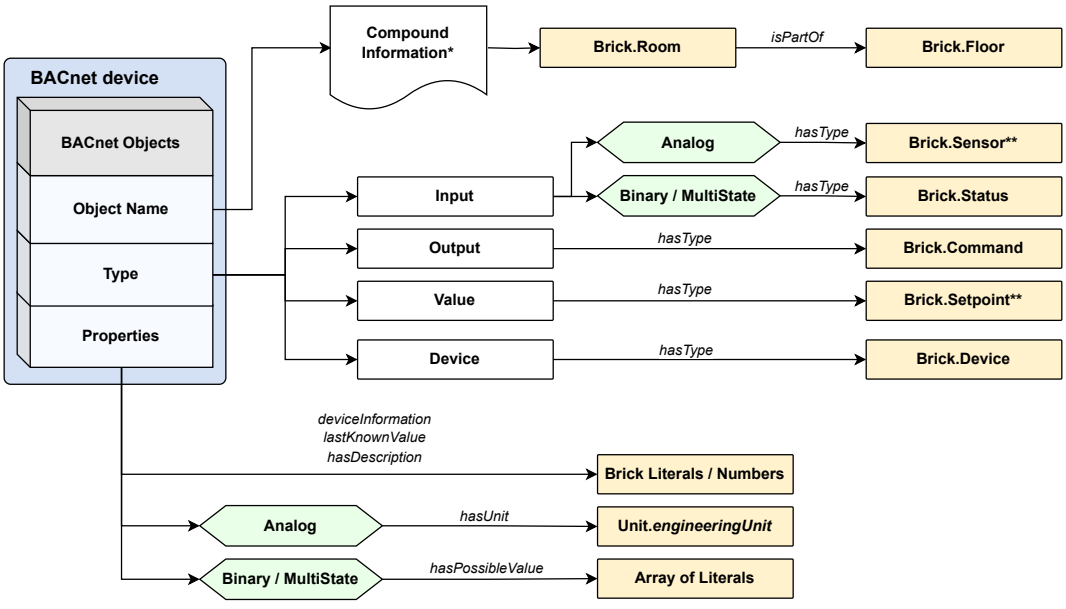

Figure 6.4: Mapping of BACnet properties to their corresponding Brick entities

** The splitting of the compound information is discussed in Section 7.1 Floor and Room Association*
*** Sensors and setpoints of type analog have a more detailed distinction shown in Equation 6.1*

6.5.2 Brick Schema (Python)

The `brickschema` python package is a software library that provides functionality for creating or loading Brick models, validating data instances and performing querying and transformation operations. The package is open-source and its code as well as documentation can be found on Github[1]. Brick is built around traditional Graph-based data structures and offers a wrapper around the RDFlib (RDFLib Team, 2023).

Graph Creation

The main Graph object is just a subclass of the RDFlib Graph library and serves as the starting point for creating a new model. Listing 6.8 shows how to import the library and bind namespaces to a newly created graph. The namespaces are used to access all necessary references and tags. Besides the predefined namespaces, a new namespace called `BLDG` is created, which will be used to reference the characteristics of the building.

```
1  import brickschema
2  from namespaces import A, BRICK, UNIT, BACNET, REF
3  from rdflib import Literal, Namespace
4
5  BLDG = Namespace("urn:commercial_building#")
6  g = brickschema.Graph()
7
8  g.bind("brick", BRICK)
9  g.bind("bacnet", BACNET)
10 g.bind("bldg", BLDG)
11 g.bind("ref", REF)
```

Listing 6.8: Using the brickschema python package to create a RDF based graph and binding namespaces

Floors and Rooms

As suggested by Fierro and Nagare (2022), the next step starts by traversing the data structures created by the serialization task (see Section 6.4). Each object contains possible information about its location like floor and room. The function displayed in Listing 6.9 get the URI of the current object as well as the floor and room passed as parameters. By accessing the `BLDG` namespace, each floor and room gets an associated reference identifier to the building namespace defined previously. Afterwards, triples are added to

[1] https://github.com/BrickSchema/py-brickschema (accessed 26 June 2023)

the graph by the use of the `add()` function. Floors and rooms are considered to be unique entities. In order to prevent duplicates, each URI gets stored in an array of already-known floors and rooms. After checking for duplicates, each room gets linked to its corresponding floor by the `BRICK.isPartOf` relationship. The last triple being added to the graph is the link between the current object and its room location by the `BRICK.isLocatedIn` relation.

```
unique_floor_uris = {}
unique_room_uris = {}

def add_object_location(obj_uri, floor, room):
    floor_uri = BLDG[floor]
    room_uri = BLDG[room]

    if floor_uri not in unique_floor_uris:
        g.add((floor_uri, A, BRICK.Floor))
        unique_floor_uris[floor_uri] = floor_uri

    if room_uri not in unique_room_uris:
        g.add((room_uri, A, BRICK.Room))
        g.add((room_uri, BRICK.isPartOf, floor_uri))
        unique_room_uris[room_uri] = room_uri

    g.add((obj_uri, BRICK.isLocatedIn, room_uri))
```

Listing 6.9: Adding floors and rooms to the Brick graph and linking location information

BACnet Devices

The modelling of device data starts by creating a URI based on the `BLDG` namespace. A BACnet device refers to a physical or virtual entity that supports the BACnet protocol. The available information about each device gets added to the graph in the form of triples. Each BACnet device is also linked to the `BACNET` namespace, by using the `BACNET.BACnetDevice` tag, shown in Listing 6.9. Most of the information about a device is stored as plain text containing possible whitespaces or special characters. In order to not invalidate the graph, these data need to be declared as RDFLib Literals. After modelling the device entity, the created URI for the device gets returned, to traverse the device objects and link them to the device.

```
def add_device_data(device_data):
    device_name = device_data[OBJECT_ID]
    device_uri = BLDG[device_name]

    g.add((device_uri, A, BRICK.Device))
    g.add((device_uri, A, BACNET.BACnetDevice))
    g.add((device_uri, BRICK.Description,
```

```
8           Literal(device_data[DESCRIPTION])))
9       g.add((device_uri, BRICK.Vendor_Name,
10          Literal(device_data[VENDOR_NAME])))
11      ...
12
13      return device_uri
```

Listing 6.10: Adding device information to the Brick model

BACnet Objects

Each BACnet object represents a specific data point or functional component within a device. This can be sensors, setpoints, status, commands and more. The available information of each BACnet object is being added to the Brick graph, capturing details such as type, description or last known value. 4.2.1 External References discusses the external references to BACnet objects, that Brick is able to model. According to this, each object gets linked with its associated BACnet information by an external reference.

```
1  object_uri = BLDG[object_name]
2
3  add_object_location(object_uri, floor, room)
4
5  BRICK_TYPE = get_brick_type(...)
6
7  g.add((object_uri, A, BRICK_TYPE))
8  g.add((object_uri, A, BACNET.BACnetObject))
9  g.add((object_uri, BRICK.Description,
10     Literal(analog_unit[DESCRIPTION])))
11 ...
12 g.add((object_uri, REF.hasExternalReference,
13     [
14         (A, REF.BACnetReference),
15         (BACNET.objectIdentifier, Literal(obj_id)),
16         (BACNET.objectName, Literal(object_name)),
17         (BACNET.objectOf, device_uri)
18     ]))
```

Listing 6.11: Adding BACnet object information to the Brick model

Because of the broad variety of BACnet objects, a helper class was implemented to map specific information about an object to its desired Brick Point, to gain as many details as possible. For instance, an analog input object that measures values with the unit of degrees Celsius can be declared as a `BRICK.TemperatureSensor` which is a subclass of the `BRICK.Sensor` Point. Listing 6.12 shows some of the extra treatments, each type of object needs. For instance, analog objects do have additional information about the

engineering units. Multi-state objects have multiple possible values or states, which can be abstracted by the `BRICK.hasPossibleValue` relationship.

```
# Analog Objects
g.add((object_uri, BRICK.hasUnit, UNIT.DEG_C))

# Binary Objects
g.add((object_uri, BRICK.Polarity,
    Literal(binary_state[POLARITY])))

# Multistate Objects
for state_text in multistate[STATE_TEXTS]:
    g.add((object_uri, BRICK.hasPossibleValue,
           Literal(state_text)))
```

Listing 6.12: Differences of BACnet objects when modeling in Brick

Serialization and Validation

Listing 6.13 shows the last two steps in generating the Brick model. Firstly the graph is expanded based on the Shapes Constraint Language (W3C, 2017). Secondly, the graph is serialized and saved as a Brick model in the Turtle (TTL) file format, which is defined in the RDF framework of the RDFLib Team (2023). To validate the created graph, a reloading of the currently serialized file takes place. Brick has a built-in validation check when loading existing graphs. After loading the `valid` parameter is read to visualize the result of the validation process.

```
# Serialization
output_file = "brick_model_" + current_date_time
g.expand(profile="shacl")
g.serialize(destination=output_file, format="ttl")

# Validation
check = brickschema.Graph()
check.load_file(output_file)
valid, _, _ = check.validate()
```

Listing 6.13: Serializing the Brick model and validating the created graph

6.5.3 Brick Generation Results

Based on the results of the serialization process a Brick model was generated as described in Section 6.5 Brick Ontology Generation. To illustrate the results, the following section will have a look at some excerpts of the generated Brick model. Listing 6.14 shows an example of how floors and rooms are connected. Each floor is associated with the `BLDG` namespace and is tagged as a `BRICK.Floor`. Each room follows a similar procedure with an additional `BRICK.isPartOf` relationship to the floor it is located.

```
1 bldg:EG a brick:Floor .
2
3 bldg:1OG a brick:Floor .
4
5 bldg:R0_027 a brick:Room ;
6     brick:isPartOf bldg:EG .
7
8 bldg:R1_112 a brick:Room ;
9     brick:isPartOf bldg:1OG .
```

Listing 6.14: Floors and rooms example of the Brick model

To describe the BACnet devices within a building, each device was tagged as `BRICK.Device` as shown in Listing 6.15. Additionally Brick offers BACnet support and therefore the device is also marked as a `BACNET.BACnetDevice`. The RDF snippet also contains information about the device description, model name and vendor information. Apparently, there was no location information stored in the BACnet properties of devices. This results in devices being the sole entities in the namespace of the building.

```
1 bldg:device_100 a bacnet:BACnetDevice,
2         brick:Device ;
3     brick:Description "Controller on first floor" ;
4     brick:Model_Name "CRN-7H9Z2" ;
5     brick:Vendor_Id 4 ;
6     brick:Vendor_Name "Vendor Technologies" .
```

Listing 6.15: BACnet device example of the Brick model

The final example shown in Listing 6.16, shows the modelling of a BACnet object from the previously described device. The selected object is a *multiStateInput*. The object was tagged as a `BRICK.Status` Point, which reports a current state, mode or condition of an item. In addition, the entity was tagged as `BACNET.BACnetObject`. The particularity of a *multiStateInput* is that it can have two or more predefined possible values. This can be modelled by using a `BRICK.hasPossibleValue` relation between the object and the possible values as literals. On top of that, Brick offers support for BACnet

objects as external references. In this example, the entity is provided with an `REF.hasExternalReference` block, which holds information about the device, identifier and name of the BACnet object.

```
<urn:commercial_building:X_CRN_7856_Switch> a
        bacnet:BACnetObject,
        brick:Status ;
    brick:Description "Operating mode switch" ;
    brick:hasPossibleValue "Off", "Auto", "On" ;
    brick:lastKnownValue 3 ;
    ref:hasExternalReference [ a ref:BACnetReference ;
        bacnet:objectOf bldg:device_100 ;
        bacnet:object_identifier "multiStateInput_1" ;
        bacnet:object_name "D100_X_CRN_7856_OpModSwi_1" ] .
```

Listing 6.16: BACnet object example of the Brick model

7 Results and Discussion

The following section presents a comprehensive analysis of the ontology graph, focusing on the comparison between the real experiment and the virtual experiment. The analysis begins by examining the overall model in terms of the `rdf:type` property. This property provides insights into the broader categorization of nodes within the ontology. Table 7.1 shows a detailed comparison of all nodes found in both the real and virtual experiments. This table presents a side-by-side comparison of the node types, allowing for a comprehensive assessment of their similarities and differences.

Table 7.1: Comparison of real and virtual ontology model nodes

Type	Real-world model	Virtual model
bacnet:BACnetDevice	9	9
brick:Floor	6	6
brick:Room	103	180
brick:Sensor	35	391
brick:Temperature_Sensor	216	298
brick:Pressure_Sensor	11	264
brick:Flow_Sensor	12	91
brick:Current_Sensor	4	83
brick:Energy_Usage_Sensor	20	212
brick:Humidity_Sensor	33	95
brick:Air_Quality_Sensor	34	90
brick:Electric_Power_Sensor	15	195
brick:Duration_Sensor	0	95
brick:Voltage_Sensor	3	71
brick:Speed_Sensor	0	91
brick:Setpoint	3041	4094
brick:Flow_Setpoint	407	125
brick:Speed_Setpoint	20	127
brick:Temperature_Setpoint	1225	390
brick:Pressure_Setpoint	3	236
brick:Time_Setpoint	2	148
brick:Humidity_Setpoint	71	118
brick:Loop	356	0
brick:Command	163	4672
brick:Heating_Command	1	406
brick:Status	112	3557

7.1 Floor and Room Association

The association of floors and rooms within Brick models plays a crucial role in accurately modelling location information for various nodes within the ontology. The incorporation of spatial context through floor and room associations allows for a more comprehensive representation of the physical layout and relationships between different objects within a building or facility. By explicitly linking nodes to specific floors and rooms, the ontology captures the spatial hierarchy and provides a structured representation of the physical environment.

However, a common problem arises when dealing with building automation systems, such as BACnet, where the location object field is only mandatory for device objects. This limitation can lead to challenges in accurately associating floors and rooms with non-device objects within the ontology model in an automated fashion. To address this issue, technicians of the non-residential building resort to using primitive datatype fields to manually associate floors and rooms with various objects. While this approach allows for some level of location information to be captured, it is less structured and may introduce inconsistencies or incomplete information. Technicians encountered the challenge by utilizing the `object_name` field in the BACnet objects metadata, which is of type string. They addressed this limitation by concatenating relevant details with character delimiters which are illustrated in Figure 7.1. This approach requires subsequent splitting of the textual information to extract floor and room data.

Figure 7.1: Concatenated object information containing floor and room

By employing this workaround, technicians were able to incorporate floor and room associations into the BACnet system. Although the approach required additional processing steps to extract the desired location information, it provided a means to represent the spatial context of different objects within the BACnet system. To overcome this challenge, it is essential to encourage the adoption of standardized practices, that promote explicit floor and room associations for all relevant nodes. BUDO (Stinner et al., 2018), a comprehensive naming scheme, offers a potential solution to the challenge faced with associating floor and room information in the BACnet system. It utilizes a hierarchical structure to encode detailed information in a predefined naming scheme. The solution is pretty similar to what technicians have

done in the practical example of this study, but the naming scheme could vary when it comes to different vendors or buildings. Therefore BUDO was invented to define an overall naming scheme. For better appliances of the naming scheme, the natural language-based machine learning algorithm AIKIDO (Stinner et al., 2019) was developed to automate the translation of building and control systems into the BUDO naming scheme. This results in the reduced manual effort required for data analysis.

7.2 Performance Measures

In this section, performance measurements in terms of time consumption associated with the experiment are explored to gain insights into the duration required for capturing a snapshot and creating the ontology model in both environments, real and virtual. Table 7.2 gives an overview of the snapshot creation times for the real and virtual examples.

Table 7.2: Time measures for snapshot creation of specific devices in MM:SS.sss

Real environment		**Virtual environment**	
Device	**Time**	**Device**	**Time**
WDXXZ_BC01	00:21.747	Device_100	00:00.134
Smart_W100BC01	00:49.086	Device_200	00:00.143
Smart_W100BC02	00:24.774	Device_300	00:00.118
Workstation_1	00:00.191	Device_400	00:00.124
Workstation_2	00:00.121	Device_500	00:00.129
W100BC001	00:25.746	Device_600	00:00.216
W100BC100	00:57.885	Device_700	00:00.125
W100BC200	01:16.160	Device_800	00:00.124
W100BC300	00:36.646	Device_900	00:00.292
Overall*	**04:55.640**	**Overall***	**00:04.632**

**The overall time includes the time for establishing a connection to the BACnet and disconnecting afterwards*

Comparing the real and virtual environments, it is evident that the snapshot creation process is significantly faster in the virtual environment. The virtual devices consistently achieve creation times in the range of milliseconds, while the real devices exhibit higher variation and generally take several seconds or even minutes to complete the process. It's worth noting that the specific devices listed in the table, as well as the measured times, are provided for illustrative purposes. The actual results and their implications may vary depending on the context and the specific devices and technologies involved.

These results suggest that virtualization provides a more efficient and streamlined environment for snapshot creation. The virtual environment offers significant benefits when it comes to testing smart building applications based on BACnet systems. One major advantage is that virtual simulation eliminates the need for real network calls, relieving the burden on the real-world network infrastructure. This not only helps to relieve the building's network but also allows for a more controlled and efficient testing process. Additionally, virtual devices benefit from reduced time overheads associated with physical communication and hardware limitations, resulting in faster and more consistent performance. Ultimately, these benefits of the virtual environment contribute to the development of more reliable, efficient and interoperable smart building solutions.

Table 7.3: Time measures for ontology creation based on system snapshots in MM:SS.sss

	Number of objects	**Time**
Real-world environment	5818	00:10.079
Virtual environment	15920	00:28.583

The second table provides time measures for ontology creation based on system snapshots in both a real-world environment and a virtual environment. Table 7.3 consists of three columns namely the environment type, the number of objects involved in the ontology creation and the corresponding time required for the modelling process.

The time consumption for ontology creation in both environments appears to be closely related to the number of objects involved in the process. In this case, the ontology models were created based on a snapshot of the system state, which allows for more accurate and comparable measurements of time. Looking at the table, it's observable that as the number of objects increases, the time required for ontology creation also increases. This relationship suggests that there is a certain level of computational effort involved in analyzing and structuring each object within the ontology.

7.3 Coverage

Coverage of the ontology model refers to the measurement of how effectively the model captures the elements or features of the target domain. In the context of setpoints coverage is a quantitative assessment of the proportion of setpoints accurately represented in the ontology model.

$$Coverage = \frac{Number\_setpoints\_ontology\_model}{Number\_setpoints\_reference\_dataset} * 100\% \tag{7.1}$$

The calculation can be mathematically expressed by Equation 7.1 where the number of setpoints in the ontology model represents the count of setpoints captured by the ontology and the number of setpoints in the reference dataset represents the total count of setpoints available in a given reference source.

7.3.1 Real-World Environment

The presented Table 7.4 illustrates the comparison between the setpoints covered in the ontology model and a comprehensive technical datasheet provided by technicians for two specific devices. To determine the coverage, the number of setpoints listed in the datasheet is compared with the number of setpoints modelled in the ontology.

Table 7.4: Coverage of setpoints in ontology model compared to technical datasheet

Type	Setpoints		Coverage
	Datasheet	Ontology	
Smart_W100BC01			
Analog	169	171	101% *
Binary	172	152	88%
Multistate	53	49	92%
Overall	**394**	**372**	**94%**
Smart_W100BC01			
Analog	113	109	96%
Binary	76	65	85%
Multistate	27	22	81%
Overall	**216**	**201**	**93%**

**see discussion below*

The analysis of the coverage measurements highlights that the extent to which the ontology model captures setpoints varies depending on the specific device and the type of setpoints. Overall, the ontology model demonstrates a commendable ability to capture a substantial portion of the setpoints listed in the provided datasheet. However, there are instances where certain setpoints were not included in the ontology model.

Missed setpoints in the ontology model can be attributed to several potential factors. One factor is the incomplete documentation provided in the datasheet, which may not offer an exhaustive list of all setpoints or lack detailed information for certain setpoints. This limited information can hinder the modelling process to have access to complete and comprehensive data, leading to the inadvertent exclusion of certain setpoints during the snapshot

creation process. Another factor is the static nature of the datasheet, which represents another snapshot captured on 20th January 2023. As a static representation, the datasheet might not encompass a list of all setpoints or provide detailed information for specific setpoints consequently resulting in the unintentional omission of certain setpoints during the snapshot creation process.

Additionally, the dynamic nature of smart building systems should be considered. Setpoints are subject to changes over time due to updates, modifications or system reconfigurations. If the snapshot used for ontology modelling does not accurately capture the most up-to-date state of the system, it can contribute to inaccuracies and the absence of certain setpoints in the ontology model. Factors such as defects in devices or devices that are currently switched off during the snapshot can also result in missing setpoints.

7.3.2 Simulated Environment

Table 7.5 showcases the coverage results of setpoints in the ontology model compared to the setpoints generated in the virtual and simulated environment. Unlike the previous scenario, there is no datasheet available for the virtual devices since they were generated based on the existing knowledge and expertise with a predefined number of total setpoints.

Table 7.5: Coverage of setpoints in ontology model compared to virtually generated devices

Type	**Setpoints**		**Coverage**
	Datasheet	**Ontology**	
Device_500			
Analog	203	203	100%
Binary	463	463	100%
Multistate	276	276	100%
Overall	**942**	**942**	**100%**
Device_800			
Analog	282	282	100%
Binary	403	403	100%
Multistate	413	413	100%
Overall	**1098**	**1098**	**100%**

Due to the ability to generate virtual devices using the extent of own knowledge, the ontology model achieves complete coverage of setpoints for both devices as reflected in Table 7.5. The model achieved a coverage of 100% for both devices which can be attributed to the enhanced level of control

over the simulated network. This increased control allows for meticulous modelling and the inclusion of all relevant setpoints.

In the virtual experiment, devices were generated based on the extent of own domain knowledge, enabling a controlled and precise representation. However, since the virtual devices were generated solely based on existing knowledge, they do not account for edge cases or specific scenarios that real-world building systems might exhibit. The focus of the virtual devices is limited to the point of knowledge that the experts possess, ensuring that the simulated devices accurately represent the known characteristics and setpoints. The simulated environment is designed to be expandable, allowing for the inclusion of not only the current modelled real-world buildings but also accommodating the potential future capturing of edge cases and intricacies. This expandability ensures that as new building systems are captured and processed, the simulated environment can be augmented to incorporate their unique characteristics and edge cases. By continuously refining and expanding the simulated environment, the virtual representations become more comprehensive and adaptable.

8 Threats to Validity

One potential threat to the validity of this study is the limited scope resulting from the inclusion of only one building as a practical experiment. While the specific observations and findings obtained from this particular building are valuable, the generalizability of the results may be restricted. The characteristics and behaviours observed in this specific building may not be representative of other buildings or vendors in the same domain. To enhance the validity of future studies, it is recommended to consider a larger sample size that includes multiple buildings from different vendors.

Another threat to validity arises from the vendor dependency of the studied building. The control and data collection for the experiment were solely provided by one vendor. During interviews with the vendor, it was confirmed that each building is treated as a separate project and there is no standardized naming scheme applied across all projects, even within the same vendor. This variability in naming schemes and project-specific configurations can introduce potential biases and inconsistencies in the data. Future research should involve collaboration with multiple vendors to gain a more comprehensive understanding of industry practices, naming schemes and configuration across different vendors and projects.

Furthermore, the utilization of optional fields in BACnet objects introduces another threat to validity. These optional fields may be used to provide additional information, but their interpretation and usage can vary between vendors and even within different projects. While optional fields may be used to provide additional information, it is important to note that mandatory fields can also be employed to store and concatenate multiple sources of information within the objects. However, the interpretation and usage of both optional and mandatory fields can vary significantly and result in more complications when it comes to modelling. These variations have the potential to impact the accuracy and generalizability of the study's results.

Additionally, changes in the network over time pose a threat to the validity of the study. The network used for data collection and analysis is subject to modifications, such as updates in network infrastructure, changes in hardware, firmware updates or alterations in the configuration of devices. These network changes can introduce variations in data quality, consistency and the behaviour of building systems.

9 Future Work

This chapter explores potential avenues for further development and enhancement of the research presented in this thesis. Building upon the successes and insights gained from the current study, this chapter outlines several key areas where future work can expand the capabilities and applicability of the developed tool. By addressing these aspects, the research can continue to advance the field of semantic data capture in smart buildings and contribute to the ongoing evolution of building automation technologies.

9.1 Timeseries Classification

In order to address situations where the sensor or object information is limited or unavailable, an important area for future exploration is the development of time series classification techniques (Ismail Fawaz et al., 2019). By leveraging time series data, such as those stored in databases like InfluxDB (InfluxData Inc., 2023), advanced machine learning algorithms can be trained to classify and determine the sensor type based on patterns and characteristics in the data. This approach would provide a solution for automating the identification and categorization of sensors, even in cases where explicit information is not readily available.

9.2 Evaluation of Different Ontologies

While the current research focused on populating a Brick ontology, there is a need to further investigate and evaluate the effectiveness of other ontologies specifically designed for the building sector, such as Project Haystack (Project Haystack, 2023) or SAREF (SAREF, 2023). Conducting a comprehensive evaluation of different ontologies would involve mapping and transforming the captured semantic data into the corresponding ontology structures. By examining factors such as expressiveness, extensibility and compatibility with existing industry standards, this evaluation would provide insights into the suitability of different ontologies for representing building-related

information. Understanding the strengths and limitations of various ontologies would foster interoperability and enable seamless data exchange and integration across different systems and platforms.

9.3 Protocol Adaptions

To enhance the versatility and applicability of the developed tool, a crucial aspect of future work involves adapting the tool to support additional protocols commonly used in building automation. This expansion could encompass protocols like LoraWAN (LoRa Alliance, 2015), Modbus (Modbus Organization, 1979) or ZigBee (Connectivity Standards Alliance, 2004), which are widely employed in various building systems. Adapting the tool to capture and integrate semantic data from these protocols would enable a wider range of building automation scenarios to be addressed. The tool would need to incorporate protocol-specific communication modules or adapters to effectively communicate with and interpret data from these protocols. By supporting multiple protocols, the tool would empower users to capture and leverage semantic data from various building systems, ensuring its compatibility with a wide range of existing and emerging technologies in the building automation industry.

9.4 Expanding the Experiment

To further validate and generalize the findings, it is crucial to expand the experiment beyond a single building from a single vendor. This expansion would involve conducting research on a larger sample size, including buildings from different vendors and architectural and technological characteristics. By capturing semantic data from a more extensive range of buildings, the robustness and reliability of the developed tool as well as the simulation environment can be thoroughly assessed. Additionally, it would enable the identification of any limitations, biases or vendor-specific factors that could impact the tool's performance and applicability in real-world scenarios.

9.5 Evaluation of Machine Learning Algorithms

In addition to the previously mentioned areas of future work, another aspect to explore is the potential benefit of structured metadata in enhancing knowl-

edge management (Fensel, 2001) to enhance the performance of machine learning algorithms. By utilizing the comprehensive metadata structures captured through the developed ontology models, it becomes possible to investigate the impact of this structured information on the effectiveness and accuracy of machine learning models.

A promising direction for future work involves designing experiments where machine learning algorithms are compared directly, with and without the availability of structured metadata ontologies. By conducting comparative evaluations, the usefulness of the ontology models can be assessed in improving the performance of AI algorithms. These experiments would involve training and testing machine learning models on datasets that include both raw sensor data and the enriched metadata from ontology models. The evaluation could focus on assessing various performance metrics, such as prediction accuracy, anomaly detection or energy optimization to determine the extent to which the structured metadata contributes to the effectiveness of the AI algorithms. This could enhance our understanding of the relationship between structured metadata and AI algorithms and create opportunities for the development of more intelligent and context-aware smart building applications.

10 Conclusion

The practical example in this research project showcased the use of affordable hardware and software requirements. Raspberry Pi, a cost-effective and widely available hardware platform, was employed to establish a connection with a real-world smart building. The flexibility and versatility of a Raspberry Pi combined with the use of Python and its available libraries and frameworks provided a practical and accessible solution for integrating with the smart building system. This approach demonstrated the feasibility and applicability of smart building technologies using inexpensive hardware and widely used software tools.

The simulation approach employed in this study has utilized dockerized applications and leveraged the virtual local network provided by the Docker framework in conjunction with Python. This combination of technologies has proven to be highly effective in simulating and understanding the intricate workings of BACnet-based building systems. By replicating the behaviour of these systems, the simulation environment has facilitated a profound understanding of the underlying concept and functionalities of the BACnet protocol. Moreover, the simulation approach has surpassed the capabilities of real hardware when it comes to creating system snapshots. It has provided a streamlined and efficient method for capturing system states. By utilizing Docker and Python, the virtual environment has bypassed the complexities and limitations typically encountered when relying on a physical network. One of the major advantages is the elimination of the overhead associated with real network calls. This independence from a physical network allows for more controlled and repeatable experiments, making the simulation environment well-suited for testing applications and evaluating their performance in a controlled and isolated setting.

The coverage results obtained through the experiments have demonstrated promising outcomes, particularly when considering the real-world example. In the practical experiment, the generated ontology model achieved coverage rates of over 90% for the most critical objects within the building. This comprehensive coverage ensures that the majority of important objects and functionalities are accurately represented within the generated Brick ontology model. The generated ontology model successfully establishes a structured and standardized mapping of semantic data within buildings. In

contrast to the real-world example, it is important to acknowledge that the virtual experiment has been designed to the extent of current knowledge and may have limitations in capturing all the edge cases and intricacies present in a real system. The virtual environment's coverage should be understood in this context, recognizing that it may not encompass all the nuances and intricacies that a real-world system could exhibit. Moreover, by additional programming effort tailored to this project, the experiment has been improved with the inclusion of floor and room mapping to display the spatial hierarchy of the building. This customization played a vital role in conducting a more thorough analysis and evaluation of the building system. This additional effort was specifically done for only one project and cannot be possibly adopted for other buildings, even of the same vendor. It is worth noting that the development of a standardized naming scheme, such as BUDO (Stinner et al., 2018), would be beneficial in ensuring consistency and interoperability across different smart building projects, addressing the current need for a unified approach in naming conventions. Implementing such a standardized naming schema would enhance the comparability and exchangeability of structured information in the field of smart building research and contribute to the ongoing advancements of the domain.

Bibliography

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2017). Agile software development methods: Review and analysis [Publisher: arXiv Version Number: 1]. https://doi.org/10.48550/ARXIV.1709.08439 (cit. on p. 17)

Agarwal, P., & Alam, M. (2020). Investigating IoT middleware platforms for smart application development [Series Title: Lecture Notes in Civil Engineering]. In S. Ahmed, S. M. Abbas, & H. Zia (Eds.), *Smart cities—opportunities and challenges* (pp. 231–244). Springer Singapore. https://doi.org/10.1007/978-981-15-2545-2_21. (Cit. on p. 18)

Ahmad, M. O., Markkula, J., & Oivo, M. (2013). Kanban in software development: A systematic literature review. *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, 9–16. https://doi.org/10.1109/SEAA.2013.28 (cit. on p. 17)

Alfalouji, Q., Schranz, T., Kümpel, A., Schraven, M., Storek, T., Gross, S., Monti, A., Müller, D., & Schweiger, G. (2022). IoT middleware platforms for smart energy systems: An empirical expert survey. *Buildings*, *12*(5), 526. https://doi.org/10.3390/buildings12050526 (cit. on p. 19)

Architecture 2030. (2022). *Why the building sector?* Retrieved July 20, 2023, from https://architecture2030.org/why-the-building-sector/. (Cit. on p. 1)

ASHRAE. (1995). *BACnet*. Retrieved April 13, 2023, from https://bacnet.org/. (Cit. on pp. 1, 6, 21, 25, 28)

BACnet International. (2014). Introduction to BACnet: For building owners and engineers. Retrieved May 13, 2023, from https://www.ccontrols.com/pdf/BACnetIntroduction.pdf. (Cit. on pp. 24, 26, 27)

BACnet International. (2023). *BTL certification* [BACnet] [https://btl.org/about-btl/]. Retrieved May 18, 2023, from https://bacnetglobal.org/btl-certification/. (Cit. on pp. 22, 23)

Baheti, R., & Gill, H. (2011). Cyber-physical systems. *The impact of control technology*, *12*(1), 161–166 (cit. on p. 4).

Baker Jr, R. A. (1997). Code reviews enhance software quality. *Proceedings of the 19th international conference on Software engineering*, 570–571 (cit. on p. 17).

Balaji, B., Bhattacharya, A., Fierro, G., Gao, J., Gluck, J., Hong, D., Johansen, A., Koh, J., Ploennigs, J., Agarwal, Y., Berges, M., Culler, D., Gupta, R., Kjærgaard, M. B., Srivastava, M., & Whitehouse, K. (2016). Brick: Towards a unified metadata schema for buildings. *Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments*, 41–50. https://doi.org/10.1145/2993422.2993577 (cit. on p. 20)

Basford, P. J., Bulot, F. M. J., Apetroaie-Cristea, M., Cox, S. J., & Ossont, S. J. (2020). LoRaWAN for smart city IoT deployments: A long term evaluation. *Sensors*, *20*(3), 648. https://doi.org/10.3390/s20030648 (cit. on p. 8)

Bergmann, H., Mosiman, C., Saha, A., Haile, S., & Livingood, W. (2020). Semantic interoperability to enable smart, grid-interactive efficient buildings (cit. on p. 19).

Bertin, E., Crespi, N., & Magedanz, T. (2013). *Evolution of telecommunication services: The convergence of telecom and internet: Technologies and ecosystems* (Vol. 7768). Springer. (Cit. on p. 18).

Brad, B. S., & Murar, M. M. (2014). Smart buildings using IoT technologies [ISBN: 2304-6295 Publisher: Production, Research and Design Institution" Venchur", Technological Cluster . . .]. *Stroitel'stvo Unikal'nyh Zdanij i Sooruzenij*, (5), 15 (cit. on pp. 5, 6).

Brick Ontology. (2023, May 10). *Brick ontology* [Brick ontology documentation]. Retrieved May 10, 2023, from https://docs.brickschema.org/intro.html. (Cit. on pp. 11, 12, 20, 41–43, 45)

BSRIA. (2018, March). *Market intelligence study on global market pentration of communications protocols*. Retrieved May 18, 2023, from https://www.bsria.com/us/product/B647Pn/world_penetration_of_communications_protocols_2018_8a707622/. (Cit. on pp. 21, 22)

BTL. (2022). *BTL database* [BACnet testing laboratories]. Retrieved May 18, 2023, from https://www.bacnetinternational.net/btl/. (Cit. on p. 22)

BTL Working Group. (2023). *Device testing* [BACnet testing laboratories]. Retrieved May 18, 2023, from https://btl.org/device-testing/. (Cit. on pp. 22, 23)

Buckman, A., Mayfield, M., & B.M. Beck, S. (2014). What is a smart building? *Smart and Sustainable Built Environment*, *3*(2), 92–109. https://doi.org/10.1108/SASBE-01-2014-0003 (cit. on p. 5)

buildingSMART International. (2023). *Industry foundation classes (IFC)* [Industry foundation classes (IFC) - an introduction]. Retrieved June 26, 2023, from https://technical.buildingsmart.org/standards/ifc/. (Cit. on p. 41)

Bushby, S. T., & Newman, H. M. (2002). BACnet today. *ASHRAE journal*, *10*, 10–18 (cit. on pp. 6, 21, 24, 26, 33).

Cao, X., Dai, X., & Liu, J. (2016). Building energy-consumption status worldwide and the state-of-the-art technologies for zero-energy buildings during the past decade. *Energy and Buildings*, *128*, 198–213. https://doi.org/10.1016/j.enbuild.2016.06.089 (cit. on p. 1)

Carlin, D. (2022, April 5). *40% of emissions come from real estate*. Retrieved July 20, 2023, from https://www.forbes.com/sites/davidcarlin/2022/04/05/40-of-emissions-come-from-real-estate-heres-how-the-sector-can-decarbonize/. (Cit. on p. 1)

Chandrasekaran, B. (1994). AI, knowledge, and the quest for smart systems. *IEEE Expert*, *9*(6), 2–5. https://doi.org/10.1109/64.363254 (cit. on p. 9)

Chandrasekaran, B., Josephson, J., & Benjamins, V. (1999). What are ontologies, and why do we need them? *IEEE Intelligent Systems*, *14*(1), 20–26. https://doi.org/10.1109/5254.747902 (cit. on p. 9)

Chaqfeh, M. A., & Mohamed, N. (2012). Challenges in middleware solutions for the internet of things. *2012 International Conference on Collaboration Technologies and Systems (CTS)*, 21–26. https://doi.org/10.1109/CTS.2012.6261022 (cit. on p. 19)

Cimetrics Inc. (2019, October 10). *The key components of the new deal for buildings* [IoT for all]. Retrieved May 17, 2023, from https://www.iotforall.com/bacnet. (Cit. on pp. 2, 19, 21)

Connectivity Standards Alliance. (2004). *ZigBee*. Retrieved April 11, 2023, from https://csa-iot.org/. (Cit. on pp. 7, 81)

Da Cruz, M. A. A., Rodrigues, J. J. P. C., Al-Muhtadi, J., Korotaev, V. V., & De Albuquerque, V. H. C. (2018). A reference model for internet of things middleware. *IEEE Internet of Things Journal*, *5*(2), 871–883. https://doi.org/10.1109/JIOT.2018.2796561 (cit. on pp. 1, 18)

Devesh, M., Kant, A. K., Suchit, Y. R., Tanuja, P., & Kumar, S. N. (2020). Fruition of CPS and IoT in context of industry 4.0 [Series Title: Advances in Intelligent Systems and Computing]. In S. Choudhury, R. Mishra, R. G. Mishra, & A. Kumar (Eds.), *Intelligent communication, control and devices* (pp. 367–375). Springer Singapore. https://doi.org/10.1007/978-981-13-8618-3_39. (Cit. on p. 4)

Docker Inc. (2022, May 10). *Docker: Accelerated, containerized application development*. Retrieved May 12, 2023, from https://www.docker.com/. (Cit. on pp. 15, 16)

Docker Inc. (2023, May 20). *Docker compose overview* [Docker documentation]. Retrieved May 20, 2023, from https://docs.docker.com/compose/. (Cit. on pp. 15, 36)

Dorsemaine, B., Gaulier, J.-P., Wary, J.-P., Kheir, N., & Urien, P. (2015). Internet of things: A definition & taxonomy. *2015 9th International Conference on Next Generation Mobile Applications, Services and Technolo-*

gies, 72–77. https://doi.org/10.1109/NGMAST.2015.71 (cit. on pp. 1, 5, 18)

Erturk, A. (2021, August 25). *What is the difference between BACnet IP vs. BACnet MS/TP?* [Blackhawk supply]. Retrieved May 19, 2023, from https://blackhawksupply.com/blogs/articles/what-is-the-difference-between-bacnet-ip-vs-bacnet-ms-tp. (Cit. on p. 33)

Essa, I. (2000). Ubiquitous sensing for smart and aware environments. *IEEE Personal Communications*, *7*(5), 47–49. https://doi.org/10.1109/98.878538 (cit. on p. 5)

European Commission. (2020, February 17). Energy efficiency in buildings. https://commission.europa.eu/system/files/2020-03/in_focus_energy_efficiency_in_buildings_en.pdf. (Cit. on p. 1)

European Environment Agency. (2022, October 26). *Greenhouse gas emissions from energy use in buildings in europe*. Retrieved March 20, 2023, from https://www.eea.europa.eu/ims/greenhouse-gas-emissions-from-energy. (Cit. on p. 1)

Fagan, M. (2002). Design and code inspections to reduce errors in program development. In M. Broy & E. Denert (Eds.), *Software pioneers* (pp. 575–607). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-59412-0_35. (Cit. on p. 17)

Fensel, D. (2001). *Ontologies: Silver bullet for knowledge management*. Springer. (Cit. on pp. 2, 19, 82).

Fetting, C. (2020). The european green deal [Publisher: ESDN Office Vienna, Austria]. *ESDN report*, *53* (cit. on p. 1).

Fierro, G., Koh, J., Agarwal, Y., Gupta, R. K., & Culler, D. E. (2019). Beyond a house of sticks: Formalizing metadata tags with brick. *Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, 125–134. https://doi.org/10.1145/3360322.3360862 (cit. on pp. 19, 20)

Fierro, G., & Nagare, S. (2022). *BrickSchema documentation*. Retrieved June 27, 2023, from https://brickschema.readthedocs.io/. (Cit. on pp. 64, 66)

Gruber, T. (2008). *Ontology. entry in the encyclopedia of database systems*. Springer-Verlag. (Cit. on p. 8).

Guarino, N. (1997). Understanding, building and using ontologies. *International Journal of Human-Computer Studies*, *46*(2), 293–310. https://doi.org/10.1006/ijhc.1996.0091 (cit. on p. 8)

Hermann, M., Pentek, T., & Otto, B. (2015). Design principles for industrie 4.0 scenarios: A literature review. *Technische Universität Dortmund, Dortmund*, *45* (cit. on p. 4).

Holler, J., Tsiatsis, V., Mulligan, C., Karnouskos, S., Avesand, S., & Boyle, D. (2014). *Internet of things*. Academic Press. (Cit. on p. 6).

InfluxData Inc. (2023). *What is time series data? — definition, examples, types & uses* [InfluxData]. Retrieved June 26, 2023, from https://www.influxdata.com/what-is-time-series-data/. (Cit. on pp. 46, 80)

Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., & Muller, P.-A. (2019). Deep learning for time series classification: A review. *Data Mining and Knowledge Discovery*, *33*(4), 917–963. https://doi.org/10.1007/s10618-019-00619-1 (cit. on p. 80)

Kagermann, H., Helbig, J., Hellinger, A., & Wahlster, W. (2013). *Recommendations for implementing the strategic initiative INDUSTRIE 4.0: Securing the future of german manufacturing industry; final report of the industrie 4.0 working group*. Forschungsunion. (Cit. on p. 4).

Kocakulak, M., & Butun, I. (2017). An overview of wireless sensor networks towards internet of things. *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, 1–6. https://doi.org/10.1109/CCWC.2017.7868374 (cit. on p. 7)

Lê, Q., Nguyen, H. B., & Barnett, T. (2012). Smart homes for older people: Positive aging in a digital world. *Future Internet*, *4*(2), 607–617. https://doi.org/10.3390/fi4020607 (cit. on p. 5)

Lee, C.-H. (2016). *Wireshark over SSH · site reliability engineer HandBook*. Retrieved May 21, 2023, from https://s905060.gitbooks.io/site-reliability-engineer-handbook/content/howto_use_wireshark_over_ssh.html. (Cit. on p. 56)

Li, H., & Hong, T. (2022). A semantic ontology for representing and quantifying energy flexibility of buildings. *Advances in Applied Energy*, *8*, 100113. https://doi.org/10.1016/j.adapen.2022.100113 (cit. on p. 20)

Linked Building Data Community Group. (2021). *Building topology ontology* [Building topology ontology]. Retrieved June 26, 2023, from https://w3c-lbd-cg.github.io/bot/. (Cit. on p. 41)

Long, S. (2019, June 25). *Buster - the new version of raspbian* [Raspberry pi]. Retrieved May 21, 2023, from https://www.raspberrypi.com/news/buster-the-new-version-of-raspbian/. (Cit. on p. 55)

LoRa Alliance. (2015). *LoRa WAN*. LoRa Alliance. Retrieved April 14, 2023, from https://lora-alliance.org/. (Cit. on pp. 7, 81)

Loriot, M., Aljer, A., & Shahrour, I. (2017). Analysis of the use of LoRaWan technology in a large-scale smart city demonstrator. *2017 Sensors Networks Smart and Emerging Technologies (SENSET)*, 1–4. https://doi.org/10.1109/SENSET.2017.8125011 (cit. on p. 8)

Mahdavi, A., & Taheri, M. (2017). An ontology for building monitoring. *Journal of Building Performance Simulation*, *10*(5), 499–508. https://doi.org/10.1080/19401493.2016.1243730 (cit. on p. 11)

Manyika, J., Chui, M., Bisson, P., Woetzel, J., Dobbs, R., Bughin, J., & Aharon, D. (2015, January 5). *Unlocking the potential of the internet of things*. Retrieved May 16, 2023, from https://www.mckinsey.com/capabilities/

mckinsey-digital/our-insights/the-internet-of-things-the-value-of-digitizing-the-physical-world. (Cit. on pp. 2, 19)

Mills, J. (2019, September 24). *Why choose BACnet IP over BACnet MS/TP*. Retrieved May 19, 2023, from https://www.kmccontrols.com/blog/why-choose-bacnet-ip-over-bacnet-ms-tp/. (Cit. on p. 33)

MKFIFO. (2001). *Mkfifo(3): Make FIFO special file - linux man page*. Retrieved May 21, 2023, from https://linux.die.net/man/3/mkfifo. (Cit. on p. 56)

Modbus Organization. (1979). *Modbus*. Retrieved April 15, 2023, from https://modbus.org/. (Cit. on pp. 6, 81)

Mohamed, M. (2018). Challenges and benefits of industry 4.0: An overview. *International Journal of Supply and Operations Management*, *5*(3). https://doi.org/10.22034/2018.3.7 (cit. on p. 4)

National Institute of Standards and Technology. (2001, November). *Advanced encryption standard (AES)* (NIST FIPS 197). National Institute of Standards and Technology. Gaithersburg, MD. https://doi.org/10.6028/NIST.FIPS.197. (Cit. on p. 7)

Ngu, A. H. H., Gutierrez, M., Metsis, V., Nepal, S., & Sheng, M. Z. (2016). IoT middleware: A survey on issues and enabling technologies. *IEEE Internet of Things Journal*, 1–1. https://doi.org/10.1109/JIOT.2016.2615180 (cit. on p. 18)

Ploennigs, J., & Schumann, A. (2017). From semantic models to cognitive buildings [Issue: 1]. *Proceedings of the AAAI Conference on Artificial Intelligence*, *31* (cit. on p. 20).

Powell, J. (1990). Intelligent design teams design intelligent buildings. *Habitat International*, *14*(2), 83–94. https://doi.org/10.1016/0197-3975(90)90038-3 (cit. on p. 5)

Pritoni, M., Paine, D., Fierro, G., Mosiman, C., Poplawski, M., Saha, A., Bender, J., & Granderson, J. (2021). Metadata schemas and ontologies for building energy applications: A critical review and use case analysis. *Energies*, *14*(7), 2024. https://doi.org/10.3390/en14072024 (cit. on p. 19)

Project Haystack. (2023, May 10). *Project haystack*. Retrieved May 10, 2023, from https://project-haystack.org/. (Cit. on pp. 12, 20, 41, 80)

Python. (2023, May 5). *Python.org* [Python.org]. Retrieved May 10, 2023, from https://www.python.org/. (Cit. on p. 13)

Qiang, Z., Hands, S., Taylor, K., Sethuvenkatraman, S., Hugo, D., Omran, P. G., Perera, M., & Haller, A. (2023). A systematic comparison and evaluation of building ontologies for deploying data-driven analytics in smart buildings. *Energy and Buildings*, 113054. https://doi.org/10.1016/j.enbuild.2023.113054 (cit. on p. 11)

Quinn, C., & McArthur, J. (2021). A case study comparing the completeness and expressiveness of two industry recognized ontologies. *Advanced*

Engineering Informatics, *47*, 101233. https://doi.org/10.1016/j.aei.2020.101233 (cit. on p. 20)

Raspberry Pi Ltd. (2023, May 12). *Raspberry pi 4 model b datasheet* [Raspberry pi]. Retrieved May 12, 2023, from https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf. (Cit. on pp. 13, 14)

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 (cit. on p. 18)

RDFLib Team. (2023). *RDFLib documentation*. Retrieved June 12, 2023, from https://rdflib.readthedocs.io/en/stable/. (Cit. on pp. 66, 69)

Rizzi, M., Ferrari, P., Flammini, A., & Sisinni, E. (2017). Evaluation of the IoT LoRaWAN solution for distributed measurement applications. *IEEE Transactions on Instrumentation and Measurement*, *66*(12), 3340–3349. https://doi.org/10.1109/TIM.2017.2746378 (cit. on p. 8)

SAREF. (2023, May 10). *SAREF*. Retrieved May 10, 2023, from https://saref.etsi.org/index.html. (Cit. on pp. 12, 13, 41, 80)

Schwaber, K., & Beedle, M. (2002). *Agile software development with scrum*. Prentice Hall. (Cit. on p. 17).

SSH. (2023). *PAM solutions, key management systems, secure file transfers — SSH*. Retrieved May 21, 2023, from https://www.ssh.com. (Cit. on p. 56)

Staab, S., & Studer, R. (2009). *Handbook on ontologies* (2nd ed). Springer. (Cit. on p. 10).

Stackoverflow. (2022a). *Technology: Most loved, dreaded, and wanted*. Retrieved June 13, 2023, from https://survey.stackoverflow.co/2022#section-most-loved-dreaded-and-wanted-programming-scripting-and-markup-languages. (Cit. on p. 13)

Stackoverflow. (2022b). *Technology: Most popular technologies*. Retrieved June 13, 2023, from https://survey.stackoverflow.co/2022#section-most-popular-technologies-other-tools. (Cit. on p. 15)

Stinner, F., Kornas, A., Baranski, M., & Müller, D. (2018). Structuring building monitoring and automation system data. *The REHVA European HVAC Journal-August, 2018*, 10–15 (cit. on pp. 73, 84).

Stinner, F., Neißer-Deiters, P., Baranski, M., & Müller, D. (2019). Aikido: Structuring data point identifiers of technical building equipment by machine learning. *Journal of Physics: Conference Series*, *1343*(1), 012039. https://doi.org/10.1088/1742-6596/1343/1/012039 (cit. on p. 74)

Swan, B., & Alerton Technologies Inc. (2022). The language of BACnet-objects, properties and services. https://bacnet.org/wp-content/uploads/sites/4/2022/06/The-Language-of-BACnet-1.pdf (cit. on p. 28)

Tcpdump Group. (2023). *TCPDUMP man page*. Retrieved May 21, 2023, from https://www.tcpdump.org/manpages/tcpdump.1.html. (Cit. on p. 56)

Tremblay, C. (2020). *BACo documentation*. Retrieved May 21, 2023, from https://baco.readthedocs.io/en/latest/. (Cit. on p. 59)

Uschold, M., & Gruninger, M. (1996). Ontologies: Principles, methods and applications. *The Knowledge Engineering Review*, *11*(2), 93–136. https://doi.org/10.1017/S0269888900007797 (cit. on p. 8)

W3C. (2017, July 20). *Shapes constraint language (SHACL)*. Retrieved June 27, 2023, from https://www.w3.org/TR/shacl/. (Cit. on p. 69)

W3C. (2023, May 10). *RDF primer* [RDF 1.1 concepts and abstract syntax]. Retrieved May 10, 2023, from https://www.w3.org/TR/rdf-primer/. (Cit. on p. 44)

Wichmann, R. L., Eisenbart, B., & Gericke, K. (2019). The direction of industry: A literature review on industry 4.0. *Proceedings of the Design Society: International Conference on Engineering Design*, *1*(1), 2129–2138. https://doi.org/10.1017/dsi.2019.219 (cit. on p. 4)

WI-FI Alliance. (1999). *WI-FI*. WI-FI Alliance. Retrieved April 10, 2023, from https://www.wi-fi.org/. (Cit. on p. 7)

Wireshark Foundation. (2023, May 12). *Wireshark network protocol analyzer*. Retrieved May 12, 2023, from https://www.wireshark.org/. (Cit. on pp. 16, 17, 56)

Wong, J., Li, H., & Wang, S. (2005). Intelligent building research: A review. *Automation in Construction*, *14*(1), 143–159. https://doi.org/10.1016/j.autcon.2004.06.001 (cit. on p. 5)

YAML. (2021). *YAML markup language* [The official YAML web site]. Retrieved July 4, 2023, from https://yaml.org/. (Cit. on p. 15)

Zhou, K., Taigang Liu, & Lifeng Zhou. (2015). Industry 4.0: Towards future industrial opportunities and challenges. *2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, 2147–2152. https://doi.org/10.1109/FSKD.2015.7382284 (cit. on p. 4)