TU WIEN Informatics

Improving energy community interoperability: a Web of Things approach

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Leonhard Esterbauer, BSc.
Matrikelnummer 01526782

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Mitwirkung: Dipl.-Ing. Jürgen Pannosch, BSc.

Wien, 18. Mai 2022

Leonhard Esterbauer	Wolfgang Kastner

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

TU WIEN Informatics

Improving energy community interoperability: a Web of Things approach

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Leonhard Esterbauer, BSc.
Registration Number 01526782

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Assistance: Dipl.-Ing. Jürgen Pannosch, BSc.

Vienna, 18th May, 2022

Leonhard Esterbauer	Wolfgang Kastner

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

Erklärung zur Verfassung der Arbeit

Leonhard Esterbauer, BSc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 18. Mai 2022

Leonhard Esterbauer

Acknowledgements

During this work, many people have supported me in moving forward and reaching my goals. Above all, my assistant advisor Jürgen Pannosch was always eager to listen to all my concerns and to give me feedback without delay. The same holds for Prof. Wolfgang Kastner, who even made it possible for me to write this thesis in the first place.

Apart from the university environment, I would like to thank my girlfriend, who has continuously supported me and tried to motivate me during difficult situations. Also, my roommates have regularly listened to my concerns and helped me in stressful phases with everyday tasks.

Finally, I would like to thank my parents, who have always believed in me and gave me the opportunity to fully concentrate on my studies.

Kurzfassung

Die EU-weiten Gesetze zu Energiegemeinschaften erlauben es Verbrauchern und Produzenten, Energie gemeinschaftlich zu nutzen. Dadurch ergeben sich vor allem für Haushalte und kleine- bis mittelständische Unternehmen neue Möglichkeiten, um den Energieverbrauch und die damit verbundenen Kosten zu optimieren. Die für eine Optimierung notwendigen Informationen werden zu diesem Zweck aus Sensordaten gewonnen. Durch die Menge an verschiedenen Übertragungsprotokollen und Datenformaten kommt es jedoch bei Integrations- und Kommunikationsprozessen mit Energiegeräten häufig zu Problemen und eine einheitliche Datenverarbeitung wird erschwert. Zusätzlich führen die Integrationsansätze unterschiedlicher Hersteller oft zu komplizierten Installations- und Konfigurationsprozessen, die für technologisch unerfahrene Mitglieder von Energiegemeinschaften nicht ausführbar sind.

In dieser Arbeit wird eine service-orientierte Architektur vorgestellt, welche darauf abzielt, die genannten Probleme mittels Einsatz von Webtechnologien zu erleichtern. Um das zu erreichen, wird im ersten Schritt die Interaktion zwischen Mitgliedern einer Energiegemeinschaft und einem optimierenden System analysiert. Im nächsten Schritt werden aus dieser Analyse für die Architektur relevante Eigenschaften abgeleitet. Das Ergebnis ist eine Sammlung von architektonischen Anforderungen an ein optimierendes Datenverarbeitungssystem. Basierend auf diesen Anforderungen wird anschließend eine geeignete Systemarchitektur gezeigt, welche sowohl für cloudbasierte als auch für lokale Umgebungen oder Mischformen geeignet ist. Um die Machbarkeit zu zeigen, wird die Architektur im Anschluss in einem simulierten Prosumer-Szenario anhand zweier verschiedener Situationen getestet. Abschließend wird eine Einschätzung der architektonischen Lösung im Bezug auf Umsetzung und Sicherheit durchgeführt.

Zusammenfassend beschäftigt sich diese Arbeit mit Interoperabilität und Integration im Kontext von Energiegemeinschaften mit einem Fokus auf service-orientierte und webbasierte Ansätze.

Abstract

Energy communities are a new legal construct that allows energy to be used collaboratively. Especially for households and small to medium-sized enterprises, there are new opportunities for optimizing energy consumption and the associated costs. Sensor data is key for the respective optimizing measures. The problem is the plethora of incompatible protocols and data formats that energy-related devices implement. Similarly, this affects device integration processes. A non-tech-savvy user may not be able to perform complex installation and configuration tasks that devices from different manufacturers require.

This thesis presents a service-oriented architecture that aims to facilitate the stated problems by utilizing web technologies. The first step analyzes the interaction between energy community members and an optimizing system. The result is a collection of architecture requirements for an optimizing data processing system in the energy community context. Further, this thesis presents a suitable system architecture design that fits into cloud-based and local environments or mixed forms. The feasibility of the architecture design is tested utilizing a simulated prosumer scenario with an exemplified implementation of the respective architecture components. The simulation is the basis for verifying the architecture by testing the integration and interaction of two different energy-related devices. The last step of this thesis is an assessment of the architecture design in terms of applicability and security.

In summary, this thesis presents a system architecture for energy communities that facilitates interoperability and integration by utilizing service-oriented and web-based techniques.

Contents

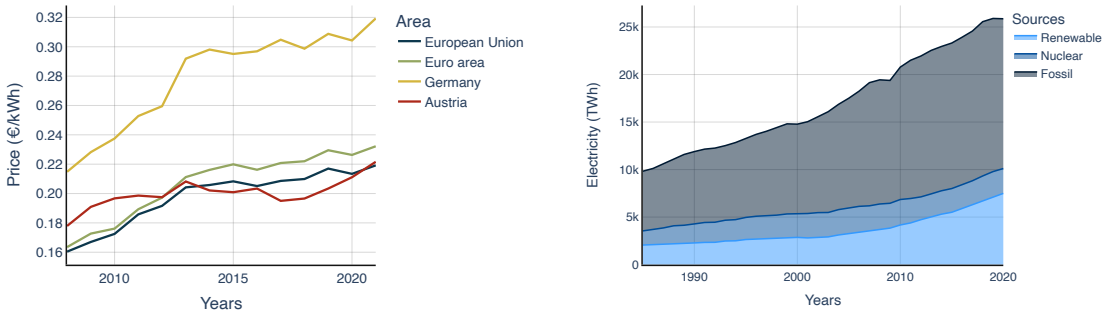
CHAPTER 1

Introduction

1.1 Motivation

The regulations in the *Elektrizitätswirtschafts- und -organisationsgesetz* (ElWOG) [1] and *Erneuerbaren-Ausbau-Gesetz* (EAG) [2] provide a legal framework to share energy across property boundaries with the introduction of Energy Communitys (ECs). Until then, the energy relationship was usually only between consumers and utilities. Now, smaller businesses and citizens can join together and set up contracts for common energy use. Moreover, the regulations enable a testbed for innovative concepts like Peer-to-Peer (P2P) energy trading as described by Long et al. [3]. Although this opportunity is limited to the respective EC members, as shown in another work by Long et al. [4], it is the first step toward a decentralized energy market. Electricity is one form of energy where P2P experiments are easy to handle due to the existence of a well-developed distribution network. If an electricity trade inside an EC fails, the grid operator acts as a backup provider and continues to assure stability in the distribution network. This way, the chance of supply interruption is reduced, and EC members do not need to make any firm commitments. However, P2P energy trading is only one aspect of ECs. Besides the economic intention, ECs also create the opportunity to mitigate energy-related problems of modern society.

The ever-increasing energy consumption causes a considerable part of these problems. In particular, the electricity sector is affected by these problems as depicted in Figure 1.1. The figure visualizes three central problems of today's electricity usage. First, Subfigure 1.1a shows the electric energy price development of selected price indices in Europe. One of the main reasons for the steady increase of electricity prices is the constant rise in taxes and levies, which is evident from Eurostat data [5]. Second, Subfigure 1.1b shows the development of electricity consumption worldwide. From 1990 to 2020, electricity production more than doubled. Third, Subfigure 1.1b visualizes the energy sources of electricity production and indicates a rising dependency on fossil fuels. This fact



(a) Electricity price (incl. taxes) development in Europe for customers with a yearly consumption between 2500kWh and 5000kWh. Visualization is based on data from Eurostat [5].

(b) The world's electricity production classified into the three energy sources fossil fuels, nuclear energy, and renewables. Visualization is based on findings from Max Roser [7].

Figure 1.1: Increasing electricity consumption, increasing dependence on fossil fuels, and increasing electricity prices are three problems regarding electricity usage.

is particularly alarming because fossil fuels emit a significant part of Greenhouse Gas (GHG) emissions, which are the driving force of global warming. Consequently, electricity generation directly impacts climate change [6] and threatens human beings.

In order to counteract these effects, researchers identified measures to reduce the impact of rising energy consumption. A practical way to achieve this is to optimize the utilization of devices since it is not feasible to waiver electricity usage. By applying such optimizing measures, Logenthiran et al. simulated the impact on energy prices and came up with a reduction of 5–10% depending on the kind of energy usage [8]. Similarly, Siano et al. implemented an Energy Management System (EMS) and managed to achieve a saving of 18% of the energy costs for residents without decreasing the level of comfort [9]. Reducing the burden on the environment can be solved in the same way. Renewable energy sources like photovoltaic systems or wind turbines depend on environmental factors resulting in volatilities in supply. Optimization measures can identify these volatilities and suggest ways to increase the portion of renewable energy sources. Han et al. took this effect into account and increased the utilization of renewables by employing monitor and control approaches [10]. Likewise, Celik et al. investigate a method to increase renewable utilization in the neighborhood area by controlling storage units [11].

The basis of scenarios like P2P energy trading or energy optimization is the value and amount of available data. According to Zhang et al. [12], the evolvement of P2P energy markets depends on the expansion of communication and control networks. The gained data is required to facilitate accounting and planning processes. Similarly, qualitative data is a vital aspect of energy optimization. Continuous optimization approaches related to the terms energy management, Demand Side Management (DSM) [13] and Demand Response (DR) [14] primarily rely on real-time data to feed the associated

optimization algorithms [8], [10], [11], [15]–[23]. The principle of these approaches is regulating electricity consumption and production to pursue a predefined goal. In other words, the optimization algorithms suggest or control how electricity is used to reach an energy-efficient behavior. However, because data is the central piece of innovative EC scenarios, it is essential to deal with the interaction of related sensors and devices.

The concept of Internet of Things (IoT) has significantly influenced this aspect [24]. Thanks to IoT, the equipping of material objects with sensors and communication interfaces has become convenient. Consumer home appliances such as bulbs and plugs have already been made *smart*, and mass production for residential applications has started. Moreover, the emerging concept of IoT has affected the energy sector [25] and initiated modernization processes of Information and Communication Technology (ICT) infrastructure. For example, the rollout of smart meters is expected to enable more profound insights into relevant processes [14]. The gained information is precious for the operation of utilities and for stabilizing the distribution network [26]. Besides the widespread use of smart meters, further energy-related IoT applications have been established. In the building sector, monitoring and automation of specific processes are no longer a rarity, and consumer products for resident energy optimization are already sold [27], [28]. However, the rapid adoption of IoT technology led to a diverse landscape of ICTs. Protocols in widespread use are e.g. Hypertext Transfer Protocol (HTTP), Constrained Application Protocol (CoAP) and Message Queuing Telemetry Transport (MQTT), but much more exist [29]. The consequences are incompatibility of devices and system dependency, which both discourage customers from adopting new IoT technology.

This aspect is especially problematic in ECs. Until now, few open standards for home automation have been established, which leads to protocol diversity in the home sector. Consequently, the development of novel systems in the energy domain is complex. Such a situation affects technology acceptance and delays the testing of innovative approaches like P2P energy trading or energy optimization. Especially the implementation of energy optimization systems is vital due to the rising threat of global warming. The solution is either modernizing the entire ICT infrastructure or extending system development to support diversity. Since modernization usually involves considerable costs, the latter should be favored. Therefore, software development needs approaches that focus on how different devices can be integrated into novel systems. This thesis presents a software architecture that supports the diversity of different devices. The focus is on facilitating integration processes and enabling legacy devices to communicate. In order to design the approach, an energy optimization scenario inside an EC is selected. The result is a software architecture that allows members of an EC to integrate and connect devices to external systems and services.

1.2 Problem statement

The primary problem domain of this thesis is the diversity of devices in an EC. Typical settings consist of smart plugs that communicate over HTTP, solar inverters that are

reachable over Modbus TCP, or smart light bulbs that publish data over MQTT. Data from these devices is an essential aspect of upcoming energy scenarios. The purpose of optimizing energy inside of an EC is one such scenario that has the potential to partly mitigate the depicted energy problems in Figure 1.1. Until now, it is not clear what optimization algorithms lead to the desired results. What is certain is that data and device access are a prerequisite for energy optimization. Hence, integration approaches and compatibility measures are needed in any case.

In other words, energy optimization inside of ECs requires a system architecture that allows citizens, devices, and services to communicate with each other despite the usage of different technologies. Moreover, the architecture should be open to promising applications such as P2P trading that may extend the system in the future. A common approach to these problems is bundling responsibilities into services by applying Service Oriented Architecture (SOA) principles. Further, the problem of interoperability is often solved by relying on fixed standards like it is common in the Smart Grid (SG) domain. However, on a local level, research mainly came up with solutions for home optimization. EC characteristics such as voluntariness are rarely covered, and suitable approaches are missing.

Summarized, the main focus of the work is to deal with the question of: **What is a suitable system architecture for an energy optimization scenario inside of ECs?** However, the formulation of this question is broad and concerns various aspects of an architecture. This thesis aims to answer the suitablility aspects in the context of integration and interoperability by utilizing a service-oriented approach:

Integration: It is not clear what data will exactly become relevant for energy optimization purposes. Therefore, the architecture must provide integration mechanisms for legacy, modern, and future data sources. The resulting research question is: *What is a suitable system architecture for facilitating device and sensor integration in the context of ECs? (RQ1)*

Interoperability: Currently, the development of systems continuously relies on web technologies. Therefore, an EC system architecture should support recent web technologies in any case. The architecture must provide a transformation mechanism for other protocols to enable widespread communication. Gateways are a suitable way to accomplish this. The derived research question is: *What is a suitable way of using gateway approaches to increase interoperability in an EC system architecture? (RQ2)*

Service oriented: Innovative use cases for energy communities are just around the corner. A suitable system architecture must concern reusability and extensibility to support innovative use cases. SOA principles help to reuse and extend parts of the architecture. Moreover, a SOA design facilitates the replacement of existing components with novel ones. The research question to this aspects is: *What is a suitable way of applying SOA design principles to an EC system architecture? (RQ3)*

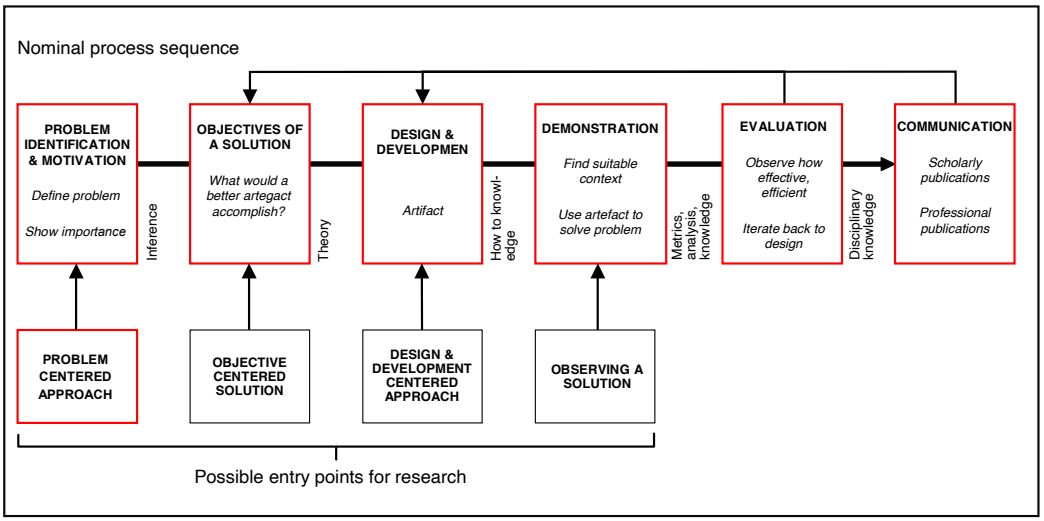


Figure 1.2: DSRM process based on the work of Peffers et al. [30]

1.3 Aim of work

The overall aim of this thesis is to present an architecture software approach that facilitates device integration and interoperability. Since ECs are a new concept, only little research on the topic exists. Moreover, utilization of regulatory measures in real-world scenarios is rare. For this reason, it is still early enough to scientifically investigate the initial situation and propose a suitable system architecture. Consequently, software developers get a broader range of possible approaches they can compare and choose from. The number of considerable solutions can sustainably affect system development and lead to a relief of interoperability in the future.

In detail, this work investigates two aspects of a suitable architecture approach. First, the findings reveal a possible way of implementing device integration while considering energy-related home appliances. Target devices are solar inverters, battery storages, EC loading stations, smart plugs, or smart meters. Second, it is shown how a gateway can be utilized to facilitate device interoperability. The goal is to convert Internet protocols through a third-party service to web-based protocols.

1.4 Method and approach

Approved scientific methods are the basis for delivering qualitative research outcomes. The investigation of the problem statement is therefore based on the Design Science Research Methodology (DSRM) process by Peffers et al. [30]. Figure 1.2 illustrates the chosen DSRM process and highlights the relevant steps in red. After motivating and identifying the problem in the previous sections, the remainder of the process path leads to the following classification of measures:

Objectives of a solution: In the context of this thesis, DSRM solution objectives concern the system architecture's properties. In other words, solution objectives are requirements that a system architecture in the EC context should cover. In order to identify such requirements, this thesis utilizes a requirements engineering process. In detail, the requirements engineering process analyzes a typical EC setting for relevant architecture attributes by applying the user stories methodology by Cohn [31]. The results are used to derive use cases and further identify the architectural requirements.

Design and development: This step outlines the actual design process of the architecture. Visual and textual descriptions of the approach explain the architectural structure. Additionally, sequence diagrams are used to outline the integration and translation mechanisms. Relevant design decisions are based on the requirements from the last step.

Demonstration: This step is about applying the proposed architecture design in a real-world scenario. The basis for this is a simulated EC environment with two different types of devices. On the one hand, it is demonstrated how a compatible device can be integrated into the architecture. On the other hand, it is shown how the architecture can be used to adapt and connect a legacy device.

Evaluation: The application of the architecture in a real-world scenario proves the feasibility of the solution. On this basis, it is shown which requirements could have been met and which not. Additionally, the verification process is discussed, and missing parts of the solution are identified.

Communication: The work is published as a master's thesis and the outcome is further used in the research project *RES* ^{2} [32].

1.5 Structure

The first part of this thesis is the outcome of a literature research in Chapter 2. The contents cover the description of related terms such as EC or energy optimization and further list a selection of related work. After this chapter, the reader should understand common concepts and approaches that are commonly used in IoT environments. Next, Chapter 3 analyzes an EC scenario. The result of this chapter is the definition of four architecture requirements that are further used as a guideline throughout this thesis. On this basis, Chapter 4 presents an architectural design with a focus on interoperability and integration abilities. Design choices orient on nine key aspects that were identified in advance. Next, these key aspects are bundled into concerns and outsourced into five separate services. To show the architecture's feasibility, Chapter 5 implements every service and simulates a prosumer scenario with two different situations of device interaction. Chapter 6 verifies the implementation in a qualitative assessment to the architecture requirements. In addition, this chapter discusses applicability and security concerns of the architecture. The last step of this thesis is summing up the presented

work in Chapter 7 before an outlook indicates future work that might be interesting for the topic.

CHAPTER 2

Background

This chapter introduces selected topics and outlines necessary background knowledge in the research field of this thesis. The primary purpose is to familiarize the reader with the concept of ECs and the related technological concepts. In detail, this includes an overview of optimization algorithms, the Web of Things (WoT) concept, protocols, standards, and other paradigms. In addition, the last section presents a collection of scientific work to which this thesis relates.

2.1 Energy communities

In general, ECs are citizen associations with the intention to consume and produce energy jointly. This abstract definition leads to a number of concrete types of ECs with different goals and application scenarios. Typical examples of ECs are superregional, regional, local and renewable ECs with different goals depending on the distance of the members. Conceivable reasons to form an EC are financial intentions such as energy discounts, distribution network reliefs for blackout prevention or ecological beliefs to decrease GHG emissions. To provide a legal framework for these objectives, the European Union (EU) came up with *EU directive 2018/2001* [33] and *EU directive 2019/944* [34] that all member states have to implement into national law.

In Austria, this requirement was met with the legislation of the EAG [2] and ElWOG [1], which allows individuals and groups to form associations in order to produce, save, consume and sell energy jointly. Conditions of this act omit concrete restrictions of hierarchical and organizational association structures, which enables the founding of clubs and cooperations. Inside an EC, the members are allowed to make up the energy prices themselves and control supply and demand. What is important is that the association must not pursue any profit-making intentions and instead should concentrate on other values such as social, economic, or ecological benefits. For this reason, the legislator

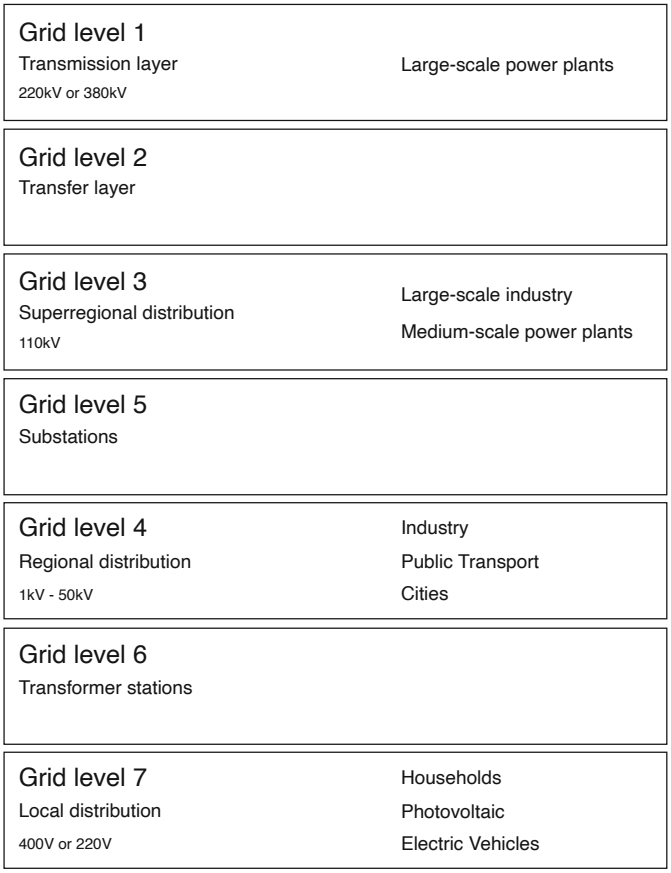


Figure 2.1: Overview of common power grid levels on the left side and example applications on the right side.

calls the associations *energy communities* and classifies them into Renewable Energy Communities (RECs) and Citizens Energy Communities (CECs).

RECs are a kind of alliance that covers local and regional unions. The advantage of this type is that the fulfillment of the EAG lowers the grid fee for regionally or locally energy usage such that just a portion of the regular net costs is due. In other words, distribution network fees are only charged for the utilized grid levels resulting in a discount for regionally consumed energy. Specifically, a local REC that fits into a village's borders gains discounts of up to 57% of the net costs. In contrast, a regional REC gets similar discounts but is tied to the level of the grid connection. This means the foundation of regional REC is only viable when the participants are connected on the grid levels 4–7. The resulting discounts are 64% for grid-level 4 and 5 and 28% for grid-level 6 and 7, respectively. All stated discounts are specified by the Austrian energy authority *E-Control* in their distribution network discount explanation document [35]. This document reflects the grid usage costs at the different grid levels visualized in Figure 2.1.

Just like RECs, CECs are energy-sharing associations. The difference is that CECs omit local restrictions and enable member formations without consideration of the grid connection levels. This way, a CEC neglects the goal of relieving the power grid, and consequently, no distribution network discounts are achievable. Additionally, the possibility of long-distance associations makes it hard or even impossible for all kinds of energy to be shared with other members. The legislator took this problem into account and limited the foundation of CECs to electric energy. Even though these conditions restrict the use, CECs still enable to profit from simultaneously consumption and production of electricity.

In general, RECs and CECs are a first step towards decentralized energy exchange platforms in the EU. The voluntariness of the specification enables citizens to choose if and when they are willing to join an EC. This approach makes it possible to test and experiment with various forms of associations and hierarchical structures. Moreover, it helps to discover acceptance factors and enables a protected setting to test decentralized energy markets while regular distribution network providers take care of network stability.

2.2 Energy optimization

Generally, energy optimization aims to find and execute energy optimizing measures. These measures can reach from modernizing home insulation to adjusting everyday habits. Additionally, optimizing measures can be grouped by the kind of energy they impact. Electricity is a popular kind of energy to optimize due to the well-developed distribution network and the plethora of electricity-consuming devices. The incentives of typical electricity optimizing measures can be:

- **Monetary:** Reduce costs or save money.
- **Grid-relieving:** Relieve the burden on the power grid.
- **Environment-friendly:** Increase renewable utilization.
- **Efficiency-increasing:** Reduce transmission lost or decrease consumption.

Data from various sources serve as a basis for optimization. For example, Chen et al. [36], and Lin et al. [37] use smart meter data to identify behavior patterns and infer individual device usage. Machine learning is utilized to analyze power consumption and subsequently suggest optimization measures. This approach is particularly suitable for detecting energy-intensive applications that can be avoided, such as an unintentionally running electric stove. However, due to the limited view of the environment, the results may not reveal the exact constellation of the system. Therefore, it is challenging to determine extensive measures for uncertain devices. In contrast, the advantage of these approaches is the non-intrusiveness of implementation and deployment. In short, the

stated monitoring approaches reveal a low-effort way of gaining approximate insights into energy usage.

More customization is required when optimizing a neighborhood's electricity. Celik et al. show a management strategy that reduces the electricity costs in a neighborhood [11]. At the same time, the use of Photovoltaic (PV) energy is increased, and the peak load is reduced. To achieve this, two genetic algorithms were implemented and further demonstrated in an exemplary scenario. The results are compared against a previously defined baseline. It is shown that a *Turn-Based Coordination Model* can save 3.35% in costs and reduces the peak load by 12.41%. To apply this algorithm, the authors assume the initial setting of a central aggregator to which all homes are connected. Further, it is assumed that each home operates an energy system that can monitor and control the respective appliances. However, the concrete implementation details are neglected due to the scope of the work. One of the main outcomes is a classification of controllable and not controllable appliances.

Similarly, Xu et al. classifies home appliances into categories such as *power-shiftable* or *time-shiftable* loads [18]. To optimize these loads, the authors formulate the scenario as a DR problem. The solution is a multi-agent Reinforcement Learning (RL) based algorithm that focuses on optimizing electricity costs while minimizing dissatisfying measures. The basis of this algorithm is an approach with estimated hour-ahead scheduling that simultaneously considers the current electricity price and the production data of a PV system. The outcome is an optimizing suggestion that indicates measures like if an Electric Vehicle (EV) should be charged or not. A key aspect of this work is the computational efficiency of the algorithm. Since the prediction has to be done many times during the day, the algorithm must be fast and efficient.

However, not all algorithms can be adjusted to run fast and efficiently in a local environment. Therefore, novel optimization products already relieve local computing units by shifting intensive computing tasks into the cloud. In most cases, the provision of optimization functionality goes hand in hand with a monitoring solution like it is realized by *ntuity* [38] or *Greencom Networks* [39]. The prerequisite of these optimization services is a piece of hardware that acts as a gateway between provider and appliances.

In sum, there is a plethora of different optimization approaches [8], [10], [11], [15]–[23]. A significant part is only tested in simulations or small setups due to the overhead of setting up a real-world environment. If the effort of device integration and ensuring compatibility is facilitated, maybe more optimization approaches would reach market maturity. In other words, there is no question that optimization algorithms exist. The question is how to foster the application of such algorithms by facilitating device interoperability.

2.3 Web of Things

The concept of IoT has facilitated the interconnection of devices, services, and systems. In particular, it is about integrating things into a common network, which in most cases is

the Internet. However, the definition of IoT does not facilitate deducing communication-related properties such as underlying protocols, interfaces, or data encodings. As a result, IoT setups encounter integration and interoperability problems due to the plethora of applied IoT technologies. The WoT concept limits the technology space and forces every participating device to implement web technologies. In other words, WoT is a subset of IoT that focuses on web technologies and standards. On the one hand, this reduces the integration burden on the consumer side due to the widespread availability of compatible applications such as web browsers. On the other hand, WoT facilitates interoperability with business services where web technologies are already established as a popular way of implementing Application Programming Interfaces (APIs).

However, the limitation to web technologies only narrows the conceivable constellations of setups. In other words, interoperability and integration problems still exist on the WoT layer. The previous solution to further limit the allowed communication technologies would not solve the initial problem. Therefore, research came up with other ways to ensure interoperability. A promising solution is to apply semantics and describe different communication technologies in a common description that devices and services understand.

Jara et al. discuss this semantic extension of WoT and identify the outcome as a Semantic Web of Things (SWoT) [40]. Further, the paper outlines the possible evolution of IoT over WoT and SWoT towards global interoperability as illustrated in Figure 2.2. According to the authors, the problem of technology spread could happen like in the beginning days of the Internet. A subset of all available technologies from IoT could establish and further be supported by novel systems and services. If this is the case, a common description of the implemented communication technologies is needed.

Therefore, two of the key points for reaching global interoperability are device abstraction and a commonly understood description, as indicated in Figure 2.2. The work by Jara et al. [40] additionally deals with a wide variety of protocols and semantic solutions that can be used at every stage of the stated evolution. However, a technology, which was not defined at the time of this work, is the WoT standard by World Wide Web Consortium (W3C). This standard aims to enable a uniform description of devices based on web technologies. More details about the WoT standard are described in the following section.

2.3.1 W3C Web of Things

W3C WoT is a modern standard and still in the specification process at the time of writing this thesis. For this reason, the reader might not be familiar with WoT and needs background knowledge to understand the approaches of this thesis. The following sections solve this issue and represent an overview of the main WoT concepts starting with the first comprehensive and detailed description by Dominique Guinard and Vlad Trifa in 2009 [41]. At this time, sensor networks research came up with new technologies and standards that were rarely compatible. Therefore, Guinard and Trifa thought of how IoT devices could utilize the widely used web technologies to achieve a common

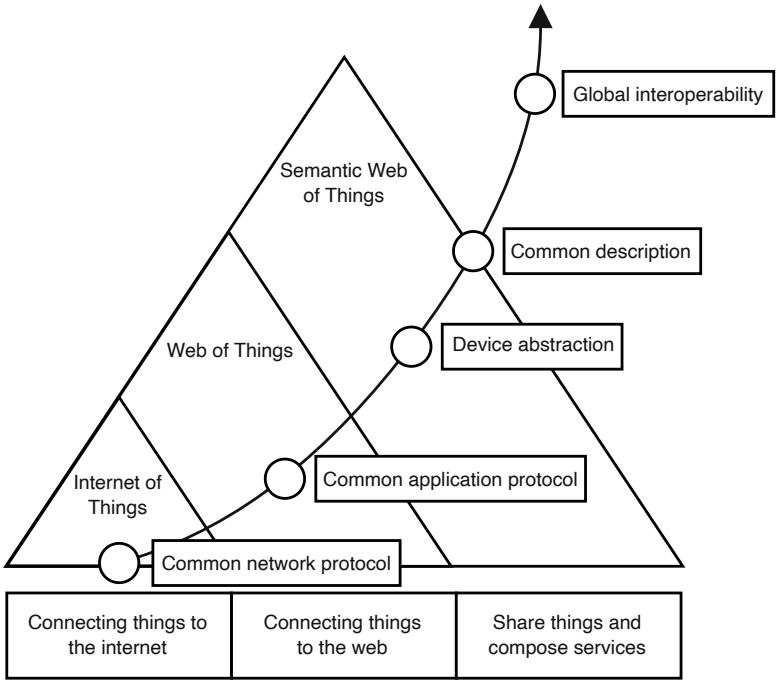


Figure 2.2: Evolution towards a semantic WoT. Adopted from [40].

basis in communication. In order to reach this goal, they defined a layered approach containing concepts for accessibility, findability, sharing, and composition of devices. Figure 2.3 visualizes their concept and additionally shows the location of certain services and technologies.

Besides research, W3C picked up the idea of integrating and describing devices using web technologies, and together with Guinard and Trifa, they formed a first WoT model specification [43]. Subsequently, companies got attentive and started to support the specification process to incorporate their own interests. As a result, W3C formed a Working Group (WG) in 2017 consisting of experts from major companies, foundations and consortia [44].

After the initial planning phase, the WoT WG came up with four different normative deliverables named *WoT Architecture*, *WoT Thing Description*, *WoT Discovery* and *WoT Profiles*. Even though the first version of *WoT Architecture* and *WoT Thing Description* have been classified as official W3C recommendation in April 2020, the specification is still ongoing and new versions will follow. Apart from normative deliverables, also informative deliverables are defined named *WoT Scripting API*, *WoT Binding Templates* and *WoT Security and Privacy Guidelines*. The next sections briefly introduce each deliverable and explain their relations.

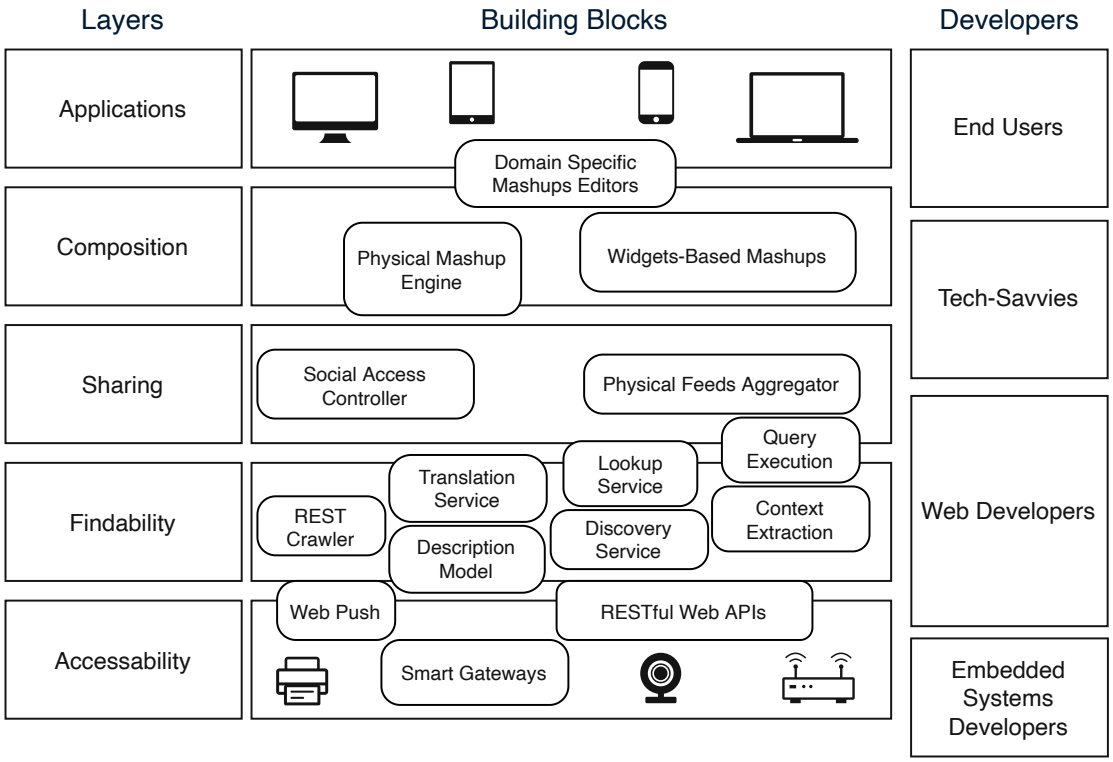


Figure 2.3: Layered WoT approach. Adopted from [42].

Web of Things Architecture

The *WoT Architecture* document acts as a starting point for the specification and explains important terms and relations [45]. A key point of the document is the definition of typical IoT scenarios and the related system architecture requirements. These requirements are divided into functional and technical requirements and consist of topics like device discovery, deployment, and accessibility. Based on these topics, the authors present an abstract WoT architecture concept consisting of multiple integration approaches. The outcome is illustrated in the architecture diagram in Figure 2.4 and a detailed description explains the architecture concepts. Due to the modularity of the approach, the document divides the solution into four parts named *WoT Building Blocks*. Figure 2.5 illustrates these four building blocks and classifies them into *WoT Thing Description*, *WoT Scripting API*, *WoT Binding Templates* and *Security and Privacy Guidelines*. Furthermore, the *WoT Architecture* document shows system and interaction examples to demonstrate how the proposed WoT architecture can be used to foster interoperability in WoT system design.

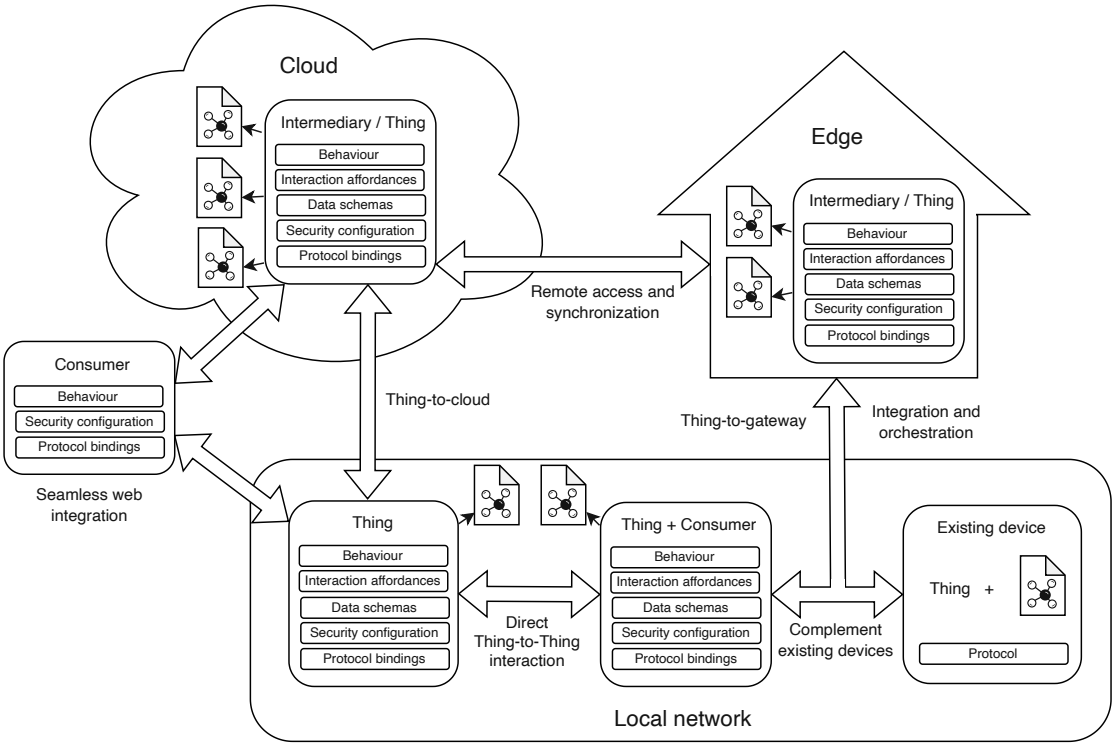


Figure 2.4: Exemplary architecture scenarios in a WoT architecture. Adopted from [45].

Web of Things Thing Description

In order to comply with the *WoT Architecture*, a device must provide a textual representation of its metadata. The resulting document is named Thing Description (TD) and has to be encoded in JavaScript Object Notation for Linked Data (JSON-LD) [46]. A typical analogy from the WoT specification documents is that a TD acts like the *index.html* file of a webpage and represents the starting point for device interaction. The content of this document is structured in a key-value-based manner and contains information like title and description, supported security mechanisms, or available properties of a device. Additionally, the usage of JSON-LD allows the specification of semantically annotated data making it possible to describe advertised data sources directly. This instrument is beneficial when annotating descriptive data in one of the three different device affordances consisting of properties, actions, and events. In order to get all this information, a device must provide the TD in one of three possible ways. Ideally, the device is providing the TD by itself, but it is also allowed that another service distributes the content. The third possibility is to manually specify and distribute the TDs which introduces a workaround for legacy or offline devices.

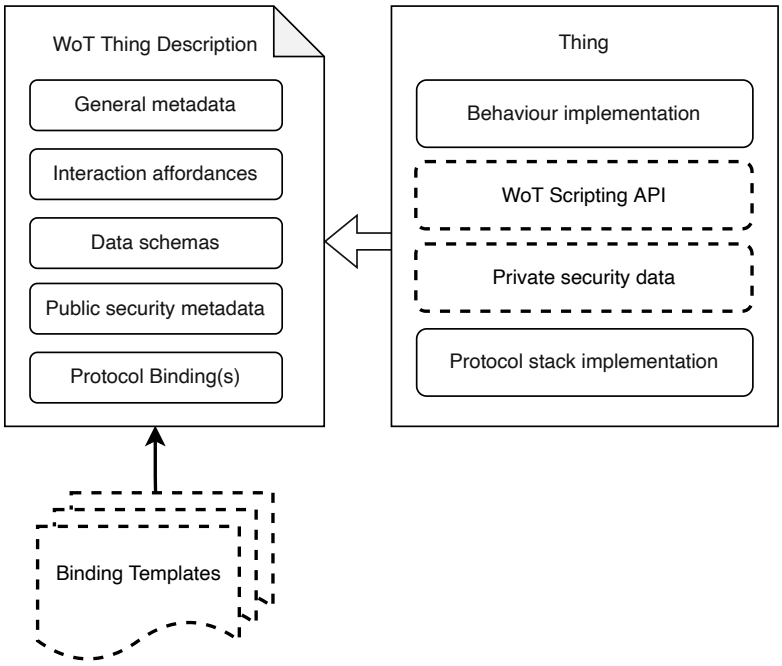


Figure 2.5: Building blocks of the WoT standard. Adopted from [45].

Web of Things Binding Templates

Modern IoT development relies on a variety of protocols and data formats, resulting in a heterogeneous communication landscape. In order to access and manage the different protocols, a description of the underlying communication properties is essential. *WoT Binding Templates* solve this issue by providing a framework that defines the structure and notation of protocol properties inside of TD documents [47]. In other words, *WoT Binding Templates* specify a TD vocabulary extension considering communication-related terms. For example, the vocabulary in the *WoT Binding Templates* specification includes abstract protocol methods like *readproperty* or *invokeaction*, the definition of media types, a data schema for payload structures, and a basis for common data types and value constraints. The advantage of this strategy is that already existing groups of communication properties can be reused, forming a kind of protocol-related template. Additionally, binding templates can exploit the contextual knowledge features of TDs which allow the naming of specific protocol properties and further ease reusability.

Web of Things Scripting API

A typical TD provides information about a device and lists available properties, actions, and event affordances. *WoT Binding Templates* abstract these affordances and bridge the relation of TDs and the underlying communication mechanisms. This separation of metadata and the actual device implementation allows developers to reuse binding templates on different devices and access them in a unified way in application software.

The missing part is a standard way to provide and access binding template functionality. For this purpose, the *WoT Scripting API* [48] defines an interface utilizing the Web Interface Definition Language (WIDL) [49], and describes possible ways to utilize the *WoT Runtime* concept. As a result, the specification enables the creation of reusable binding templates and code snippets, also called scripts or apps. This strategy enables the separation of binding templates and apps, leading to various possible business cases. As an example, future IoT hardware may provide a small runtime environment to execute short scripts or simple apps. Similar to computers or smartphone apps, the definition of common interfaces and runtimes could lead to a market for third-party providers that access IoT hardware only relying on the information available in a TD.

Web of Things Security and Privacy Guidelines

Recently, security and privacy issues have developed into important software design topics. For this reason, the *WoT Security and Privacy Guidelines* document states typical IoT scenarios and examines related threat models that are common in software design [50]. In order to resolve these issues, the authors outline privacy concerns and reference best practices that enable secure and private WoT application design. Special focus is given on WoT related concerns such as configuration metadata abuse or process escalation of WoT runtimes. Essentially, this document has an informal character, and the collection of use cases and mitigations helps system designers to include important security and privacy mechanisms related to WoT.

Web of Things Discovery

Similar to security and privacy issues, the discovery of devices is a cross-cutting concern and an important topic in most software architectures. Thus, W3C's WoT outlines an optional directory structure named Thing Description Directory (TDD) [51]. As the name already reveals, the concept describes a Representational State Transfer (REST) API for TD registration and management. This includes typical scenarios like creation, update, and deletion of TDs and additionally specifies three different ways of searching for certain device properties. JSONPath [52] is one of these defined search endpoints and is required in the standard. The other two endpoints are optional and utilize XPath [53] or SPARQL Protocol and RDF Query Language (SPARQL) [54] queries to expose the search functionality. All three search endpoints enable the submission of queries to explore the content of registered TDs and allow the browsing of desired device properties. The management and search for other TDs is additionally beneficial when different TDDs connect among each other and form interconnected metadata networks. Besides the specification of TDDs, the discovery document also takes into account a group of device discovery mechanisms like Multicast DNS (mDNS) [55], Constrained RESTful Environments (CoRE) link format [56], Well-Known Uniform Resource Identifiers (URIs) [57] or Decentralized Identifiers (DID)-Documents [58] and shows the process of utilizing these technologies to publish a TD document. In sum, the

discovery specification shows how to get into a network, discover certain devices and manage or query available metadata of devices for better interoperability.

Web of Things Profiles

One problem of WoT is that two devices are not necessarily able to communicate even if they comply with the WoT standard. The reason for this is the fact that TDs are simple representations of the provided protocols and formats of a device. The actual devices still communicate over protocols like HTTP or MQTT which leads to a diverse communication landscape. In order to define a common set of communication standards and content formats, a thing can be compatible with one or more *WoT Profiles* [59]. These profiles define a minimum level of properties a device must provide to be compatible with a profile. Consequently, devices compatible with a certain profile are most likely compatible among themselves. As an example, the *WoT Core Profile* suggested by W3C [59] requires every compatible device to provide TD properties like the *title* and the *description* of a device. Additionally, it enforces the device to utilize HTTP for communication and encode the payload utilizing JavaScript Object Notation (JSON). Thus, the definition of different profiles enables compatibility over a set of properties and enables a kind of sub-standards to which different manufacturers can comply.

2.3.2 Smart Grid relevance

According to Faheem et al., IoT is a key technology for enabling SGs [60]. In other words, the interconnection of energy-related devices is one of the cornerstones of modern energy usage. Consequently, developments in IoT impact the evolvement of the energy sector. This means if IoT evolves toward the WoT, adoption in the SG sector likely follows. Indeed, the movement towards web technologies cannot be applied to the entire communications infrastructure in the smart grid. Critical processes still require the use of low latency protocols with certain Quality of Service (QoS) constraints. For example, IEC61850 [61] for substation communication will not be replaced any time soon. However, the trend toward web technologies is undeniable. Novel smart grid protocols such as IEEE2030.5 [62] protocol for *Smart Energy Profiles* already make use of web-based protocols like HTTP as communication basis. To summarize, web technologies are especially well-suited in the smart grid area for non-critical use cases such as monitoring or billing.

Furthermore, the exchange of optimizing measures is a suitable task for web technologies that could build upon the current infrastructure of the Internet. However, ECs likely not utilize these novel protocols yet as older protocols currently dominate. One example of such an old protocol is SunSpec in solar inverters [63]. The specification of this Modbus TCP [64], [65] based protocol is standardized by the SunSpec Alliance. In detail, the specification consists of an information model for decentralized energy generators or storage systems. The open and free protocol is widely used and supported by many manufacturers such as SMA, Tesla, and Texas Instruments. In addition, also other standards depend on the SunSpec specifications. For example, IEEE1547 [66] has defined

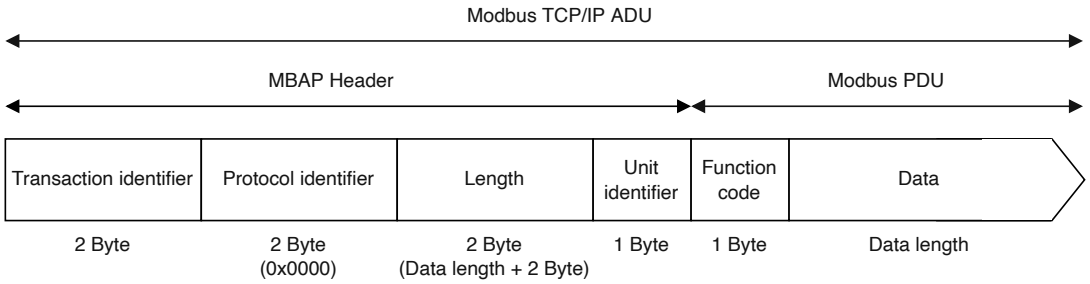


Figure 2.6: Modbus TCP protocol structure.

Modbus SunSpec as one of three protocols for networking in decentralized energy scenarios. However, Modbus is based on an outdated communication paradigm and is mainly used in local industrial environments for observing or controlling systems. The basis of Modbus goes back to the communication via serial interfaces such as Telecommunication Industry Association (TIA)/Electronic Industries Alliance (EIA)-232 [67] or TIA/EIA-485 [68]. At this time, data exchange consisted of reading and writing one or more registers of a computing unit. Hence, Modbus TCP also structures data via registers and coils and provides them with a request/response paradigm.

Figure 2.6 shows the structure of a Modbus TCP message. First of all, the whole message is called Application Data Unit (ADU), which consists of the Modbus Application Protocol (MBAP) header and the Modbus Protocol Data Unit (PDU). Important information in these blocks is the length, the function code, and the data block. In the function block, predefined bit patterns are used to specify whether a register should be read or written. In addition, there are function blocks for reading or writing several registers. The subsequent data block holds the start address of the register, and subsequently, data is appended. By default, a single register is 16 bits long and can hold any kind of binary data.

This basis is utilized by SunSpec, which specifies the data that is stored in the respective Modbus registers. In detail, SunSpec provides different models, which can be used depending on the device type. In any case, SunSpec-specific information starts at register 40001, 50001, or 00001, followed by a mandatory block called *Common Model*. This standard block is thought to contain general information about the device, such as the manufacturer or the serial number. Data representation is covered by the definition of typical data types like int, float, or characters and can also span over several registers. After the first common model, further models follow. The length and semantic information of these models have been defined in the SunSpec specification upfront. Figure 2.7 shows the structure of an exemplary SunSpec compatible device. The figure shows a selection of commonly used SunSpec blocks that most compatible devices provide. If a manufacturer wants to introduce individual blocks, it is necessary to specify the respective block in collaboration with the SunSpec Alliance. Predefined values for blocks are available for inverters, string combiners, storage systems, Distributed Energy Resourcess (DERs), and

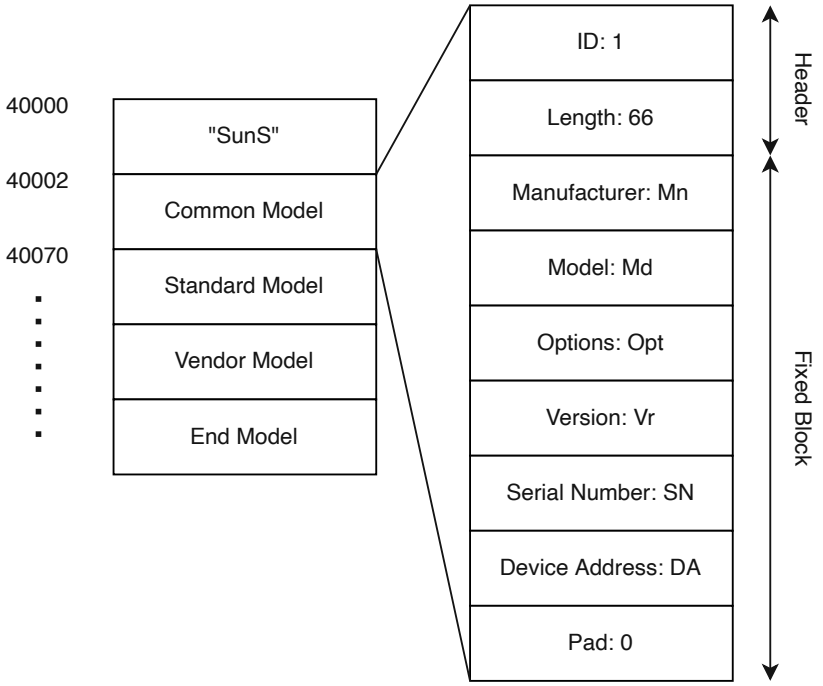


Figure 2.7: SunSpec register assignment.

many more.

Another communication technology that is relevant for energy-related scenarios is HTTP. The rise of IoT modernized home appliances and made them smart. To reduce the integration effort, manufacturers equipped the smart devices with Internet Protocol (IP)-capable interfaces. Consequently, HTTP has increasingly become a standard protocol for smart home devices. For example, there are already smart lighting controls or smart sockets that can be controlled via HTTP.

In particular, HTTP is a text-based protocol and the foundation of the visible web. The first standardized version was published in 1999 by the Internet Engineering Task Force (IETF) and the W3C through Request for Comments (RFC) 2616 [69]. Since then, a lot has changed. Intentionally developed for use in a browser, the protocol emerged as a de-facto standard in business service development. Moreover, it received special attention in combination with Simple Object Access Protocol (SOAP) [70], Web Services Description Language (WSDL) [71] or REST [72]. In detail, HTTP follows a request-response scheme. A typical use case is the download of a resource from a server, which is visualized in Figure 2.8. The actual structure of the protocol consists of two parts. First, a header specifies the protocol method, such as HTTP `GET`, the requested resource, the content length, and some variable properties. The content follows the header and can be formatted in various ways. JSON and Extensible Markup Language (XML) are common formats to exchange content over HTTP.

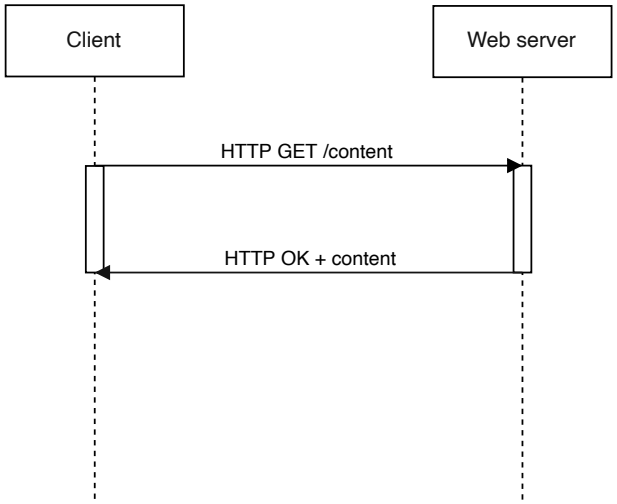


Figure 2.8: Sequence of the common use case of downloading content over HTTP.

Despite its widespread use, the protocol has some disadvantages. First, it is text-based and, by default, provides no compression mechanism to reduce the content size. Second, the keywords in the header have to be advertised in total length. This overhead is a hurdle for many IoT devices. For this reason, the IETF CoRE WG introduced CoAP [73].

CoAP is an application protocol that is based on concepts from HTTP and REST, but builds on the User Datagram Protocol (UDP). In detail, CoAP implements a URI structure and some of the HTTP method verbs. The limitation of verbs and the basis of UDP make the protocol lighter and suitable for constrained environments. Nevertheless, the increase in computing power enables modern devices to be powerful enough to communicate over HTTP. Even though other protocols may be more efficient, HTTP will likely also emerge in the IoT sector. Especially in the home sector where few industrial gateways are deployed, HTTP has a good chance to develop to the dominant protocol standard.

However, the request-response paradigm of HTTP may not suffice for all use cases. Therefore, also other paradigms like publish-subscribe have to be concerned. A popular protocol in this area is MQTT [74]. MQTT is a publish-subscribe-based protocol and requires a broker to relay messages from publishers to subscribers. This structure becomes handy when the publisher is not performant enough to distribute messages to all subscribers. For example, a scalable broker in the cloud can overtake this task and distribute messages from a constrained IoT device. The respective MQTT messages are divided into a header and a variable data block that can consist of a topic and a payload. The topic has the purpose of identifying the channel to which the payload data relates. This topic is formulated using a string with forward-slashes to structure the channels hierarchically like *house/device/sensor*.

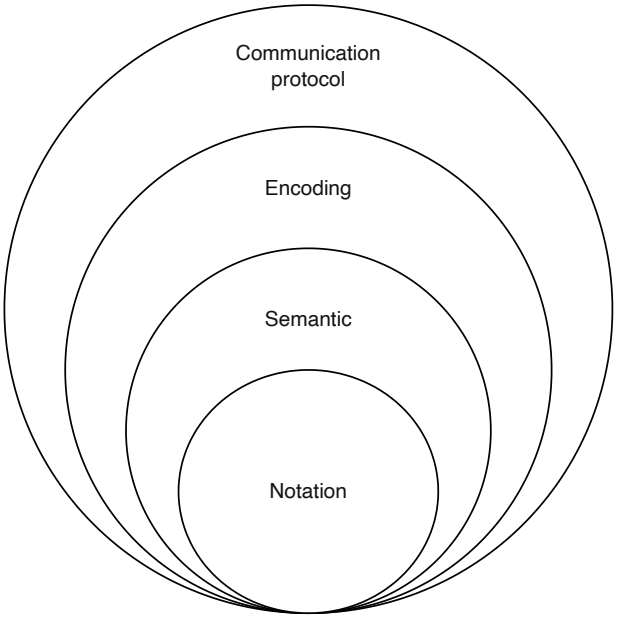


Figure 2.9: Interoperability layers. Adopted from [75].

2.4 Gateways

The outlook in Section 2.3 indicates global interoperability if communication technologies converge. However, from today's perspective, there are still some problems during reaching and after reaching this goal. First, it is unclear how to provide interoperability until novel technologies establish. Second, when a subset of novel technologies has been established, it is necessary to provide backward compatibility to legacy devices even if novel devices are compatible with each other. In general, the problem of communication interoperability can be divided into different levels. Figure 2.9 shows a possible classification based on the work by Paniagua et al. [75]. The classification in this figure is as follows:

1. Communication protocol: covers the underlying communication protocol like HTTP, MQTT or CoAP.
2. Encoding and serialization: represents the format of the data like XML, JSON or plain text.
3. Semantics: gives the encoded data some meaning like SensorML [76], Web Ontology Language (OWL) [77], SunSpec.
4. Notation: defines the naming of the semantics. This is important to infer the same meaning.

Interoperability is usually determined when developing a system. In other words, it is part of the design phase. The reason is that it is difficult to adapt already deployed systems

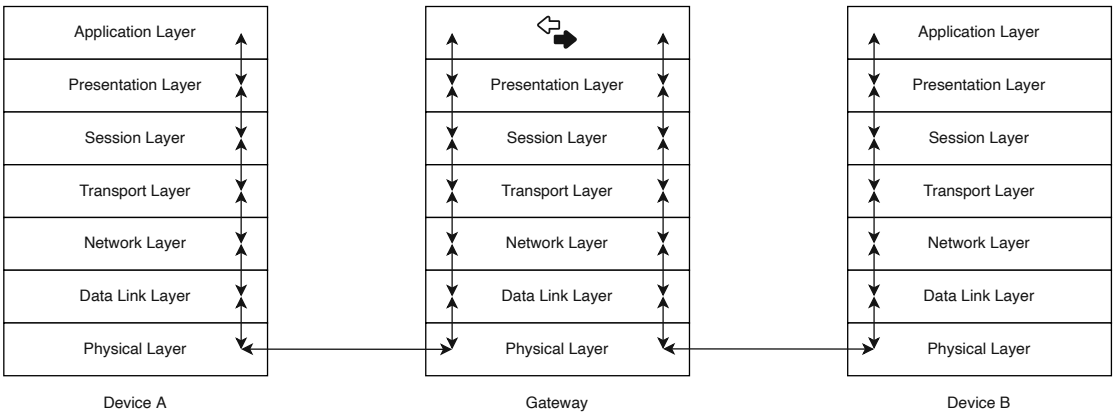


Figure 2.10: Exemplary gateway with respect to the Open Systems Interconnection model (OSI) layers.

to novel communication paradigms or technologies. This leads to the question of how to create interoperability after the devices or systems are deployed. One common way to do this is the utilization of gateways. Gateway approaches exist both as stand-alone systems and as extensions to existing systems. The gateway itself can have different characteristics and can consist of software components, hardware components, or both, depending on the respective application. To illustrate these levels, Figure 2.10 shows the data flow via a gateway along the levels of the ISO/OSI model [78]. In particular, messages are passed from one layer to another and transformed on the highest level of the middle gateway. However, gateways can cover certain technologies on all these layers or only on some. For example, a hardware gateway on the *Physical Layer* can transform simple voltage levels, while a software gateway like in Figure 2.10 can transform the encoding of data on the *Application Layer*. Further, this means that the effort and complexity of a gateway directly depend on the respective layers that should be covered. For example, a gateway that operates only on the software level is easier to change and keep up to date with software updates than a transformation on hardware layers.

Therefore, device-to-IP gateways are popular in the IoT sector for reaching a common consensus. Hardware-related protocols such as ZigBee, LoWPAN or Z-Wave implement individual protocols and in the most cases can not directly interact with other applications. In order to connect these devices devices to computer systems, a device-to-IP gateway is usually deployed. Such gateways are especially common in the home sector to connect IoT devices to an existing Local Area Network (LAN). The resulting network then makes it possible to control the IoT devices with end-user hardware such as smartphones. The common basis of IP allows further gateways to be deployed on a software basis.

Internet routers follow a similar concept than gateways but actually transmit data on lower OSI layers. These routers translate network traffic from the local network to the public network. In general, the router concept is one of the cornerstones in the networking sector. In most cases, the network providers still rely on hardware components.

However, increasing computing capacity makes dedicated network hardware more and more obsolete and leads to the increased use of virtual networks. In fact, the concept of virtual networks has been around for a long time. For example, Virtual Local Area Network (VLAN) is a popular technique on the hardware and Virtual Private Network (VPN) on the software layer.

2.5 Related work

The paper that probably comes closest to this thesis was written by Caballero et al. [79]. Its authors analyzed the current status of the SG and identified heterogeneity as one of the biggest problems. Furthermore, they thought that concepts from the WoT standard could help to simplify the integration of sensor networks. An abstract architecture approach that consists of five layers should solve the issues. The aim is to enable a common Web of Energy (WoE) by transforming all physical devices into common virtual objects. This means the architecture's five layers are abstraction and translation layers to get a common virtual understanding of physical devices. The layers are built on web technologies and utilize the WoT standard. Even though this work repeatedly refers to WoT concepts and states the use of WoT, the integration of devices is merely dismissed with the presence of a gateway. In other words, gateways or smart devices are supposed to communicate directly within the WoE, and WoT is used to describe these devices. Further, it is assumed that incompatible IoT protocols are translated through a gateway. In a proof of concept implementation, the authors show a translation from MQTT to Websockets using the *Actor Model* paradigm. In conclusion, this work aims to create a common basis for communication and connectivity. Semantic interoperability is neglected.

In contrast, Desai et al. make semantics a central part of the solution [80]. The authors propose a gateway architecture that enables a translation of IoT protocols into other protocols. As an example, messages from CoAP are translated to MQTT and from MQTT to REST. A software component named *Multiprotocol Proxy* implements interfaces to various clients such as CoAP or MQTT. Messages from these interfaces are fed into a message broker and represented as *topics* managed by a component named *topic router*. The message broker additionally stores the values and, if requested from outside, semantically annotates them. A semantic annotation service takes care of this process and transforms JSON formatted sensor data into semantic knowledge utilizing SensorML, SSN ontology, and other related technologies. The outcome is a semantically annotated Resource Description Framework (RDF) message that is further published over REST formatted as JSON-LD or XML. In this way, external services can request understandable and informative data even if the underlying IoT devices only implement device-specific protocols. Similar to this thesis, it is assumed that all devices are already on a common physical layer. However, it is not mentioned how the device interfaces are integrated into the whole software architecture. Additionally, the paper neglects the management of the devices and how the devices interact. The focus is clearly on facilitating the connection to the cloud. External devices like smartphones or services should be able to universally retrieve data from home appliances.

Similarly, Kim et al. present a gateway architecture that abstracts IoT protocols and semantically annotates the retrieved data [81]. In a home energy-saving scenario, the authors outline a gateway that supports the technologies Devices Profile for Web Services (DPWS) and MQTT. Additionally, other protocols are supported if a translation to MQTT is feasible. The gateway takes over tasks like device management and data aggregation and publishes data to an *IoT Service Platform* utilizing REST. Data on this level is enriched with ontologies, and a SPARQL engine is able to infer the interrelationship. A service executor takes this data as input and tries to fulfill the requirements of the system. As a result, the service executor can investigate a specific situation and determine what actions are needed to reach a predefined goal. For example, if a temperature sensor in a home environment reports a low temperature, the executor can take action and sends the home gateway the command to start heating. This way, it is possible to connect different IoT devices in a common platform and analyze the situation without being compatible with every single device. A shortcoming of this work is the reliance on DPWS and MQTT even though other protocols could be connected when a transformation process is implemented. Additionally, the self-defined description of devices could be a problem for different systems and may complicate the usage with external service providers.

In contrast, Chen et al. take third-party service providers into account [82]. The work even assumes that DSM services are typically cloud-oriented. With this assumption, the authors identified the problem of high latencies between the actual demand side and the cloud. An architecture approach with on-site parts and cloud parts should solve the issue. To achieve this, the authors presented an Artificial Intelligence (AI) powered hardware part that is located on-site and executes real-time analytics. Through an energy management controller, the whole setup is connected to DR services. Additionally, the on-site setup is connected to the cloud, where other third-party services can use the information of the system. For example, a prototypical implementation shows the usage of If This Then That (IFTT) and sends messages if electricity rises over a specific threshold. This way, residents will be informed by online messaging or push services over relevant events related to electricity usage.

Another paper that mentions third-party services was written by Moghaddam et al. [83]. This work presents a fog-based energy architecture that enables external services to gather data over the cloud. The key aspect of this work is to split services into fog-related tasks and cloud-related tasks. Energy optimization, e.g., is a fog-related task that benefits from a low latency connection. Additionally, fog nodes can aggregate data and prevent that private energy consumption data is published to the cloud. This way, only justified data like billing information is sent to the cloud. On the sensor device level, a home gateway comes into play for managing and interconnecting devices. In detail, the authors show the integration of devices and implemented a proxy for HTTP and CoAP. The proxy is capable of translating these protocols among each other and sends the outcome to the respective fog or cloud nodes. Additionally, the paper presents a price function for electricity prices that enable optimized day-ahead scheduling. Nevertheless, this approach concentrates on the abstract architecture with a special focus on the fog

layer and additionally presents an optimization of costs. The terms integration and interoperability are mentioned but not discussed in detail.

Likewise, Atzori et al. deal with the fog-level [84]. The paper presents an IoT service architecture that aims to reduce the distance between home and cloud scenarios. In detail, Software Defined Networking (SDN) and Network Function Virtualization (NFV) are utilized to extend the network range of current topologies. For example, a typical local home network with virtual network functionalities can reach over multiple layers up to the cloud domain. In the fog domain, it is very likely that services provide better latencies than the cloud. On the other hand, the fog domain likely provides more computational power than the home devices. In other words, parts from the home network and parts from the cloud can be transferred into the fog, where latency is reduced, and reliability is increased. Additionally, the fog layer can establish new services. When thinking of an IoT device that transfers data into the cloud, the fog layer could aggregate the data or provide caching mechanisms. Further, device-related services could be transferred into the cloud. The resulting concept is named *Smart Devices as a Service (SDaaS)*. The connection of devices and services can be changed with low effort due to the utilization of SDN and NFV. Also, the authors identified the current Internet Service Providers (ISPs) to be a suitable provider for operating the fog layer. The justification is that an Internet connection always requires some sort of gateway that is connected to a network. ISPs could extend their services to also run a kind of virtual network on their gateway hardware.

Another gateway architecture is proposed by Han et al. [10]. The authors outline the architecture of a Home Energy Management System (HEMS) that is connected to solar panels utilizing Power Line Communication (PLC). Other consuming devices in the household are connected over ZigBee. This way, the HEMS gains an overview of the current situation of energy consumption and production. As a result, data aggregation and analysis are possible that enable DSM purposes. On the communication layer, the gateway supports PLC, Ethernet, and RS-485. The paper does not exactly state how the data is transformed. Instead, an illustration of a ZigBee message encoding is given. Data inference is also not stated, only a fixed calculation schema is presented. In other words, the gateway consists of different blocks, but all have hardcoded functionalities and interfaces.

This problem is further investigated by Brundu et al. [85]. The aim of this work is to present an IoT architecture for energy management with the ability to test associated measures in simulations. For this, the authors first divide the architecture into three layers, namely *IoT Devices and Technology Integration Layer*, *Service Layer*, and *Application Layer*. The purpose of the classification is to determine standards within the layers. In detail, the application layer covers clients like smartphones, desktops, or other web clients. The restriction is to access the service layer over Web Service APIs. The service layer is responsible for maintaining a device catalog, storing historical data, performing simulations, and managing semantic metadata. Also, this layer relies on the use of Web Service APIs. The translating part of the architecture is the device connectors from

the integration layer to the service layer. Different devices or systems must provide a translation from device or service-related protocols to Web Service APIs. The paper outlines the use of HTTP and MQTT and mentions the use of SensorML. Nevertheless, the concrete translation process is not covered in detail. It is assumed that the device connectors expose Web Service APIs.

The missing part of the previous paper is discussed by Derhamy et al. [86]. This work proposes a SOA based multiprotocol translator. By using the Arrowhead framework, the authors present a transparent translator service that can be used on-demand and does not introduce design-time dependencies. Additionally, it can also be executed in a cloud environment and handle many concurrent connections or run in a local cloud to reduce latency. In detail, the architecture specifies an intermediary format that is used to describe protocols. In other words, a message of one protocol is first transformed into the intermediary format and then transformed into another protocol. This way, every protocol needs only one translator from the intermediary format to a specific protocol and back to communicate with other devices. The concept of this approach is further analyzed by Paniagua et al. [75]. Here, the different types of interoperability are investigated, and the individual translation of every encoding, protocol, and semantic scenario is assumed to be infeasible. Therefore, the authors introduced a new language to model the protocols named Contract Description Language (CDL). Documents of this type are based on XML and are collected in a CDL database. If two actors use different protocols, an additional actor named adapter can look up the protocol translations and instantiate an adapter that translates one protocol to another. For the part of translation, this solution comes nearest to the solution presented in this thesis. The problem with the approaches from Derhamy et al., and Paniagua et al. is that they both use individual formats for describing the related protocols. If the intermediary protocol is another non-standardized mechanism, it is likely that this solution will never establish. Additionally, it is neither exactly specified how the SOA based architecture of the solutions instantiate the services nor how the lookup process for protocol descriptions works. The following architecture solves this issue and outlines how this approach can be used for establishing a software architecture in an EC related scenario.

CHAPTER 3

Requirements

The following chapter analyzes the current situation in ECs. The aim is to identify architectural system requirements that represent the basic needs of an EMS. In other words, this chapter describes a requirements engineering process with the goal of finding high-level requirements. The first part of the chapter explains the applied process in more detail, and the other parts describe the actual examination of the requirements engineering process. Finally, the last part comes up with four architectural requirements that a suitable system architecture in the context of energy optimization should cover.

3.1 Process

The requirements engineering process consists of three steps that are illustrated in Figure 3.1. The figure represents the outcome of each step that is classified into:

- **Step 1: User stories**: Requirements engineering starts with the derivation of interaction scenarios while adhering to the regulatory measures that the Austrian government defined in the respective energy acts ElWOG [1] and EAG [2]. This action yields a high-level view of the environmental situation and comes up with a collection of user stories that are formulated according to the template by

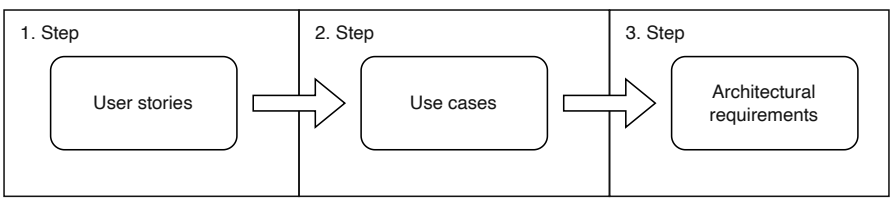


Figure 3.1: Graphical representation of the applied requirements engineering process.

Cohn [31]. The foundation of this step has emerged in the context of the research project *RES* [2] [32].

- **Step 2: Use cases**: This step utilizes the identified user stories and forms more concrete use cases. The aim of this step is to structure the user stories from the previous step and identify user stories with overlapping actors. The findings are illustrated in a Unified Modeling Language (UML) use case diagram and described in a list of associated functional and non-functional requirements.
- **3. Architectural requirements**: The last step is to infer architectural requirements from the specified use cases. In detail, this step identifies requirements that apply to a wide range of EC scenarios. In other words, the aim is to identify architectural requirements which fit the identified use cases but at the same time suit various other EC scenarios.

3.2 Step 1: User stories

In general, the Austrian government specified only a few restrictions for ECs. In other words, the purpose of optimizing energy usage inside of ECs can be implemented in various ways. However, there are two reasons to favor a central optimizer outside of an EC member's building. First, the conduction of optimizing measures needs collaborative and member-overlapping knowledge. Second, optimizing algorithms have increased in complexity such that end devices are no longer capable of running these algorithms. Therefore, this thesis assumes the optimizer to be a central unit that is located outside an EC member's building. This assumption leads to the identification of two actors, namely EC members on the one hand and optimizers on the other hand. The concrete characteristics of each actor are examined by placing oneself into their respective roles in the architecture. The outcome of this step is outlined in the following collection of user stories that describe the actor's expectation of an EMS:

1. EC members want to utilize their devices for optimizing energy usage.
    a) EC members want to integrate their devices into optimizing systems.
        i. Technically inexperienced EC members want to integrate their devices into optimizing systems without technical know-how.
        ii. Technically experienced EC member want to manage devices themselves.
        iii. EC members want to integrate already deployed devices into optimizing systems.
    b) EC members want to have an overview of their devices.
    c) EC members want to perform optimizing measures.
        i. EC members want optimization measures to be performed automatically.
        ii. EC members want to decide about what devices can be utilized for optimization measures.

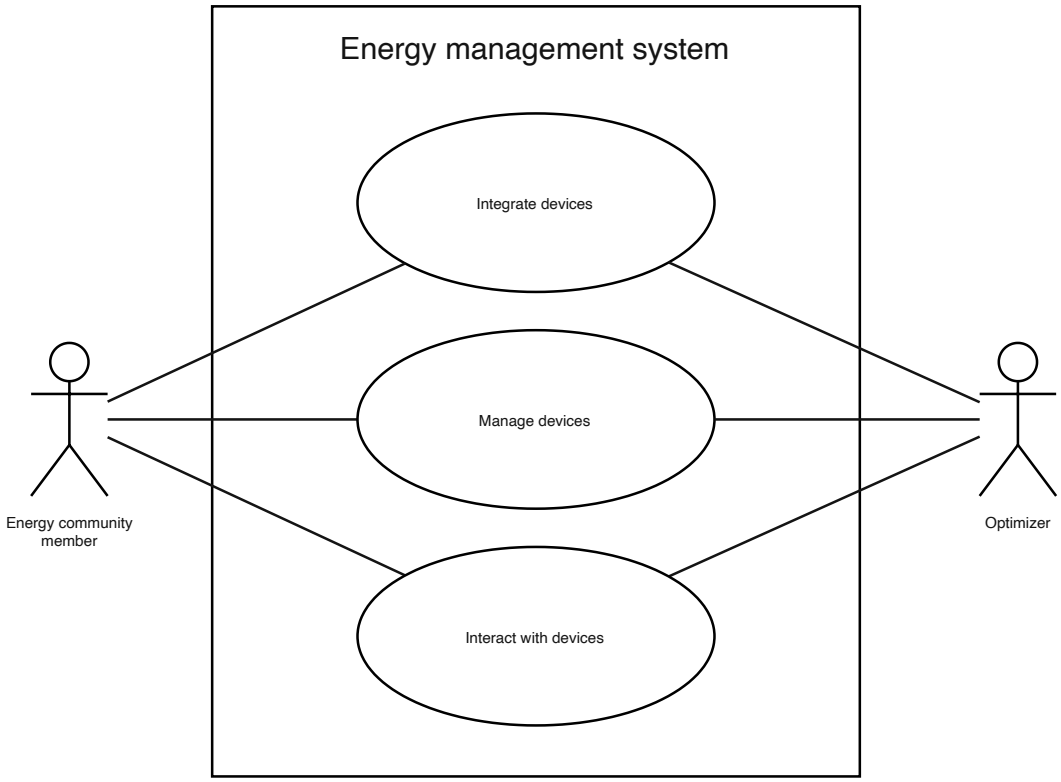


Figure 3.2: High-level use cases modeled in an UML use case diagram.

d) EC members want to link or extend the system for their purposes.

e) EC members want to expose their devices securely and privately.

2. Optimizers want to optimize energy usage.

a) Optimizers want to get an overview of available devices.

b) Optimizers want to read sensor data from devices.

c) Optimizers want to control devices.

3.3 Step 2: Use cases

The user stories from the last section presented each actor's view of the system. This is helpful to get an idea of what goals the different actors pursue. As a result, the user stories reveal a focus on device management and interaction. In short, the optimizer wants to get access to an EC member's devices, and the EC member wants to enable that under certain conditions. From a functional perspective, the device interaction can be bundled into three similar use cases that are depicted in Figure 3.2. The use cases in this figure represent a high-level view of the functional requirements of the system. The

following subsections explain these functional requirements in more detail and further outline the non-functional requirements.

3.3.1 Integrate devices

At any point in time, EC members should be able to integrate new devices into the system. Similarly, the optimizer should be able to access new devices on purpose. In other words, it should be possible to dynamically expand and shrink the system as the actors want to. The actual device integration use case primarily involves EC members that provide some devices and want to establish a connection between the system and the respective devices. The basis for this requirement is mainly derived from user story 1a. The associated sub-stories 1(a)i and 1(a)ii further require the system to provide integration mechanisms for tech-savvy and non-tech-savvy EC members. In other words, EC members should be able to choose the level of detail they want to be involved in the integration process. Consequently, the system should cover the respective level of effort that an EC member is willing to spend.

3.3.2 Manage devices

The actors should be able to change or observe the constellation of the running system. The respective requirement contains tasks like linking, extending, or configuring devices and the associated system as indicated in user stories 1b and 1d. Above all, this requirement concerns EC members due to the ownership of the respective devices. However, optimizers also need insights into the device constellation and, therefore, should be able to get an overview as indicated in user story 2a. This purpose additionally relates with the EC member's user stories 1e and 1(c)ii that deal with security and privacy concerns. In detail, EC members should be able to fine-control which devices are accessible by the optimizer and which are hidden.

3.3.3 Interact with devices

This use case covers the actual interaction of EC members and optimizers with the respective devices. For the EC member, this means to access the full capabilities of a device. In other words, a device's interfaces, which could provide informative or controllable functionalities, should be accessible by EC members without limitation as outlined in user stories 1b and 1c. Similarly, the optimizer wants to interact with devices as indicated in user stories 2b and 2c. However, device access for optimizers must be protected due to the security and privacy concerns of the EC member in user story 1e. Another essential topic of this use case is the uncertainty of the communication protocol. EC members and optimizers should be able to interact with devices despite different communication technologies. This aspect is especially important when dealing with already deployed devices as outlined in user story 1(a)iii.

3.4 Step 3: Architectural requirements

The last step in the requirements engineering process is to derive architecture requirements for an EMS. This step reveals the desired properties of an architecture which enables EMS development that meets the user's needs. Naturally, the most important task is to ensure that the architecture allows the implementation of the functional requirements. From a technical point of view, this means the architecture must support at least one way of implementing the intended use cases from Section 3.3. However, the strict implementation of one such EMS is not the aim of this work but rather is an architecture approach that exhibits specific characteristics. In other words, the focus of this work is to present and argue architecture design decisions that lead to specific system properties. This approach requires the definition of measures to evaluate the quality of the solution. The following properties represent these quality measures and outline the desired characteristics of a suitable EMS architecture.

3.4.1 Confidentiality and access control

Description: Every public network or device exposes reachable endpoints that attackers may exploit. In the case of the proposed EMS in Figure 3.2, the main attack points are the communication of devices and service providers. For this reason, the system must take precautions so that no other parties gain access to devices and data. Additionally, access to the local network of EC members should be restricted as far as possible. A best-practice measure to lower such threats is the least privilege principle. That means the system configuration restricts access and authorization policies to the least minimal functionality for the system to work. No other provider should have access to data that is unnecessary for operation. This measure also follows best practices regarding privacy issues and is in line with the orientation of the General Data Protection Regulation (GDPR) [87].

Objective: EC members want their devices and data to be confidential, and no unauthorized person should gain access. Likewise, Third Party Service Providers (TPSPs) and the optimizer want to protect their services and use various authentication, authorization and privacy mechanisms. An EMS architecture should support such mechanisms.

Fulfillment criteria: The requirement is fulfilled if the architecture supports authorization mechanisms that allow fine-grained access control. Additionally, the architecture must ensure that traffic between local devices and external parties is fully encrypted.

3.4.2 Extensibility

Description: ECs are a new concept, and the related system development has just begun. This situation leads to new opportunities, and additional business cases may evolve. Consequently, an EMS should enable the expansion of functionality. For example, an EC member who wants to integrate a new device needs a way to expand the current constellation of the system. TPSPs benefit from this property in a similar way. If

new business cases evolve and services are needed, TPSPs want to expand the system functionality to provide additional value.

Objective: An EMS architecture should enable the system to be open to changing use cases and provide room for extensions. For this reason, EC members and TPSPs should be able to integrate new devices or services at any time.

Fulfillment criteria: The requirement is fulfilled if additional devices and services can extend the architecture after configuring the initial setup.

3.4.3 Interoperability

Description: According to Wegner, interoperability is the property of a software component to communicate with other software components without a particular focus on environmental factors like programming languages, runtimes, or interfaces [88]. Technical progress makes this property an integral part of sustainable system development. EC members already use a variety of different devices with a wide range of communication technologies. That means there are devices based on modern communication technologies and devices based on older ones. A crucial part of the EMS is to communicate with this variety of devices. Therefore, system design requires a particular focus on the integration process of already existing and upcoming devices. In other words, a sustainable EMS architecture needs to support legacy devices and state-of-the-art protocols. The challenge is to design a future-proof EMS architecture to fulfill the changing demands of technological progress. Just like power grids, EMSs should also be compatible with future systems. This especially means that no strict limitations regarding communication abilities should be considered leading to an open and holistic interoperability approach.

Objective: The communication options of an EMS architecture should be developed holistically. This requirement aims to ensure that the architecture includes measures that foster interoperability.

Fulfillment criteria: The requirement is fulfilled if devices and services can communicate without adjustments. For legacy devices, the architecture must provide a procedure to make them compatible.

3.4.4 Usability

Description: EC members want to integrate new devices into an EMS. Therefore, the integration process of new devices is a common use case and requires special attention. On the one hand, there are technically proficient members that want to integrate and configure their devices themselves. On the other hand, non-technical proficient members want a simple way to enhance the integration process. An important part of an easy integration process is a smooth discovery process to establish a connection. For this reason, support of discovery and integration measures increases user satisfaction and acceptance. Another benefit of an easy integration process is the decrease in experimentation effort. For example, it is hard to assess the optimization potential of specific devices precisely.

An easy way to find out if a device impacts the overall electricity usage is to test a concrete constellation. A complicated integration procedure hinders this approach, and instead, a low-effort way to integrate devices is desirable.

Objective: The EMS architecture should enable discovery and integration mechanisms are crucial parts of the EMS.

Fulfillment criteria: The requirement is fulfilled if the EC architecture provides support for standard discovery and integration mechanisms with effort.

CHAPTER 4

Architecture

This chapter is about the presentation and description of a system architecture suitable for ECs. To assist the reader in understanding the concepts, a typical prosumer scenario is assumed. This includes a household with a local network to which energy-related devices are connected. The concrete setup and related assumptions are outlined in the first section. Moreover, the first section gives a technical overview of the problem domain. As a result, the overview identifies nine essential topics that a suitable architecture should cover to satisfy the requirements from Chapter 3. The second section presents a service-oriented architecture that is specifically designed with these topics in mind. Subsequently, every component of the service-oriented architecture is described in detail. This includes an explanation of the purpose and the interplay with other components.

4.1 Prosumer scenario

Figure 4.1 illustrates a prosumer scenario consisting of a router, a solar inverter, and an EV charging station. The inverter and the charging station are connected to the router and span a local network that is further connected to the Internet. In detail, the local network is provided by a set-up box from the ISP. The box provides basic network functions such as Dynamic Host Configuration Protocol (DHCP), Network Address Translation (NAT) and a firewall. In other words, it is a typical Internet router that connects IP-capable devices. This circumstance leads to the first restriction of the following approach, namely to only cover constellations where all devices are IP capable. Moreover, this thesis specializes in Transmission Control Protocol (TCP) and UDP traffic and neglects to investigate other protocols. However, the restriction to TCP and UDP should not be a problem in real-world scenarios, as alternative communication systems such as Long Range Wide Area Network (LoRaWAN) or ZigBee are usually already deployed with TCP capable gateways.

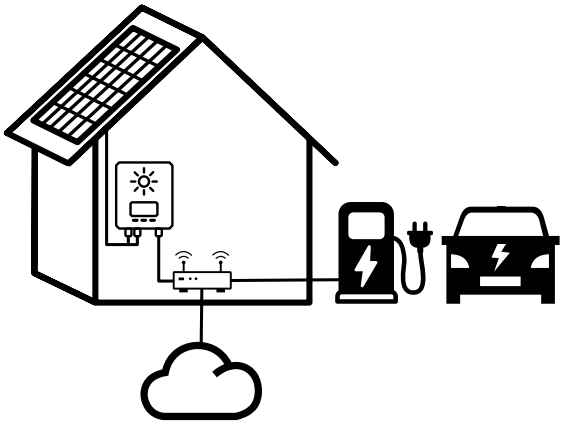


Figure 4.1: Typical prosumer scenario

Furthermore, a commitment to hardware-oriented protocols would not comply with the WoT principles. Novel systems should already be designed with the support of web technologies in mind to accelerate the divergence towards web technologies, as stated by Jara et al. [40]. TCP and UDP are the basis of most web protocols and hence, a suitable choice for further investigations. In contrast to hardware-oriented protocols, TCP and UDP provide additional benefits. On the one hand, they are widely used and supported by most consumer products. On the other hand, subsequent changes of web protocols only require an adjustment of software. This way, hardware incompatibility is decreased, and users can access their devices by simply updating software components. The initial scenario in Figure 4.1 may already give this advantage. If available, a user can access the inverter from a terminal device such as a smartphone. The prerequisite is that the user knows the inverter's IP address, and the smartphone can somehow interact with the inverter's interface. If this is not the case, the common basis of TCP and UDP does not require an adjustment of hardware. Instead, the user can install additional software that is capable of communicating with the inverter. In other words, the resulting problem domain only concerns the application layer of the IP suite [89].

However, this also means that a solution consists of software components that must be executed in a compatible runtime. In the simplest case, the software components can be run directly on the inverter or on the smartphone. Another possibility is to outsource the software execution to other devices or centrally locate it on the Internet router. Since manufacturers usually restrict foreign code execution, it is important to also consider other hardware environments. Moreover, the interaction scenarios in Chapter 3 involve TPSPs, which opens the door for a flexible distribution of software. In other words, parts of the solution may be outsourced and executed at the edge, fog, or cloud level, depending on the concrete requirements. As a result, the desired software components must be designed with code portability in mind. This situation extends the requirements of Chapter 3 and adds another aspect to the design of a system architecture.

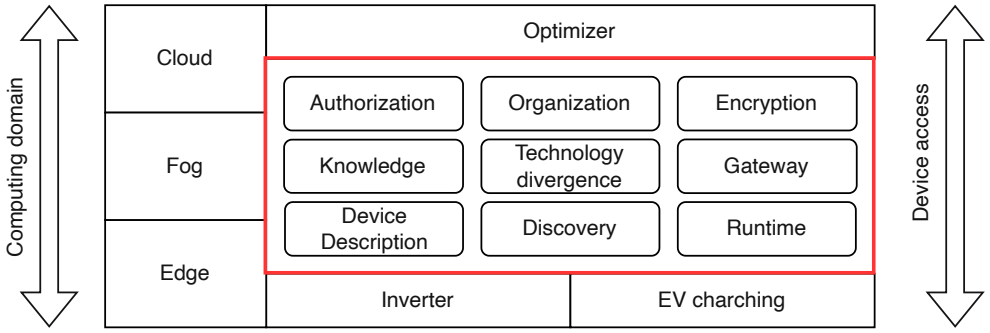


Figure 4.2: Illustration of the architecture problem domain and associated topics

4.2 Key aspects

Considering all the requirements, Figure 4.2 visualizes the key aspects of the architecture. The left side of the figure indicates the ascending level of centralized computing, namely edge, fog, and cloud computing. The right side visualizes the association of the initial prosumer scenario with the architecture problem domain. From bottom to top, the lowest level contains devices like inverters or EV charging stations. These devices are distributed over multiple households and do not know about each other. In contrast, the optimizer on the top of the figure preserves the full view of the EC and is therefore located at the cloud level. The architecture problem domain of this thesis is depicted in the middle of the figure and delimited by a red frame. It consists of nine topics that emerged through the previous collection of requirements. The following subsections describe each of these key aspects in more detail.

Technology divergence

Due to the diverse landscape of available communication technologies, it is not feasible for an EC architecture to support all of them. According to Jara et al., this is likely not necessary [40]. The authors set up the theory that technology divergence in the WoT context could happen the same way as during the spread of the Internet. If this is the case, it would suffice to only support a subset of the available technologies and still be able to participate in the emerging WoT communication network. In order to increase the chance of being compatible with this subset, it is necessary to think about suitable technologies in advance. In other words, before implementing the architecture is started, it is essential to assess the divergence of technology and determine a suitable subset of communication paradigms. The architecture should concentrate on this subset and take it as fixed points for developing new components. This means new services should be compatible with the selection of suitable technologies, and the system should initially be able to communicate without workarounds. Incompatible devices are made compatible by a gateway approach afterward.

Encryption

Local networks are often Demilitarized Zones (DMZs). In other words, there are few or no security mechanisms inside local networks, and every connected device can talk to every other device. This is especially problematic when not all network devices have encryption capabilities which in the worst case leads to clear text communication. Due to the high resource requirements of state-of-the-art encryption mechanisms, constrained devices cannot add another layer of security. Consequently, communication with external actors over the Internet runs at risk of being disclosed. For example, if the optimizer in Figure 4.1 reads the current electricity production from the inverter, adversaries may have access to the electricity production if communication takes place in cleartext. To mitigate this problem, the communication of devices outside the local network must be encrypted. Moreover, access from outside the network must be restricted. The aim is to reach a secure state-of-the-art network and simultaneously preserve the secure environment of the local network.

Authorization

Not every device functionality is intended to be used by every participant, even if the communication is encrypted. Therefore, device access of external parties must be restricted. A scope-based authorization approach is a suitable choice for this task. However, the selected authorization mechanism should be the same for all devices. Otherwise, each device would implement different credential and authorization mechanisms, which lead to compatibility issues. It is also possible to regulate access in a decentralized manner. For example, each device could provide individual authorization providers that are compatible with the architecture's authorization flow. This is especially the case if devices already implement suitable authorization mechanisms through the manufacturer side. In most cases, the user receives authorization requests from the respective provider and must agree or refuse the request accordingly.

Device Description

A common Device Description (DD) is necessary to know about a device's functionalities and interfaces. Looking at the prosumer scenario in Figure 4.1, the inverter will most likely format its sensor data in one way and the EV charging station in another. Similarly, the stated devices may provide different communication interfaces and interpret data differently. This initial situation makes it hard to gain an overview of the concrete constellation and further complicates the development of applications. The first part of the solution is to know the encodings, protocols, and interpretations that a device supports. If this information is available, the next step is to bring it into a format that users, devices, and services understand. In other words, a standard format for exchanging device information is needed. Further, the resulting DD should be understandable by humans and machines and subsequently be used by other parties. For example, online services could describe parts of their functionality in a DD to simulate device functionalities.

Discovery

Integrating new devices into the architecture should happen with little effort and adaptation overhead. For this reason, all devices must support a uniform discovery mechanism that provides information about implemented device functionalities. The purpose of describing and formatting these properties is already covered by the aspect of a standard DD. Discovery is responsible for receiving the DD of a device. To that end, novel devices and services must implement DD discovery according to a predefined discovery mechanism. Furthermore, it is necessary that devices that are not compatible with the standard discovery mechanism also have a way to be recognized in the network. Hence, the architecture must provide a workaround for legacy devices. This way, every device can utilize a discovery mechanism to establish a connection to other devices. However, it is not feasible that every device constantly triggers a discovery process or preserves its own state of DDs. Therefore, the architecture additionally needs a shared collection of DDs.

Knowledge

Introducing a standard format for describing devices only partly solves interoperability problems. Further, related parties must share a common understanding of DD content. To achieve this, DDs must extend their format with semantic meaning. In other words, keywords and properties of the DD format must lead to the same interpretation throughout communication parties. Moreover, DDs should be extensible by individual semantics. This means the DD format allows to specify properties or keywords beyond the initial specification. A suitable way to achieve this is to support semantic annotations. The use of semantic annotations covers the attachment of additional information and further enables extensibility in the future. Furthermore, semantic annotation can be used to model ontologic relations. This aspect is relevant for knowledge representation and connection. Knowledge-based systems or smart devices could interpret the meaning and set the information in context. Subsequently, modern approaches from ontology research could be used to relate different meanings and generate new knowledge.

Gateway

Technology most likely diverges towards novel protocols, encodings, and interpretations. Consequently, older devices are rarely compatible with emerging WoT scenarios. Utilizing gateways is a suitable approach to solve this problem. The gateway encapsulates another device and virtually provides its functionalities. In other words, the gateway substitutes the device's place in the communication chain and transforms messages into a compatible format. For example, Internet gateways are instances of gateway solutions that relay messages between the local network and the Internet. In this case, an Internet gateway abstracts the complexity of the local network and adjusts the communication accordingly. However, this example is about relaying messages and only takes care of simple transformations. More complicated gateway approaches transform each level of the communication stack and translate it to other technologies. Figure 4.3 illustrates the

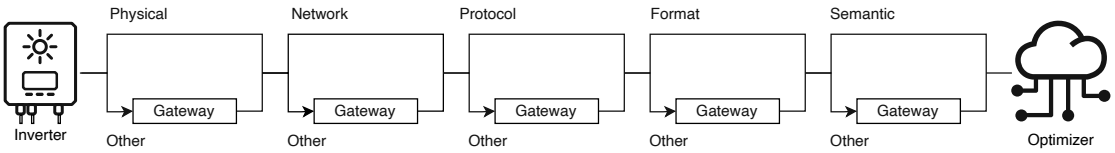


Figure 4.3: Different translation levels of energy related devices.

different levels of gateway transformations covering physical, network, protocol, format, or semantic incompatibility. The architecture needs a mechanism to support such gateway approaches to reach device compatibility.

Runtime

The architecture should support the dynamic instantiation of gateways. Therefore, software modules that contain gateway application code must be executed in a compatible runtime environment. In other words, if a particular gateway functionality is needed, the architecture must provide a mechanism to execute the gateway code. Furthermore, an execution runtime can also perform additional tasks. For example, it is conceivable that the optimizer distributes local applications for energy optimization. In the event of a communication loss, the local application could continue to run and synchronize when the connection is restored.

Organization

Ideally, device management happens automatically. Unfortunately, this is not possible due to the limited intelligence of the system. For example, the setup in Figure 4.1 can hardly decide if the optimizer is allowed to access the inverter or not. Consequently, user input is required. In this case, the user should be informed about the required interaction, and the required effort should be kept low. However, if desired, it should also be possible to manage the whole architecture on behalf of the user. This task requires the architecture to provide extensive configuration for experienced users but well-defined default values for ordinary ones that rarely want to touch the system.

4.3 Design

Figure 4.4 illustrates an architecture design that specifically targets the key aspects of the previous section. In detail, the design follows the concept of outsourcing key aspects to individual services. This strategy extends the initial prosumer scenario in Section 4.1 with five additional services. Before every service is explained in detail, the following bullet points outline general conditions that must be fulfilled for the architecture to work:

- Services' software components are appropriately executed. This assumption targets the runtime for hosting the respective services. As stated in the prosumer scenario,

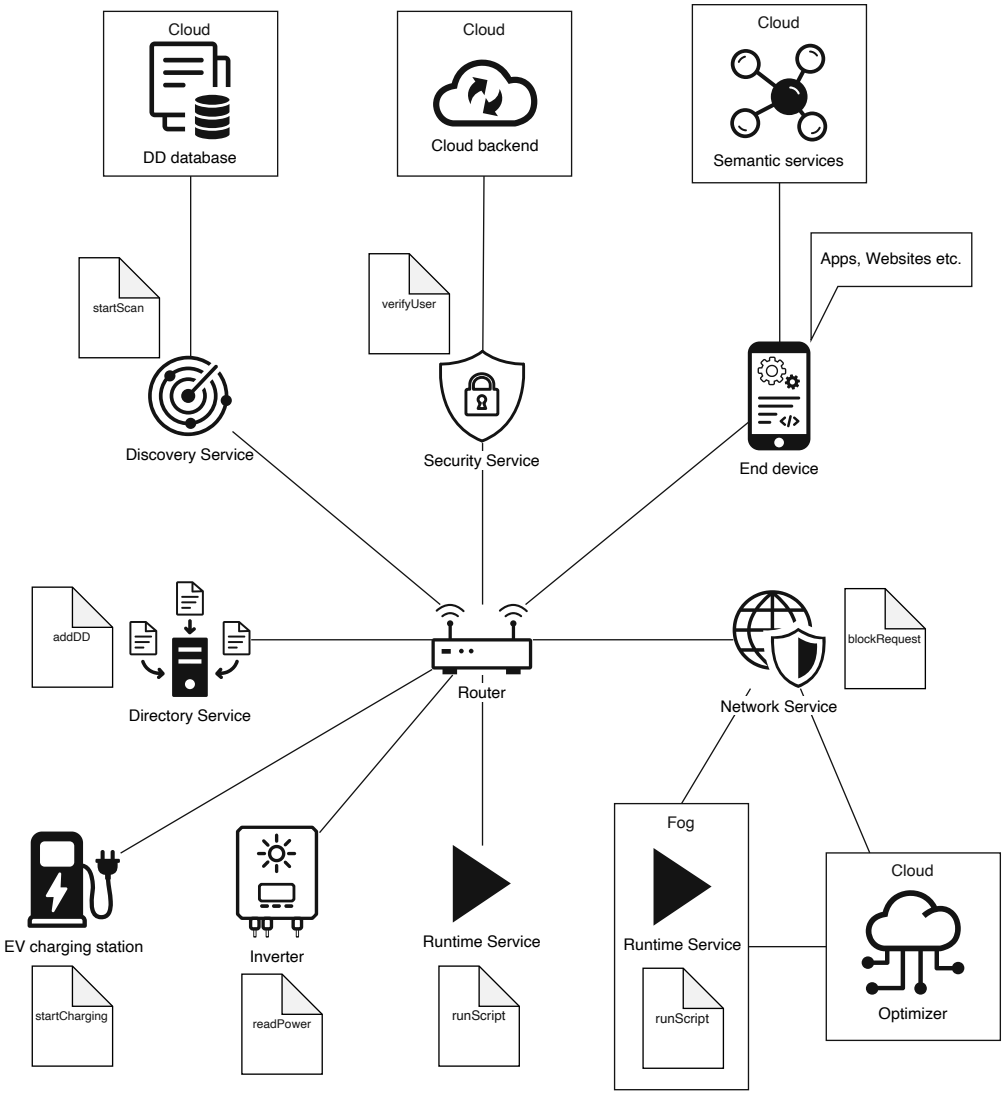


Figure 4.4: Overall architecture design.

the software may be executed on the ISP gateway. Otherwise, alternative hardware or a virtualized environment is required.

- Services are compatible with the determined technology assessment. Because the five services are newly introduced in the scenario, they should at least support a subset of the determined technologies. The aim is that communication can occur without compatibility issues between the newly introduced components.

- Services support the determined authorization mechanism such that access control can take place in a uniform manner. If technology selection determines multiple authorization mechanisms, at least one must be supported.

- Services support the determined discovery mechanism. This assumption is similar to the last one. Moreover, older devices depend on the discovery mechanisms of novel devices and services. Hence, at least new devices must implement a compatible discovery mechanism.

- Services support the predefined DD structure, format, semantics, and related document properties. This assumption also includes that a service must self-describe its affordances and functionalities with a DD document. At least new services should know about each other and understand each other's functionalities.

- Configuration and organization functionalities are also part of the DD. End devices and other services can understand these capabilities and implement suitable mechanisms for management tasks.

- External services provide transformation and matching services for unknown ontologies. Conceivable implementations are knowledge-based systems in the cloud or manual translation mechanisms that involve the user, as outlined by Novo et al. [90]. Moreover, such cases may be covered by an ontological gateway approach.

- User end devices such as smartphones are compatible with the technology choice and further understand DD and related information. They can automatically provide a compatible User Interface (UI) for system insights and control. Moreover, they can access external services for translation if they do not understand relevant information.

Additionally, it must be stated that not every service is necessarily required in every scenario. If all devices are already compatible with each other, there is likely no reason to introduce the architecture's full overhead. However, the benefit of the architecture is to be prepared for legacy devices. If parts of the architecture are already present, the extension to full compatibility is less effort.

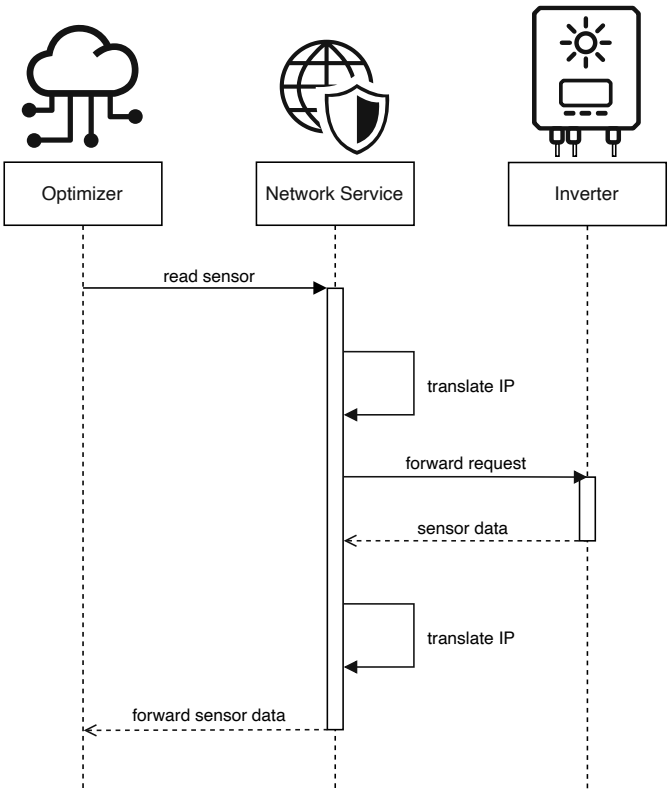


Figure 4.5: Network Service interaction exemplified in a data request scenario.

4.4 Network Service

In nearly every EC scenario, the Network Service is the central part that establishes and controls the communication between external parties and local devices. For example, if the optimizer wants to read the actual electricity production from the inverter, the request first reaches the Network Service, as visualized in Figure 4.5. Subsequently, the Network Service checks the access rights and translates the request into local network messages. Next, the inverter answers the request and sends back the production data to the Network Service, which in turn answers the optimizer's request by relaying the message. In other words, the Network Service provides similar functionalities as an Internet router that translates messages from Wide Area Network (WAN) to LAN. Moreover, the Network Service has a built-in firewall to protect legacy devices inside the network. Indeed, this scope first looks like it does not differ from an Internet router. The following outlines why the Network Service needs to be separate.

In general, there are multiple ways to exchange messages on the Internet. Besides the different communication paradigms like polling or pushing, an essential factor is who started the communication. In the prosumer scenario in Figure 4.1, this can either happen from inside the local network or from outside. For example, if the inverter pushes

data to the optimizer, the communication is initiated from inside the local network. The optimizer externally listens for the messages and receives the actual sensor data. This procedure is commonly implemented in IoT and also allowed by most of the Internet gateways. The resulting problems are twofold.

First, the inverter needs to know the related endpoints to which the sensor data should be pushed. If these endpoints change regularly, the inverter needs to update its configuration. Similarly, the location of endpoints is a problem when polling data. Because most households get a dynamic WAN IP, the public Internet address changes from time to time. Consequently, the optimizer does not know the updated address of the inverter and therefore is unable to initiate a polling process.

Second, due to the default configuration of most firewalls, the optimizer cannot start communication with the inverter by itself. In the case of pushing the data to the optimizer, this is only a problem when the optimizer wants to answer requests or inform the respective devices about changes. However, the polling scenario does not work with a strict firewall. In other words, the optimizer cannot initiate the connection that is necessary to transfer the respective request. Due to the publicity of the Internet, the default security configuration of Internet routers blocks these requests from outside and discards the messages. In most cases, these security settings are justified to protect local networks from adversaries.

However, in both cases, workarounds exist. The first workaround concerns the configuration of the Internet router firewall and dynamic IPs. Common approaches solve these issues by opening TCP or UDP ports on the Internet router. Nevertheless, there are scenarios where port forwarding is not possible or comes with high effort, despite the existence of configuration-facilitating protocols such as Universal Plug and Play (UPnP). For example, a Small and Medium Enterprise (SME) could communicate over a hosted network that does not allow the opening of ports and directly resolves into an intranet. In this case, UPnP is likely not implemented, and the structure of the network consists of multiple gateway boxes with their own firewall rules. Apart from that, in the context of households, it is bad security practice to open a port without providing appropriate security mechanisms. The second problem, which concerns dynamic IP addresses, is less complicated to solve. In most cases, some external providers, such as operators of Domain Name System (DNS) servers, provide information about the actual IP of the internet router. This approach requires that some device inside the local network updates the DNS or IP entry accordingly when the dynamic IP address changes.

Creating bidirectional streams is another option for the stated problems, which usually requires no additional configuration and still offers a high level of security. In this case, the inverter initiates the connection to the optimizer and opens a secure tunnel. Afterward, both parties can use the tunnel to send messages. This way, the connection is established from inside the local network and does not require ports opening. However, in the context of interoperability, novel streaming protocols like WebSockets [91] are rarely implemented in IoT. Consequently, making devices compatible with such protocols comes with transformation efforts. For example, the Network Service could take care

of the transformation of local messages onto a streaming protocol like WebSockets. For this purpose, the Network Service must map each device to a corresponding tunnel with the optimizer. However, the structure of different protocols and formats complicates this task. Additionally, dependency on the streaming protocols increases, and the Network Service must provide complex application logic to support network functionalities like discovery.

For these reasons, the Network Service must provide a higher-level streaming protocol and abstract the network interface. Network virtualization is a suitable approach to solve this problem. Virtualized networking generally abstracts the network traffic and maps it from standard interfaces to an overlay streaming protocol. The underlying devices do not notice that the network is virtualized because the interfaces stay the same. Therefore, it is possible to extend the regular network with a virtualized overlay network that can communicate without the adjustment of local devices. In the prosumer scenario, the Network Service can be seen as a gateway to another isolated network in which the optimizer is located. Due to the visibility and control properties of this network, participation can be restricted, and the user can control network access. In short, the Network Service provides an overlay network that extends the local network. This network is controllable, and due to the utilization of streams, secure tunnels can be established from inside the local network. With this solution, port forwarding is prevented in most cases, and network interoperability is provided. The only restriction is that external actors, such as the optimizer, require support for the virtualized network technology. Even though this is a dependency on the architecture, the switch of network virtualization technology is decoupled from the primary EC scenario. Moreover, the Network Service is a novel component in the system, which on the one hand, can be designed to be updated and, on the other hand, can support multiple virtualized networks. The following bullet points summarize the stated aspects:

- Virtualized networks mostly work without special firewall configurations. This means less configuration effort and increased security.
- Virtualized networks can provide encrypted tunnels, which lead to increased security.
- Virtualized networks can provide access control mechanisms. Due to the requirement of a gateway, also additional firewall functionality can be implemented.
- Virtualized networks can bypass dynamic IPs and provide network-based identification mechanisms. This leads to less configuration overhead.
- Virtualized networks can simulate broadcasting inside the network. This allows the utilization of local discovery mechanisms that are not applicable over the Internet.

However, the Network Service should act as a router that only forwards messages from one network to another in the regular case. For simplicity reasons, the following sections neglect to indicate the Network Service in every scenario explicitly. In other words,

subsequent scenarios assume that communication with external parties automatically happens over the Network Service without explicit mention.

4.5 Security Service

The Security Service is primarily responsible for the administration of access rights. When considering the prosumer scenario in Figure 4.1, a typical operation is that a user allows the optimizer to access the inverter. For this to make sense, device access must be restricted in advance. The least-privilege principle is generally a suitable way of defining access rights in the architecture. This means third-party access to devices is restricted by default, and permission grants take place in a prior, explicit, and minimal way. However, this strategy is not always possible. Devices that do not provide authorization mechanisms cannot adhere to authorization principles. Moreover, devices that implement legacy authorization mechanisms have to be adopted. This thesis tackles the stated problems by outsourcing authorization concerns to the Security Service. In general, the Security Service distinguishes between compatible and legacy devices. The following paragraphs describe the two scenarios and outline the respective authorization procedure. For the sake of simplicity, it is assumed that the optimizer already knows about the interface of the energy-related devices and has access to them. Concrete details about discovery aspects are part of the Discovery Service in Section 4.6.

Compatible devices

Ideally, a device such as an EV charging station already implements a web-conform interface like HTTP with a state-of-the-art authorization mechanism like OAuth 2.0 [92]. If this is the case, the optimizer requests an authorization token before data retrieval starts. Figure 4.6 visualizes a token-based authorization procedure. First, the optimizer accesses the EV charging station's interface and requests its DD through a compatible procedure like well-known URIs. The obtained DD contains information about the authorization procedure and indicates an authorization endpoint, which in this case resolves to the Security Service. Next, the optimizer requests an access token from the Security Service. If there are no default authorization rules configured, the Security Service stalls the request and notifies the user to give permission. Consequently, the optimizer must wait until the request is approved. After the user permits access, the Security Service returns the access token to the optimizer. From now on, the optimizer can access the EV charging station by attaching the related token to every request.

Workaround for legacy devices

In the worse case, the device provides legacy or no authorization mechanisms. Here, it is necessary to consider the dimension of possible harm. For example, if the inverter's interface only provides a single unsecured endpoint with reading capabilities, it is most likely not necessary to restrict access further. However, it is also conceivable that the inverter provides sensitive write endpoints that must be secured. In this case, access

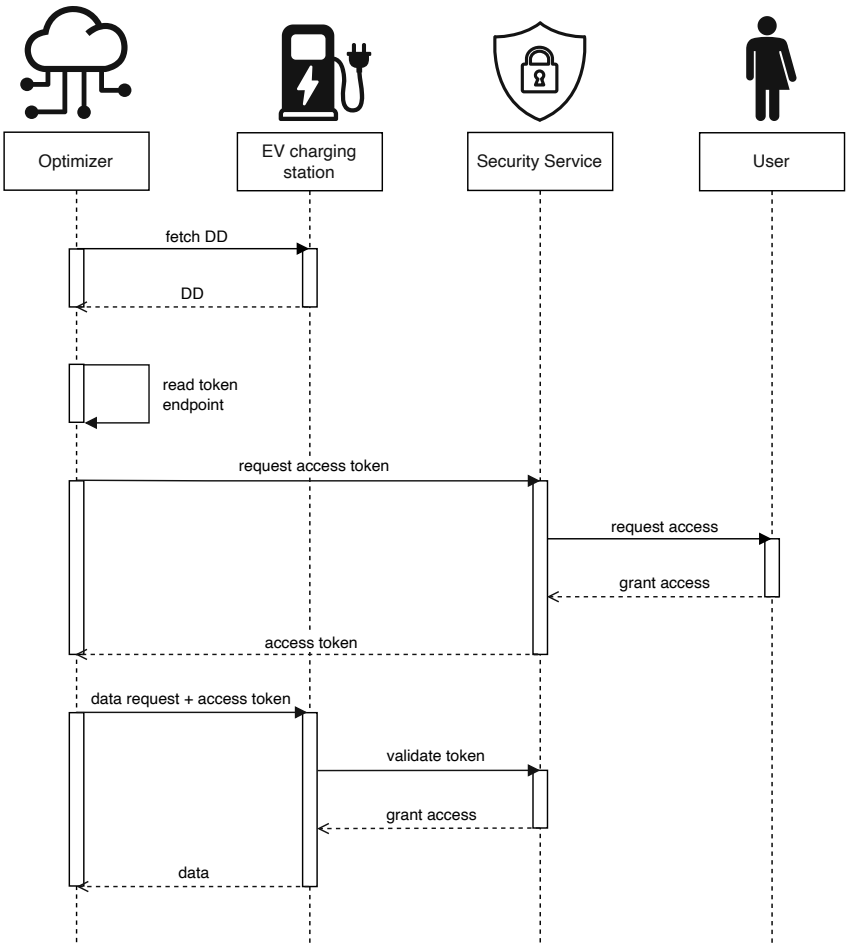


Figure 4.6: Token based authorization procedure for a compatible EV charging station.

to read and write must be separated through fine-grained access control. Figure 4.7 visualizes the process of adding token-based authorization capabilities to a legacy device. In detail, the scenario outlines an extended authorization process for an inverter that does not provide any authorization mechanism. The solution is to encapsulate the inverter's interface by sending requests through a gateway. In the simplest form, the gateway only adds authorization capabilities like some HTTP proxies do. This approach comes with little overhead and is applicable if the charging station already provides an HTTP API. Indeed, this is a simple case, and the overhead depends on the individual situation. Further, it is not specified in detail how the gateway handles the separation of reading and write access. These aspects are considered to be implementation details part of Chapter 5.

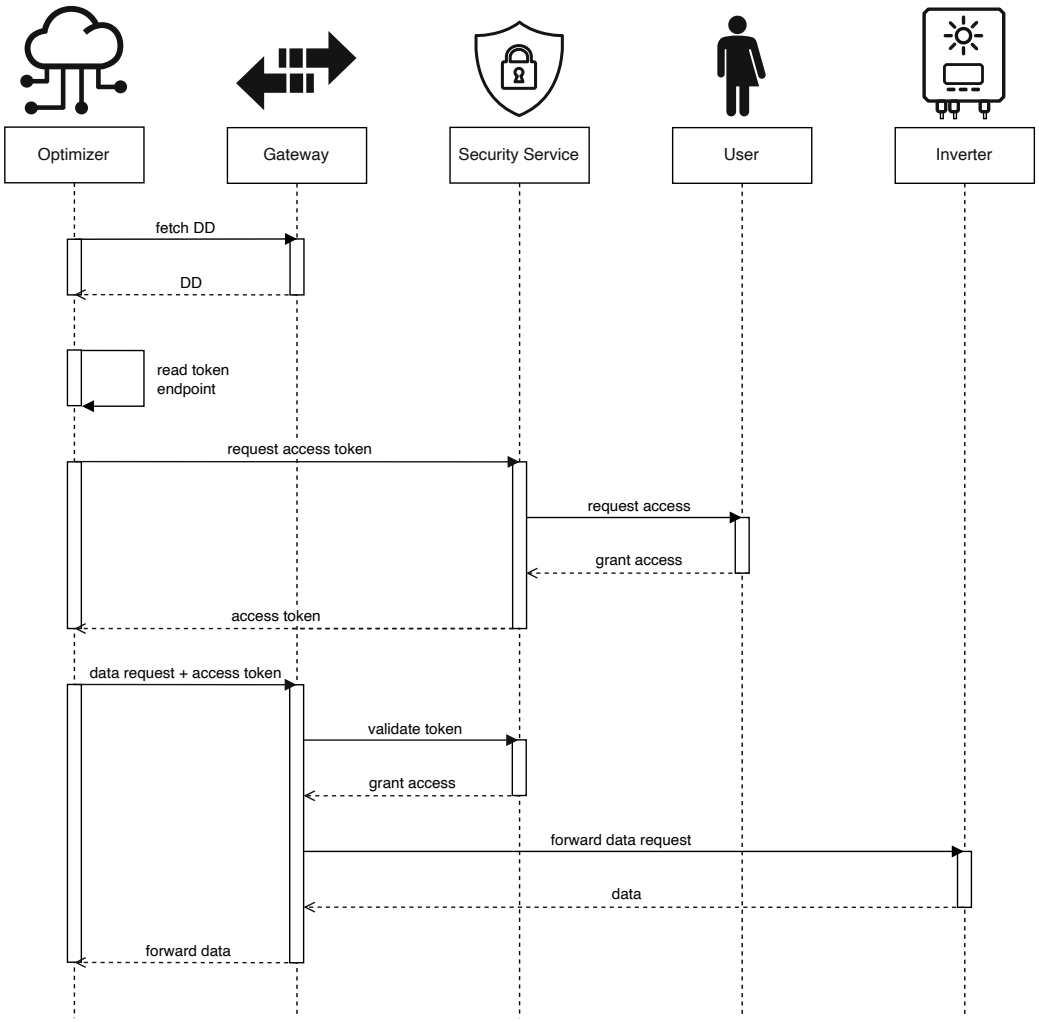


Figure 4.7: Token based authorization procedure for a legacy inverter.

4.6 Discovery Service

If devices do not implement the determined discovery mechanism, the Discovery Service comes into play. In general, there are no restrictions on what techniques are internally used in the service. The aim is to discover legacy devices and provide related DDs. Figure 4.8 illustrates an example where a legacy device is identified by scanning the local network. For the sake of simplicity, the example neglects authorization procedures and associated interactions with the Security Service. However, the example starts by connecting the inverter to the local network. Next, the inverter requests an IP address from the set-up box via DHCP. Depending on the concrete constellation of the network, the Discovery Service gets notified about the new device and starts a device scan. Alternatively, it is possible to start network scans on a regular basis or provide

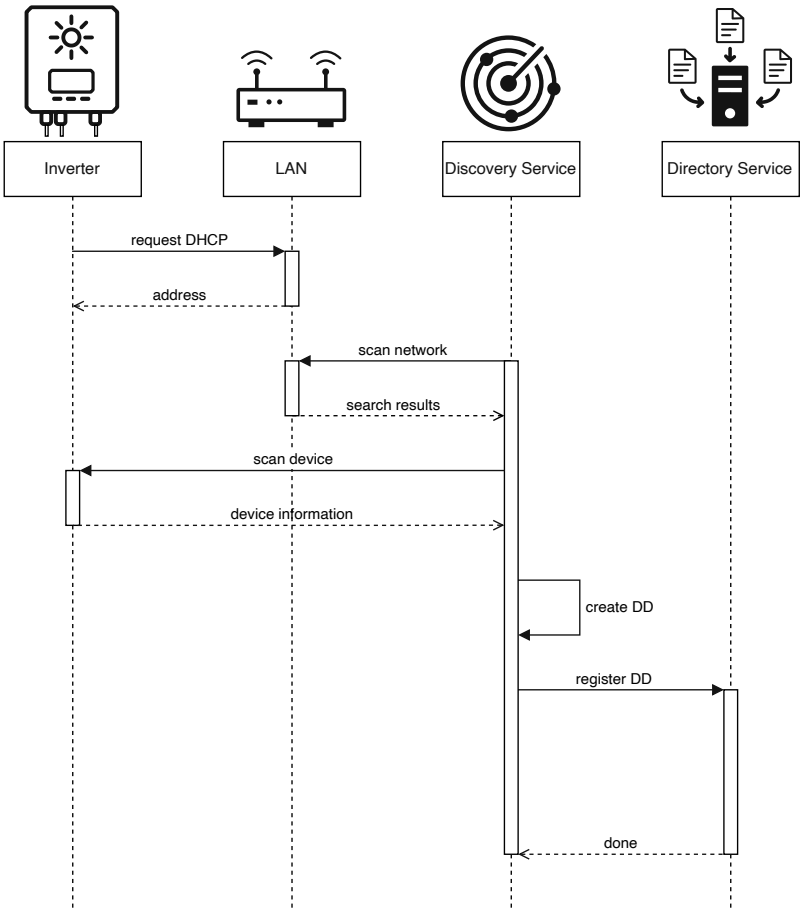


Figure 4.8: Onboarding procedure of a legacy inverter.

endpoints through which users can trigger network scans manually.

When the connection is established, the Discovery Service tries several techniques that could identify the device. A conceivable approach is to send a probable protocol request and check if it is successful. Another way of device identification is to analyze its traffic, as proposed by Aksoy et al. [93]. If the identification is successful, the Discovery Service creates a suitable DD. The relevant information for device identification and DD creation is not further specified. For example, the Discovery Service could be connected to external cloud services, as illustrated in Figure 4.4. This way, the application logic is outsourced to external services, which return the corresponding DD. The DD is subsequently registered in the Directory Service, and from this moment, also other devices and services know about the existence of the legacy device. Indeed, this procedure most likely does not cover all legacy devices. Alternatively, the Discovery Service can support the device onboarding and retrieve missing information through user inputs. However, in the context of the architecture, the aim is to concentrate discovery-related tasks in the Discovery Service, which enables a flexible architecture for additional discovery processes.

However, additional concerns must be determined in the Discovery Service. One of them is the problem of identification. Even though novel devices already provide identification mechanisms, legacy devices most likely do not. For this reason, it must be managed how legacy devices get a standard Identifier (ID) that can be uniquely be used to identify the device in a related DD document. A simple workaround for this problem is to utilize randomly generated IDs. Nonetheless, caution must be taken when distributing randomized IDs in dynamic environments. Besides the entropy properties of the ID generation algorithms, it is also necessary to think about reusability. For example, if the inverter gets its IP address via DHCP, it could be the case that the address changes from time to time. Due to the new IP address, the inverter may look like a new device to the network. Consequently, the Discovery Service triggers a discovery process and generates a DD including a new ID. In this case, the other devices may not notice that the inverter was already part of the architecture. Consequently, properties like access control are restricted, and settings may be reset. The Directory Service and the architecture implementation must take care of such cases and find a suitable approach for the respective constellation.

4.7 Directory Service

The Directory Service collects available DDs and maintains device information. This purpose mainly concerns the actors' need to get an overview of available devices. For example, a user in the prosumer scenario in Figure 4.1 wants to uniformly access information about the inverter and the EV charging station. Therefore, the architecture requires a directory structure that can be used to access and organize device information. This structure is also a suitable place for storing and managing the previously determined DDs. Reasons for this are multifaceted. First, splitting device information and related descriptions can lead to inconsistencies during updates. Second, maintaining a combined state of device information and DDs facilitates management tasks. Third, serving device information and DDs from the same service reduces communication overhead. This thesis covers the stated aspects by introducing a directory service that manages device information and related DDs accordingly.

The underlying data primarily involves property relationships. For example, parts of the device information could represent in which room a particular device operates. In this case, the information stands for the relationship between house and device. Triple stores are therefore a suitable choice for maintaining and managing device information. Also, lightweight implementations with key-value stores are conceivable. This alternative is especially appropriate if a directory service's computing power does not suffice for running a triple store. However, the internal structure of the directory service is an implementation detail. The central aspect is the interface that provides access to device information and DDs. For this purpose, the first step is to define a data exchange format. Currently, devices and services are only required to understand the DD format. Moreover, DDs are semantically extensible and can therefore represent individual relationships. These features make DDs a suitable choice for data representation and exchange. As a

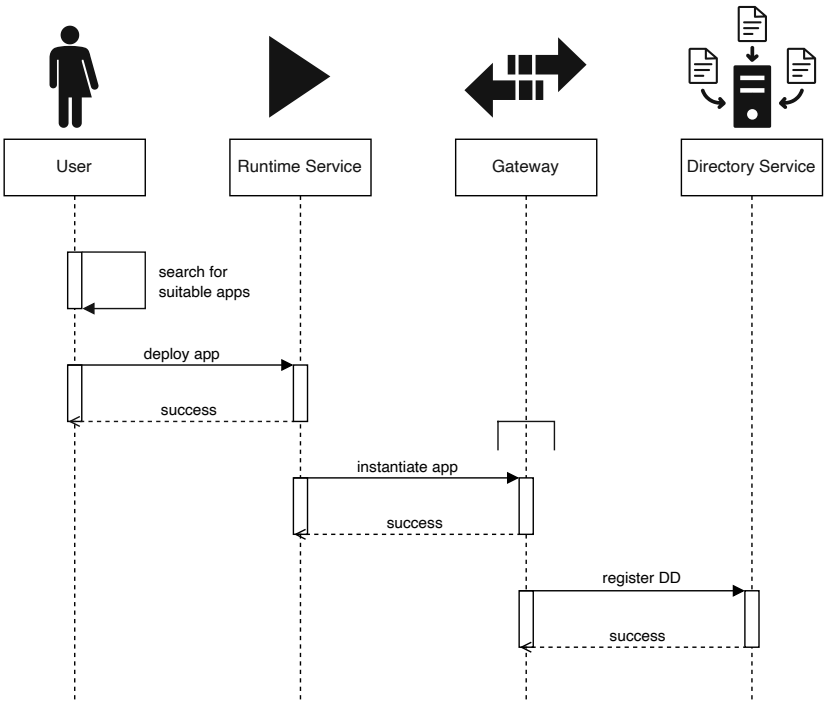


Figure 4.9: Gateway instantiation procedure.

result, the directory service's interface only needs to support a single exchange format. Communication and access procedure of the DDs are once again specified in the Directory Service's DDs.

4.8 Runtime Service

The Runtime Service comes into play if an extension in the architecture is necessary. Conceivable tasks reach from executing self-contained apps that control energy-related devices to parts of external services that facilitate data access. In other words, the introduction of such a service can be used for a broad range of applications. However, the Runtime Service's job for this architecture is to provide interoperability with gateways. Therefore, Figure 4.9 illustrates the process of how a gateway is instantiated. For the sake of simplicity, security and authorization-related concerns are neglected in this example.

The scenario's background is a compatibility issue, and the user is already aware of the problem. In the first step, the user searches for available apps from third-party providers. If a suitable and trustable gateway code is found, the user can deploy the software to the Runtime Service, which further initiates the gateway instantiation. After the gateway is instanciated, the user configures the legacy device that should be encapsulated, and the gateway registers its DD at the Directory Service. From this moment on, other devices can recognize the gateway as an individual device and talk to it. This process can also be

automated. Due to the semantic extension in DDs, a novel device could understand what incompatibility issues are encountered. With this information, the device could search for a compatible gateway and instantiates the needed gateway application on its own.

CHAPTER 5

Implementation

This chapter describes a real-world implementation of the proposed architecture from Section 4.3. The aim is to show the architecture's feasibility and provide a test setup for evaluation. For this purpose, the prosumer scenario from the last chapter is concretized and simulated in a virtual environment. The implementation process starts with assessing technology and choosing a subset to which the architecture's services are compatible. Next, the subset of technology is used to implement the architecture's services and construct the associated DDs. Finally, the interplay of the services is exemplified in two interaction scenarios that demonstrates integration and interoperability properties.

5.1 Testbed environment

The background story of the implementation is a concrete instance of the prosumer scenario in Section 4.1. In detail, the implementation outlines an exemplified scenario with a legacy inverter and a fully compatible EV charging station. This situation is simulated by mocking every device and service in a virtual Docker environment. The resulting testbed consists of eight containers that are executed on standard computing hardware. Moreover, the Docker environment is used to simulate a prosumer's network situation. For this purpose, the containers are split into two networks by utilizing the Docker bridge driver. Figure 5.1 illustrates the concrete constellation. The left side of the figure represents the local network with the subnet `10.10.0.0/16`. The right side represents a private network in the optimizer's cloud with the subnet `10.20.0.0/16`. Every subnet is configured to act as the respective network in the prosumer scenario. This means the optimizer and the local devices are split in connection and cannot directly communicate with each other. However, both subnets are connected to the Internet. Additionally, all devices and services get static IP addresses in advance. The main reason for this policy is the avoidance of dynamic environments, which could lead to configuration issues. If needed, self-reconfiguring measures can be added in a more

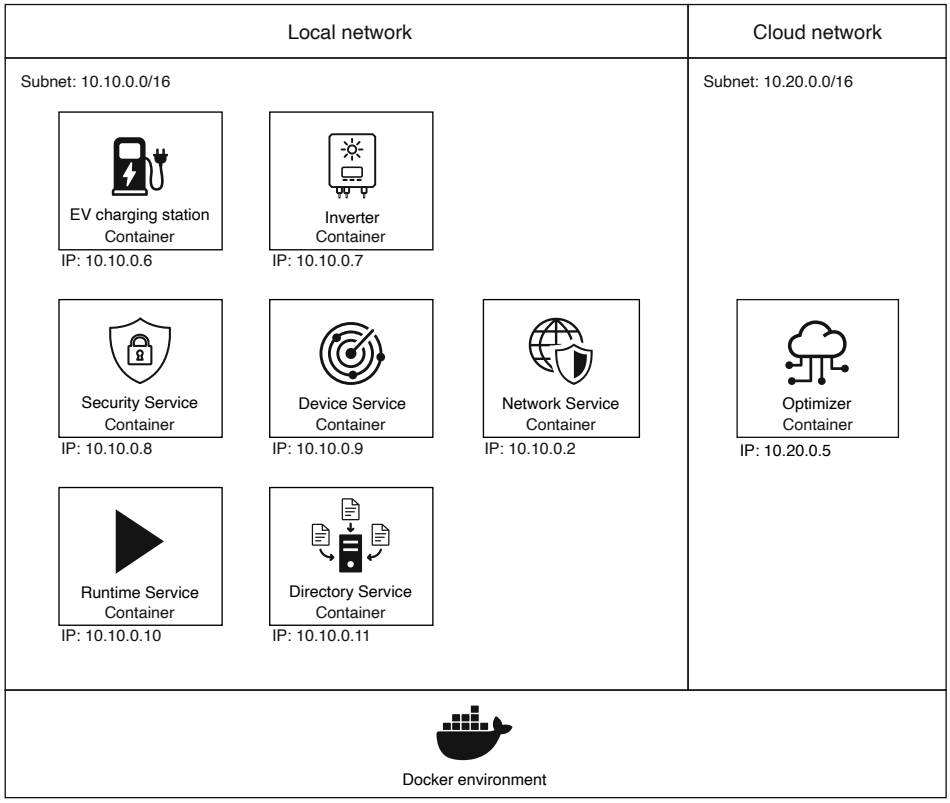


Figure 5.1: Docker based testbed.

extensive implementation. In this implementation, it is assumed that users are able to take care of IP configurations themselves.

5.2 Technology choice

The key aspects in Section 4.2 start with the selection of a suitable subset of technologies. Therefore, the first step of the implementation is to assess current trends and choose a subset of suitable technologies. The criteria for this selection process are twofold. On the one hand, the selected technologies must suit the prosumer scenario and its consumer-oriented environment. On the other hand, selected technologies must conform with the WoT concepts and prepare for future scenarios. That said, the following sections compare suitable technologies and justify the respective choices. In any case, the following selection represents only an exemplary assessment and therefore does not claim completeness.

5.2.1 Device Description

The problem with DDs is understanding their formats and exchange procedures. For this reason, standards like Sensor Model Language (SensorML) [76], OMA Lightweight M2M

(LwM2M) [94] or one Machine to Machine (oneM2M) [95] specified individual information models for the representation of device and communication functionalities. However, the problem with these approaches is that they replace parts of the communication stack. In other words, they introduce new specifications to which devices must adhere. This strategy is especially problematic with legacy devices that may communicate via older technology stacks. The W3C WoT standard [45] addresses this problem by introducing a DD on a higher level. The advantage of this approach is the decoupling of the description and the actual communication procedure. This way, the WoT standard intentionally bypasses restrictive dependencies of specific communication protocols. As a result, the so-called WoT TDs are also compatible with older devices and can be defined after the initial deployment [46].

Likewise, the definition of a higher-level DD could be implemented by using other technologies. For example, the utilization of knowledge representation formats such as RDF [96] leads to similar capabilities as provided by TDs. In combination with suitable ontologies like Smart Applications Reference (SAREF) [97] or Semantic Sensor Network (SSN) [98], it is possible to model essential IoT devices and related properties. However, this approach was discarded, since RDF is not intended for IoT use. In contrast, the TD format was specifically developed for IoT and is anyway extensible by semantic annotations. This means, information from external ontologies like SAREF or SSN can be included into TDs similar to the representation in RDF. Moreover, the freedom of RDF may lead to a misunderstanding of specific communication terms. In other words, the reliance on RDF would only solve a formatting issue but still allow different vocabularies for the same meaning. In contrast, W3C standardized the needed vocabulary for device communication in the WoT standard. Due to the involvement of global manufacturing companies like Siemens or Huawei, it is likely, that the WoT vocabulary is adopted by a significant part of IoT devices. These facts make the TD format a suitable choice for describing devices and is therefore selected for further use.

5.2.2 Communication

Promising and established protocols in IoT are MQTT [74], CoAP [73] and HTTP [69]. In the context of web development, especially HTTP proves to be popular and established as a de-facto standard for web applications and API development. In contrast, MQTT and CoAP are designed for constrained environments and therefore provide lighter protocols than HTTP. However, IoT devices continuously increase in computing power and novel device can already handle the HTTP overhead. Additionally, HTTP provides integrated support for insecure and secure communication over Transport Layer Security (TLS). Even though CoAP provides similar properties, HTTP is a de-facto standard in the web. For these reasons, HTTP is further selected as target protocol. If CoAP largely establishes, the similarity of CoAP and HTTP allows protocol transformation with little overhead. Therefore, the selection of HTTP provides a solid foundation for current use and additionally prepares for future developments.

The next point in the context of communication is to select a standard data exchange

format. JSON [99] and XML [100] are encodings that are commonly used in accordance with HTTP. Both of these formats are text-based and hence, also human-readable. However, due to the overhead of XML, JSON seems more suitable for communication purposes in the WoT. Another promising candidate in the context of data encoding is Concise Binary Object Representation (CBOR) [101]. In short, CBOR is designed as a binary drop-in replacement for JSON to reduce its text-based overhead. However, due to the binary format, CBOR is not human-readable. In addition, there is no standardized query language yet. These facts discourage the adoption of CBOR, and hence, the encoding did not reach widespread use until now. Nevertheless, if CBOR establishes, it is little effort to extend devices and services in a way to support CBOR. Until then, GZIP [102] or Brotli [103] compression of JSON also achieve comparable results. To summarize, the tradeoff is acceptance and compatibility vs. performance. Due to the interoperability focus of this work, JSON is selected for further use.

5.2.3 Discovery

Private networks allow to perform local discovery measures. Suitable protocols for this purpose are UPnP [104], Simple Service Discovery Protocol (SSDP) [105], Domain Name System Service Discovery (DNS-SD) [106] and mDNS [55]. In fact, mDNS and DNS-SD were coordinately developed by IETF for the purpose of local service discovery. In contrast, SSDP works with UPnP, but the standards are not strictly standardized for joint use. Additionally, the primary focus of UPnP is the application layer which also provides application related features. Parts of these features are already contained in the WoT TD format. This overhead makes mDNS in combination with DNS-SD a simple and straightforward solution.

Hence, novel devices and services in this implementation must provide mDNS with DNS-SD for discovery. However, even though this selection enables proper device discovery, it must also be specified how the discovery mechanism reveals information about the device or service. In other words, the outcome of the service discovery process must lead to a TD. Therefore, it is determined that capable things must provide an additional endpoint that follows the well-known URI scheme [57]. This approach is also covered in the WoT discovery document [51]. The document states that compatible devices must reveal to the subpath `/.well-known/wot-thing-description` and therefore fits with the previous technology choice of HTTP.

5.2.4 Authorization

Authorization is a complex topic in local environments. The reason is that an authorization's initial configuration and management effort does likely not pay off on a local level. Therefore, most of the IoT devices already bring their own authorization mechanisms. The problem is that there are few standards for authorizing device access. One standard in the context of the web is OAuth 2.0, which is specified by IETF [92]. This standard covers parts of the authorization process and handles detailed use cases by the definition

of OAuth 2.0 extensions. In contrast to approaches where every device provides its own authorization mechanism, OAuth 2.0 enables the central management of access grant and validation. Additionally, it allows the authorization server to be externally managed by a third party. This approach is especially beneficial if users are unwilling to operate an independent authorization server. However, the authorization flows in OAuth 2.0 are not suitable for the prosumer scenario. The reason is OAuth 2.0's concept of clients and the way the credentials must be provided. For example, OAuth 2.0 does not directly address user interaction when a device wants access to another device. The recommended OAuth 2.0 flow for this scenario is named *Client Credentials Grant* and assumes the inquiring device already has the credentials. Therefore, this implementation only relies on concepts of the OAuth 2.0 standard but defines an individual authorization procedure based on JSON Web Tokens (JWTs) [107]. Even though this does not solve interoperability problems in the context of authorization, at least it specifies the format of modeling authorization information.

Despite this, the utilization of JWT provides a framework for defining token claims. On the one hand, this includes registered claims such as `aud` (audience), `iss` (issuer), `exp` (expiration time) and `sub` (subject). These claims are defined in the *IANA "JSON Web Token Claims" registry for JWT Claim Names* [107] and, therefore, can be considered publicly understandable. In this implementation, the registered claims are extended by a private claim named `scopes`. In short, the `scopes` claim is an array and represents the access rights of a JWT. Novel devices and services in the architecture are required to understand the `scopes` claim and implement their authorization mechanisms accordingly. The concrete items that must be included in the scopes claim to access a device's endpoints are specified in its TD. Exemplary scopes that are used in this implementation are `observe`, `control`, or `admin`. The advantage of this approach is that JWTs are self-contained and can be validated offline. This way, the authorization server can be outsourced to external parties, and offline devices can verify the token validity if a common certificate is trusted in advance. The last thing to mention is how the token claims inside JWTs are signed. With the possibility of specifying the algorithm inside of JWTs or the security schemes in TDs, the security mechanism provides two ways for variable use of signing algorithms. However, due to convenience reasons, this implementation favors the *RS256 (RSASSA-PKCS1-v1_5 with SHA-256 hash algorithm)* variant, which is recommended by IETF [107].

5.2.5 Virtual Private Network

In the context of VPNs, OpenVPN [108] and IKEv2/IPSec [109] are still secure solutions and supported by a significant amount of platforms. However, novel P2P protocols are likely more performant and secure. One such solution is the WireGuard protocol [110] by Jason A. Donenfeld. A key aspect of this protocol is the lightweight encryption mechanism ChaCha [111], which in contrast to OpenVPN only causes little communication overhead. Besides the superiority in performance, WireGuard also opens the additional property of P2P communication. As a result, the communication to external parties is less error

prone. However, WireGuard requires more configuration effort and is less flexible in the basic version. Tailscale [112] addresses this problem by encapsulating WireGuard into a user-friendly network application. In other words, Tailscale abstracts the interaction with the WireGuard daemon and the associated setup. This way, the only requirement on the Tailscale clients [113] is to install the related software that automatically configures the virtual network adapters.

5.2.6 Code execution and portability

This step of the selection process primarily concerns the runtime environment for extending the architecture's basic functionalities. A proven technique for such intentions is to use Java and provide extensibility through the Open Services Gateway initiative (OSGi) framework [114]. This technology choice is used by several gateway solutions like the energy gateway OpenEMS [115]. However, the resource intensity of Java Virtual Machines (VMs) is only partly suitable in the IoT domain. Therefore, primarily machine-oriented languages such as C are used for constrained devices. The reason is that machine-oriented languages can be compiled explicitly for the respective processor architectures of the devices and thus can max out hardware constellations with little overhead. The problem with this approach is the high level of dependency and low degree of code portability.

WebAssembly is meant to find a middle ground between hardware orientation and code portability [116]. The essential aspects of WebAssembly are twofold. On the one hand, compiled code is executable in various runtimes for different purposes. On the other hand, it supports lightweight runtimes and achieves machine-near performance. Originally, WebAssembly was conceived for better performance in web browsers, but at present, there are already browser-independent runtimes that can even be deployed to Microcontrollers (MCUs). Extensions like WebAssembly System Interface (WASI) [117] provide a standardized way to communicate with system interfaces such as network capabilities or file system access. Due to the proximity to the web, also other novel technologies are compatible with WebAssembly. Especially web browsers or other JavaScript (JS) environments like NodeJS or Deno already provide compatible runtime environments. Furthermore, JS runtimes that are optimized for MCUs such as lowjs [118] or Espruino [119] are also WebAssembly compatible. As a result, code from Rust, Go, or C can be compiled to the WebAssembly executable format `.wasm` and executed on various devices.

In other words, the prospects for WebAssembly are promising. On the one hand, its machine-like performance and code portability properties make it a suitable choice in the IoT domain. On the other hand, the security by design concept and its compatibility with JS facilitates a secure and rapid integration into state-of-the-art processes. For these reasons, WebAssembly support should be considered when developing applications in the context of the web. However, due to the ongoing development and definition of WebAssembly, it is not yet fully applicable. In particular, the bridge between the WoT standard and WebAssembly needs to be explored in more detail. The primary focus of the WoT standard was on JS because of the proximity to the web. Hence, the runtime choice

of this implementation determines JS even though it has similar resource requirements to Java. However, the difference to Java is that NodeJS is compatible with WoT and WebAssembly by default and hence gives a better initial position for future developments. In this implementation, especially gateways or external services with sufficiently enough computing power can run a JS runtime for now and later execute WebAssembly code if desired.

5.2.7 Semantics

W3C's WoT standard already contains a vocabulary for communication-related topics. However, a common understanding of the architecture's functionalities requires the definition of additional terms. Consequently, the choice is between extending an existing vocabulary or defining a new one. In any case, it is necessary to define architecture-specific terms that most likely no other vocabulary covers. Therefore, this implementation chooses to define a lightweight standalone vocabulary specifically developed for the given scenario. The concrete definition of terms is divided into two vocabularies that are listed in Appendix A. In detail, the `ec` A.1 namespace contains the diverse terms that are required to describe the architecture's functionalities. Examples of this collection are terms for access scopes, Direct Current (DC) power, or the action to start a charging process. The second vocabulary is a subset of the SunSpec specification and is located in the `sunspec` A.2 namespace. This part covers the needed terms to understand the DC power properties of a SunSpec compatible inverter. Even though the vocabulary development has not followed a recognized definition procedure, it works for the given implementation and is sufficient for demonstration purposes. In other words, the defined vocabulary only covers a part of the possible EC scenarios and must be extended or adjusted if used in production.

5.3 Network Service

The architecture design in Section 4.3 outsources the purpose of communicating with external parties to the Network Service. This way, the Network Service is described as an abstract architecture component that establishes the connection between local devices and external parties. In this implementation, the Network Service establishes the connection to the optimizer by utilizing the aspiring WireGuard protocol. The associated software components are installed on top of an Alpine Linux base image that is encapsulated in a Docker image. Furthermore, the previously determined Tailscale software for managing WireGuard connections is installed. This way, the operating system of the Network Service provides two relevant network adapters, as illustrated in Figure 5.2. The first adapter, which is named `eth0`, is the default network interface in the local subnet with an IP address of `10.10.0.2`. This interface is provided on the Docker level and simulates a physical network interface. The second interface is named `eth1` and is virtualized on the operating system level by WireGuard. In fact, this procedure is compatible with all environments where WireGuard is supported.

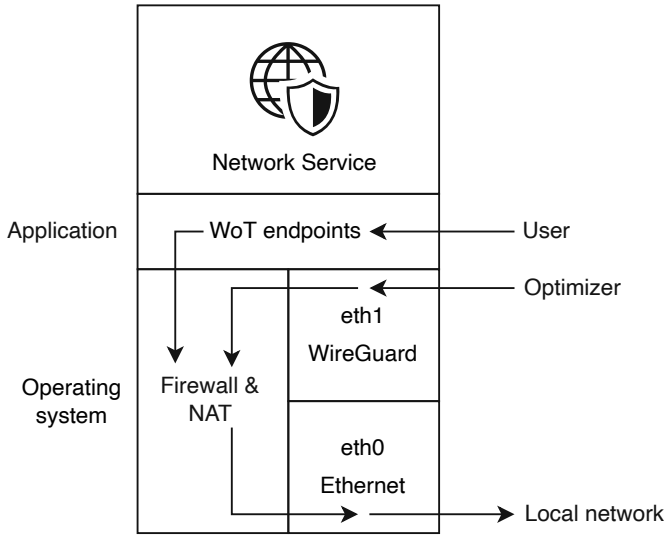


Figure 5.2: Network Service implementation.

Furthermore, Figure 5.2 visualizes the overall structure of the Network Service. In detail, the figure divides the software aspects into two layers. On the one hand, the application level at the top of the figure provides an interface for controlling the Network Service. On the other hand, the operating system on the bottom is responsible for network-related services such as NATing and firewall tasks. A sample request from the optimizer to the local network is visualized on the right side of the figure. First, the request reaches the `eth1` network interface and is further forwarded to the firewall. The firewall checks for applicable rules for the optimizer's IP address in the virtual network and drops invalid requests. Next, the source and target addresses are respectively adjusted to the Network Service's `eth0` interface and sent to the recipient in the local network. Due to the Linux-based environment, the desired forwarding and firewall behavior is configured by utilizing the `iptables` application. Moreover, permitting and denying access is mapped to the application layer where associated endpoints in a NodeJS application allow the configuration over HTTP.

The Network Service's TD contains the usage information for these endpoints as listed in B.3. A user's end device can recognize this TD and identify the Network Service's capabilities. Further, a user application can generate according UIs and provide suitable forms on the TD's basis. However, caution must be taken as firewall configuration is a sensitive task. Therefore, the `admin` scope is necessary to access the respective endpoints, as indicated on lines 54 and 84 in TD 5.1. Moreover, line 16 indicates the necessary JWT structure for configuring the firewall rules. By default, the firewall is configured to block all incoming and outgoing connections, so no unintended communication is possible. Due to the simplicity of the firewall rules endpoint, users can either inspect the actual firewall rules or adjust them accordingly. Updating an existing firewall rule is done by

submitting a new rule. The old one will get obsolete by an entry in the respective IP table. However, configuring network settings means an effort for the user. Therefore, the Network Service searches the local network and automatically passes secure devices and services to the optimizer. This means devices or services that strictly require HyperText Transfer Protocol Secure (HTTPS) and provide authorization control can automatically communicate with external parties such as the optimizer. The needed information for this intent is retrieved from the Directory Service. In the case of doubts, the automatic inclusion of firewall rules is prohibited.

TD 5.1: Snippets of the Network Service's TD. Complete TD listed in B.3.

```
⋮
    ...
12  "title": "Network Service",
13  "description": "Service that is responsible for connecting
    ↪ external parties.",
14  "id": "21da2b3f-2afa-4d2c-9b09-daba90eb75c7",
15  "securityDefinitions": {
16      "bearer_sc": {
17          "scheme": "bearer",
18          "in":"header",
19          "format": "jwt",
20          "alg": "RS256",
21          "authorization":
    ↪ "https://10.10.0.8:8080/.well-known/wot-thing-description",
22          "ec:scopes": ["observe", "control"]
23      }
24  },
25  "security": ["bearer_sc"],
26  "base": "https://10.10.0.2:8020",
27  "properties": {
28      "firewallRules": {
⋮           ...
50          "forms": [{
51              "href": "/properties/firewallRules",
52              "contentType": "application/json",
53              "op": ["readproperty"],
54              "ec:scopes": ["admin"]
55          }]
56      }
57  },
58  "actions": {
59      "setFirewallRule": {
⋮           ...
80          "forms": [{
81              "href": "/actions/setFirewallRule",
82              "contentType": "application/json",
83              "op": ["invokeaction"],
```

```
84                  "ec:scopes": ["admin"]
85              }]
86          }
87      }
 ⋮      ...
```

5.4 Security Service

The Security Service maintains a list of rights and is responsible for granting and denying access. Internally, the service represents the respective data structures as a JSON model with the previously determined JWT claims. In other words, the data consists of key-value pairs specifying the IDs of the issuer, inquirer, and the respective audience. Additional parts of the data are the array of access scopes and two timestamps for indicating issuing and expiration times, respectively. The underlying data is persisted in a LevelDB database and manipulated by a NodeJS application. Moreover, the NodeJS application serves the respective security functionality for users, devices, and services. The full scope of the functionalities is listed in the Security Service's TD in B.5. In detail, the API consists of seven endpoints that allow the respective authorization flow visualized in Figure 5.3. The prerequisite for this flow is that the participating devices and services understand and trust the Security Service's signature in a JWT.

Concretely, the scenario in Figure 5.3 visualizes the process where the optimizer wants to request data from the EV charging station. It is assumed that the EV charging station is fully compatible and already trusts the Security Service's signatures. That said, the first step for the optimizer is to find out about the authorization procedure of the EV charging station. For this purpose, the optimizer accesses the EV charging station's well-known-URI and fetches the respective TD as listed in B.6. Next, the optimizer parses the TD and searches the content for the token issuer's endpoints. The related information in TD 5.2 is stated on line 22. Additionally, the optimizer infers the EV charging station's functionalities and checks the required scopes for reading the EV charging station's state. In this case, the required scope is `observe`. Subsequently, the optimizer builds the authorization request with the intended scope and sends the request to the Security Service. For the purpose of identification, the Security Service sends back a newly generated JWT and stalls the request until the user grants access. The returned JWT initially has an empty scope array meaning the optimizer is not allowed to read the EV charging station's state. In order to get the desired scopes, the optimizer polls the Security Service for new tokens until the desired scopes are included. In other words, the optimizer has to trade in an old JWT token to get a new one with extended scopes. The scope array will remain empty until the user grants access to the optimizer. That is why the Security Service notifies the user about the recently received request. The subscription endpoint on line 245 in TD 5.3 provides a functionality to notify the user.

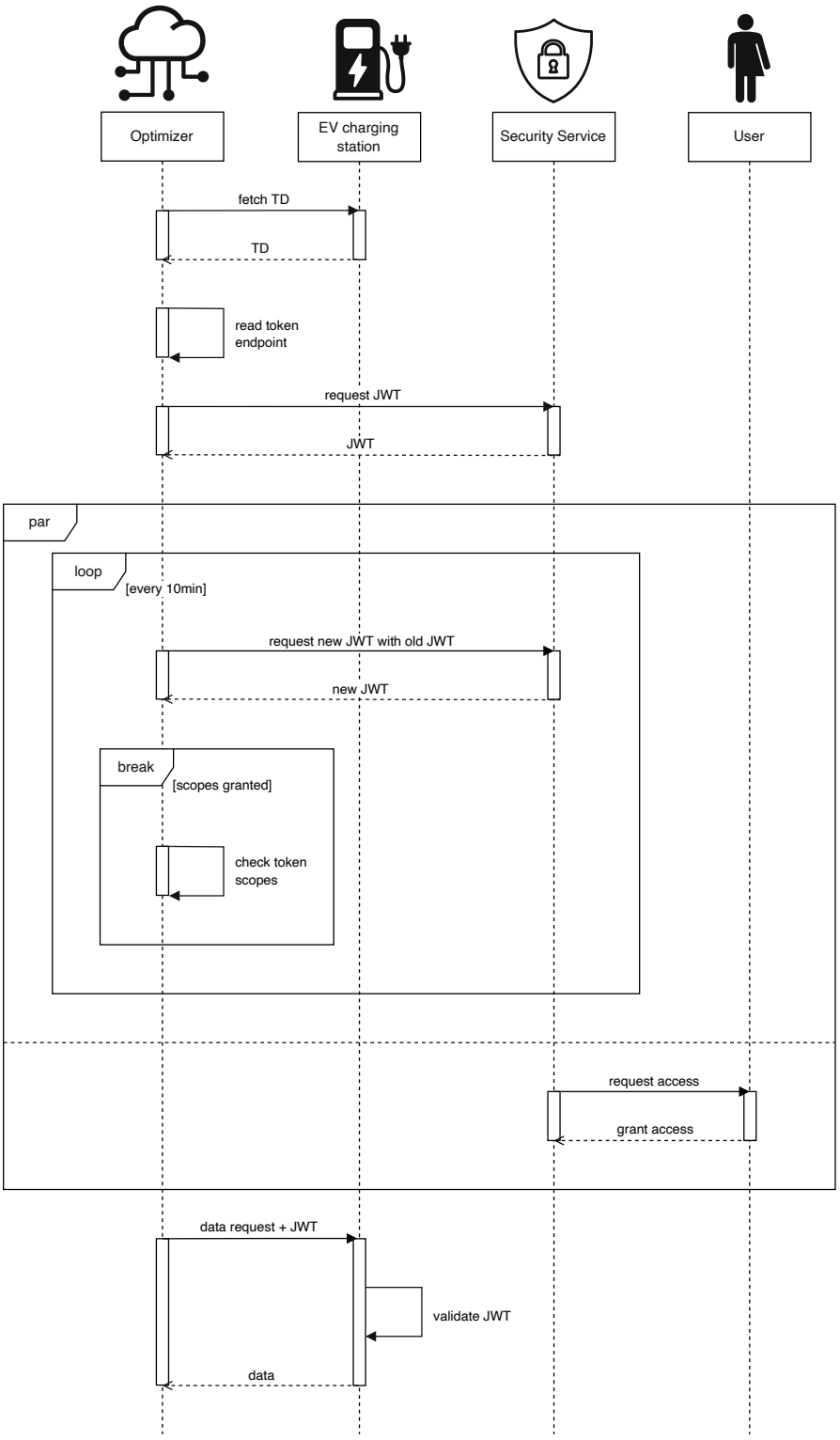


Figure 5.3: The architecture's authorization flow.

TD 5.2: Snippets of the EC charging station's TD. Complete TD listed in B.6.

```
 ⋮     ...
13     "title": "EV charging station",
14     "description": "Simulated charging station for electric
   ↪   vehicles",
15     "id": "2972af70-7059-41c7-b417-0be91ffb2d5f",
16     "securityDefinitions": {
17         "bearer_sc": {
18             "scheme": "bearer",
19             "in":"header",
20             "format": "jwt",
21             "alg": "RS256",
22             "authorization":
   ↪   "https://10.10.0.8:8080/.well-known/wot-thing-description",
23             "ec:scopes": ["observe", "control"]
24         }
25     },
26     "security": ["bearer_sc"],
 ⋮     ...
```

TD 5.3: Snippets of the Security Service's TD. Complete TD listed in B.5.

```
 ⋮     ...
218    "events": {
219        "newAccessRequest": {
 ⋮             ...
243            "forms": [{
244                "href": "/subscriptions/newAccessRequest",
245                "subprotocol": "longpoll",
246                "op": ["subscribeevent"],
247                "ec:scopes": ["admin"]
248            }]
249        }
250    }
 ⋮     ...
```

After the user has accepted the request, the Security Service includes the granted token claims in the next issued JWT. In other words, the optimizer gets the desired claims when refreshing the JWT. From this moment on, the optimizer can access the EV charging station's endpoints by attaching the JWT to the HTTP header of future requests. When the EV charging station receives such a request, it checks the claims of the attached JWT. Moreover, it checks the expiration time and ensures the validity by verifying the JWT's

signature. In this implementation, the token automatically expires after ten minutes. This means the JWT can be used for ten minutes until a new token is required. An inquirer may refresh the token before the old one expires. This policy is necessary to provide uninterrupted operation of the system. The actual refreshing procedure works the same way as the initial issuing of the token works. In other words, an inquirer has to prove previous access by passing an old JWT. The Security Service checks the signature and the current access rights and issues a new token. In case of an error, the JWT request fails with an HTTP error. When the device tries to make the request again, the Security Service immediately rejects it, so unnecessary user notifications are prevented. This way, a rejected device is permanently banned from the Security Service. In order to undo the ban, the user must manually intervene and adjust the security settings accordingly. However, the most important task of the process is that the user gets sure about an inquirer's identity. When requesting the JWT, an attacker could forge the inquirer's identity and fool the user by imitating another device. Consequently, the user might think that a malicious device wants access and unintentionally accepts the request. Hence, it is necessary that a user gets sure about a device's requests. After the first access is granted, the identification process is ensured by requiring old tokens to get a new one.

5.5 Discovery Service

The Discovery Service is a flexible component that is responsible for identifying devices and registering their TDs in the Directory Service. In fact, the Discovery Services's implementation consists of two parts. First, a NodeJS application periodically broadcasts mDNS/DNS-SD requests in the network to find compatible devices. If a compatible device answers, the Discovery Service searches for the device's TDs. In the compatible case, a device or service publishes its TD over HTTP and makes it accessible by a well-known URI. In this case, the Discovery Service fetches the TD and further publishes the content to the Directory Service. In the legacy case, the Discovery Service tries to identify promising devices and generate the respective TDs on its own. For this purpose, the Discovery Service's implements the following procedure:

1. Periodically scan the network for devices. This is implemented by scanning TCP port 502.
2. If a promising device is found, check if it is Modbus TCP compatible by sending a SunSpec Modbus request.
3. If the device answers accordingly, read the model specification and generate the related TD.
4. Publish the TD in the Directory Service.

Like the mDNS/DNS-SD approach, this procedure is implemented in a NodeJS application that periodically checks the network in an interval of ten minutes. Due to the automatic

process, the control endpoints for users are limited. The implementation only allows users to start or stop the discovery process as described in TD 5.4 on lines 46 and 56, respectively. The available properties on line 27 are even less extensive, consisting of a single endpoint that indicates the state of the discovery process.

TD 5.4: Snippets of the Discovery Service's TD. Complete TD listed in B.2.

```
⋮     ...
45    "actions": {
46        "startDiscovery": {
47            "@type": ["ec:startDiscovery"],
48            "description": "Start discovering",
49            "forms": [{
50                "href": "/actions/startDiscovery",
51                "contentType": "application/json",
52                "op": ["invokeaction"],
53                "ec:scopes": ["admin"]
54            }]
55        },
56        "stopDiscovery": {
57            "@type": ["ec:stopDiscovery"],
58            "description": "Stop discovering",
59            "forms": [{
60                "href": "/actions/stopDiscovery",
61                "contentType": "application/json",
62                "op": ["invokeaction"],
63                "ec:scopes": ["admin"]
64            }]
65        }
66    }
⋮     ...
```

5.6 Directory Service

Typical discovery processes on the Internet consist of a central point that collects service information and further broadcasts the information to interested parties. This way, services can uniformly exchange information and must only be compatible with the central directory. Two substantial factors must be covered to make this work. First, the format of device information has to be specified. Second, the interface of the directory has to be known. In the proposed architecture in Section 4.3, the central point is represented by the Directory Service. To provide the intended functionality to other devices or services, the Directory Service needs to implement a standard interface and use a standard exchange format. One possible solution to this task is selecting proper communication technologies and specifying the related API endpoints independently from existing solutions. However, due to the determination of HTTP and the WoT standard in the last steps, a suitable solution is to implement the specially developed HTTP

API from W3C. The specification of another API would only lead to incompatibility of devices, services and directory structures. Therefore, WoT's directory API is determined for registering and looking up TDs.

Another topic of the Directory Service is the internal handling of data and the implementation of related query functionality. The latter task is implemented by utilizing WoT Hive [120]. WoT Hive is an implementation of W3C's thing directory API. The respective data management is done by utilizing the triplestore Apache Jena Fuseki. Its backend is executed as a separate Docker container but isolated from all but the Directory Service container. Indeed it would be possible to integrate the Fuseki features directly into the WoT Hive container. However, it is more suitable to leave the separation of concerns as it is due to maintenance aspects. This approach additionally has the advantage of exchanging the backend without changing the Directory Service implementation. Due to the standardized interface of Fuseki's SPARQL endpoints, every other triple store implementation is eligible. The public API of the Directory Service is described in TD B.1. In particular, the TD largely follows the officially thing directory API in [51].

5.7 Runtime Service

Similar to the other services, the Runtime Service provides a host application and a common API to control it. In detail, the Runtime Service utilizes an instance of the NodeJS runtime with the *vm2* package [121]. This setup enables a secure environment for remote execution of JS code. The API of the Runtime Service provides access to the hosted runtime by providing three endpoints. First, it is possible to get an overview of the running jobs by querying the `runningScripts` endpoint. This endpoint is listed on line 28 in TD 5.5. The other two endpoints on lines 55 and 91 allow the user to deploy a script or stop a running instance. Indeed, remote code execution is a sensitive topic. Therefore, the interaction with the control endpoints is restricted to admins only. Additionally, it is assumed that users self-determine the needed script functionalities and deploy the respective scripts over the `startScript` endpoint.

TD 5.5: Snippets of the Runtime Service's TD. Complete TD listed in B.4.

```
⋮       ...
27      "properties": {
28          "runningScripts": {
⋮               ...
52          }
53      },
54      "actions": {
55          "startScript": {
⋮               ...
90          },
91           "stopScript": {
```

```
  ⋮                ...
104          }
105      }
  ⋮      ...
```

5.8 Device interaction

The previous sections described the implementation of the architecture's key components. This section demonstrates how the key components interact when integrating devices. For this purpose, the following sections describe two cases of device integration in the prosumer environment. On the one hand, a regular scenario with a WoT compatible EV charging station demonstrates a simple case where no additional translation of the communication is necessary. On the other hand, a legacy inverter scenario demonstrates the adoption of a legacy device and the instantiation of the corresponding gateway.

5.8.1 Compatible device

In this scenario, the user onboards a novel EV charging station. The first step is to startup the device as illustrated in Figure 5.4. This step is simulated by starting up the respective Docker container, which contains a dummy implementation of an EV charging station. After the container has started up, the Discovery Service identifies the EV charging station in its next service discovery round. Subsequently, the Discovery Service registers the EV charging station's TD in the Directory Service. This fires an event in the Directory Service's API on which the Network Service and the optimizer have subscribed. Due to the full compatibility of the EV charging station, the Network Service automatically creates a firewall rule and forwards requests to the EV charging station. Furthermore, the optimizer classifies the EV charging station as a relevant device for optimization purposes.

Consequently, the optimizer wants access to this device and contacts the Security Service. The respective sequence of steps is already visualized in Figure 5.3. In this case, the necessary claims to control the EV charging station are `observe` and `control`, which the optimizer requests from the Security Service. The Security Service stalls this request and, in the meantime, asks the user for permission. When the user grants access, the next time the optimizer issues a new token, the respective claims are set in the JWT. From this moment on, the optimizer can attach the JWT to requests and proves the EV charging station allowance. This way, the user has to interact with the system only once for granting access to the requesting systems. The resulting process comes with low effort by sending the required messages on the user's end devices. Indeed, requests have to be carried out by each device and, in the case of multiple external parties, lead to multiple possible interactions with the system. Additionally, this scenario assumes that the Security Service's signature key is already installed on the EV charging station and trusts the signed JWTs from the Security Service.

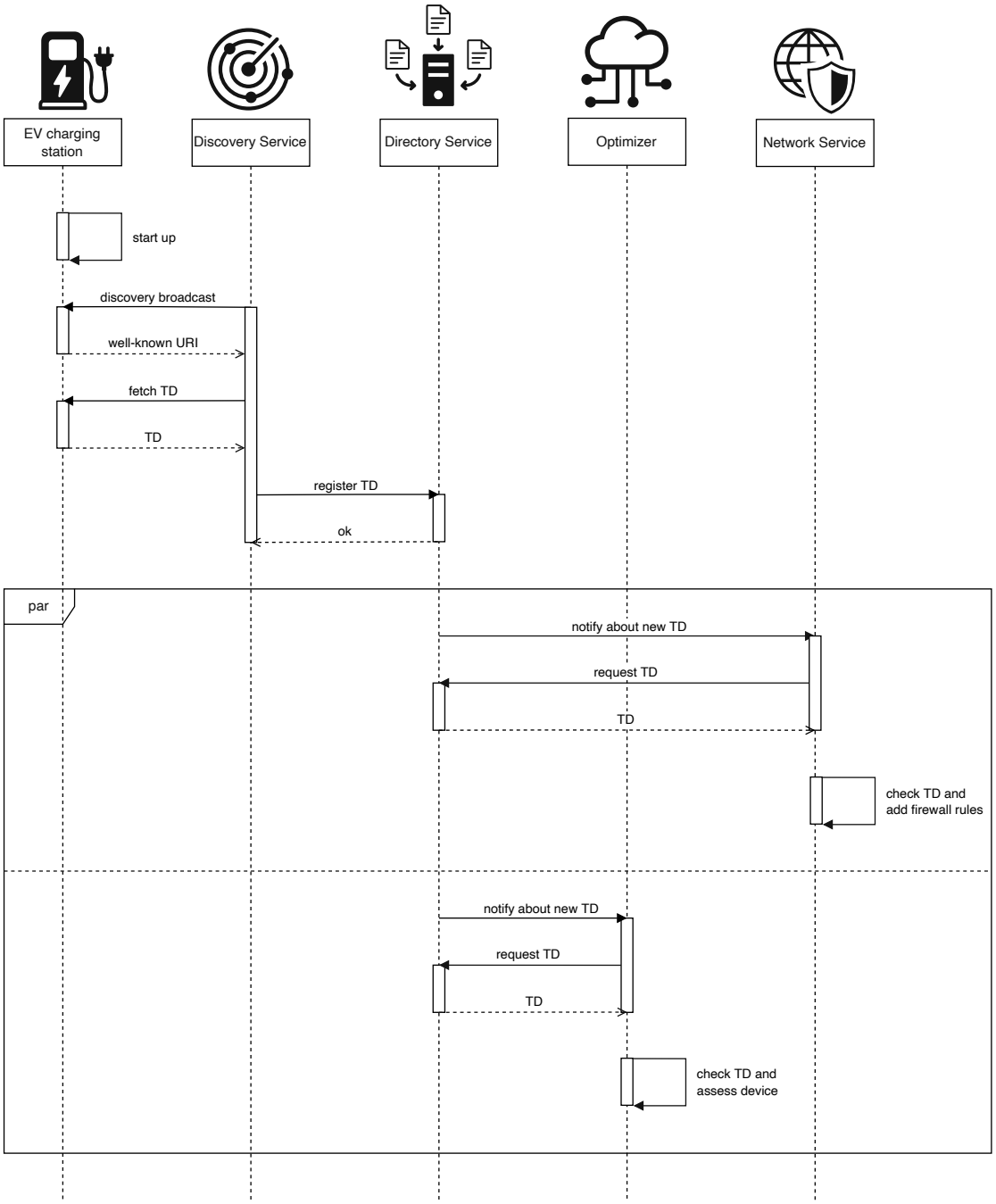


Figure 5.4: Onboarding steps of a compatible EV charging station. The procedure resumes with access requests as illustrated in Figure 5.3.

5.8.2 Legacy device

The second scenario demonstrates how the user can onboard a legacy inverter. In detail, the inverter implements the SunSpec Modbus protocol over Modbus TCP. The scenario begins like the compatible scenario by starting up the corresponding Docker container. The respective sequence is visualized in Figure 5.5. After starting up the container, the Discovery Service identifies the inverter in the next service discovery round. In this case, the Discovery Service recognizes the legacy device and creates a legacy device TD B.7. The onboarding procedure continues like in the compatible scenario with the registration of the TD in the Directory Service. In this case, the Network Service does not automatically add a route to the inverter due to the absence of required security mechanisms. Likewise, the optimizer behaves differently by not requesting access due to incompatibility with Modbus TCP. Next, the user is required to solve the problem by searching for a compatible gateway translator. If a suitable code is found, the user instantiates the gateway code by deploying the script to the Runtime Service. Subsequently, the user has to configure the script code and link it to the inverter's TD. Next, the instantiated gateway publishes its TD B.8, and the Discovery Service can recognize it in the next scan. From this moment, the onboarding procedure takes place like with a compatible device in Figure 5.4. This means the Directory Service gets notified about a new TD, the Network Service passes the requests and the optimizer requests access. In the gateway case, the scope is only observable due to the restricted functionalities of the legacy inverter. However, compatibility problems are solved, and communication can occur like the legacy inverter would support the technology choice.

The related gateway code consists of three components. First, it implements an HTTP interface that exposes a server that simulates the inverter's functionalities. Second, it implements a Modbus TCP client that is able to query the related registers at the inverters interface. The third component is a middleware function that translates SunSpec Modbus content to a JSON format like it is described on line 30 and 31 in the gateway's TD 5.6. In detail, the gateway reads two properties of the inverter if a request arrives. The inverter supports the SunSpec model 103 that splits the DC power value of the inverter into two registers. One of the registers contains the actual value, and the second register the value's scaling factor. This means the representation of power is divided into two separate Modbus entries that must be combined for further usage. For this, the gateway simply fetches the data from the Modbus interface and passes the content in binary format to the transformation function. Next, the function locates the two values and scales the power value with the corresponding scaling factor. The transformed value is returned to the requester. The meaning and unit of this value are already known due to the gateway's TD B.8.

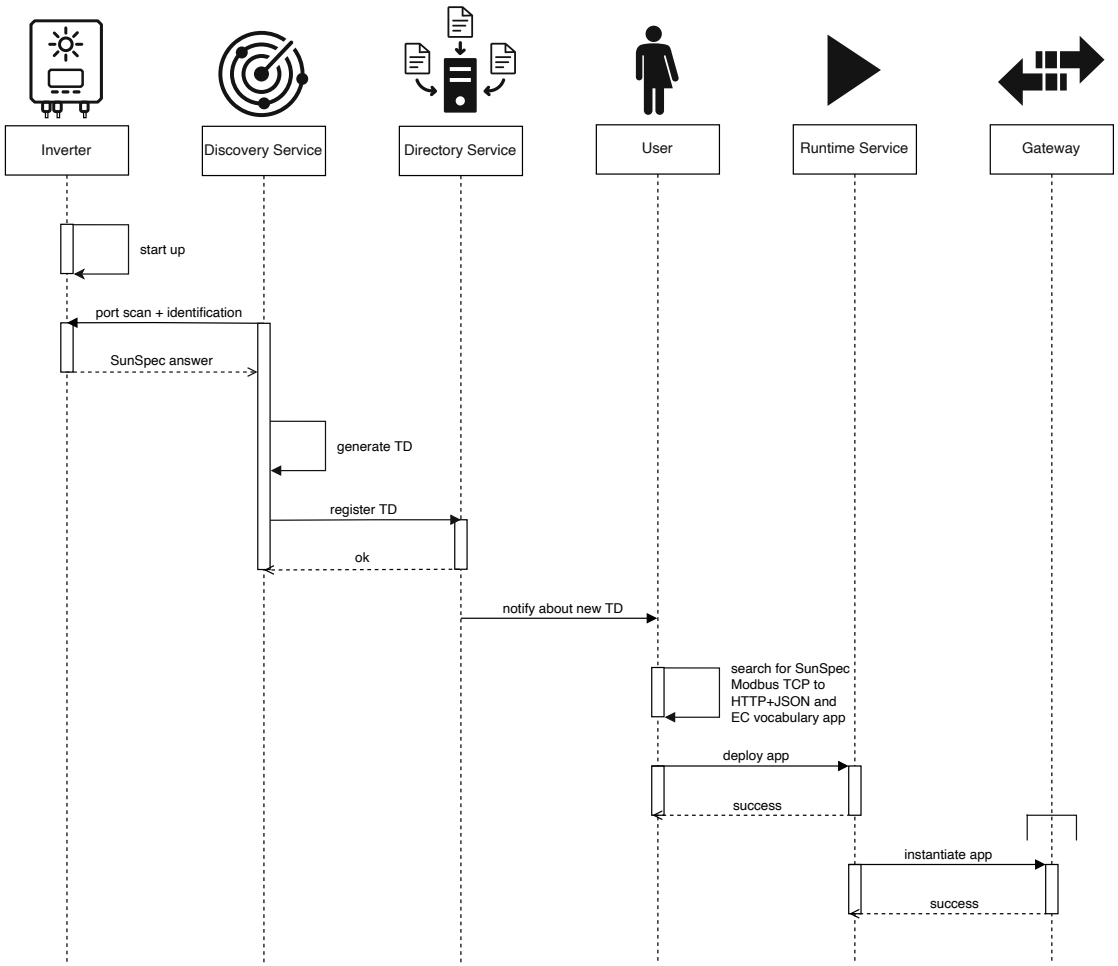


Figure 5.5: Onboarding steps of a legacy inverter. The procedure resumes with onboarding the gateway like in the compatible case visualized in Figure 5.4.

TD 5.6: Snippets of the gateway's TD. Complete TD listed in B.8.

```
⋮     ...
29    "properties": {
30        "DCPower": {
31            "@type": ["ec:DCPower"],
⋮             ...
42        }
43    }
⋮     ...
```

CHAPTER 6

Evaluation and discussion

This chapter evaluates the proof-of-concept implementation from the previous chapter by checking if the predefined architecture requirements of Section 3.4 are fulfilled. As it turns out, the architecture covers the requirements of the respective EC scenarios and significantly facilitates interoperability and integration problems. However, the benefits of the solution depend on multiple factors. A subsequent discussion covers these factors and performs a shallow security analysis, discusses the architecture's overhead, and examines acceptance assumptions of the initial situation. As a result, the discussion brings up open questions and states factors that may also be relevant for other IoT scenarios.

6.1 Requirements fulfillment

The following subsections recapitulate the fulfillment criteria from Section 3.4 and qualitatively verify the architecture behavior of the proof-of-concept implementation in Chapter 5.

6.1.1 Requirement 1: Confidentiality and access control

Fulfillment criteria: The requirement is fulfilled if the architecture supports authorization mechanisms that allow fine-grained access control. Additionally, the architecture must ensure that traffic between local devices and external parties is fully encrypted.

Verification result: The initial deployment of the architecture's services fulfills this requirement by design. Communication is encrypted, and fine-grained access control through the implemented authorization mechanism ensures the realization of the least privilege principle. In the context of non-architecture devices, two scenarios in Section 5.8 demonstrate the corresponding behavior. The EV charging station scenario shows how a compatible device can participate in the architecture. In fact, the device has the same security properties as an architecture service and, therefore, also fulfills the security

requirement by design. The second scenario demonstrates the behavior of a legacy security implementation. In this case, the architecture provides the gateway mechanism for encapsulating insecure endpoints with the Runtime Service. Additionally, the Network Service takes care of encrypting even insecure traffic. Due to the use of WireGuard, traffic to external parties such as the optimizer is fully encrypted on the IP layer. In other words, the architecture implements a state-of-the-art security scheme but also provides mechanisms for legacy security instruments. This way, secure and private EC environments can be built, even if it comes to the interaction of local devices and external parties.

6.1.2 Requirement 2: Extensibility

Fulfillment criteria: The requirement is fulfilled if additional devices and services can extend the architecture after configuring the initial setup.

Verification result: The service-oriented architecture design from Chapter 4.3 leads to a separation of architecture concerns. In fact, the different services are primarily independent and can be configured and distributed as desired. The central aspect of fulfilling this requirement is the description of device and service interfaces. Unlike it is best practice in software engineering to only document web APIs for other developers, this approach made use of semantic annotations in the TD format. Devices and services interpret these annotations and automatically infer the required endpoints. Subsequent integration of novel devices and services can read the endpoints and extend the functionality of the architecture with little effort. The second case of extensibility deals with already deployed devices that want to use novel systems. In this case, it can be the case that not all of the semantic annotations are understandable. Consequently, software components of the architecture must be updated if they do not outsource semantic translation to external services. In other words, the architecture supports extensibility and provides a future-proof environment for energy-related use cases due to the modular design and the utilization of a standard DD.

6.1.3 Requirement 3: Interoperability

Fulfillment criteria: The requirement is fulfilled if devices and services can communicate without adjustments. For legacy devices, the architecture must provide a procedure to make them interoperable.

Verification result: The scenarios in Section 5.8 demonstrate two different initial situations that the architecture handles. On the one hand, the first scenario demonstrates how the system interacts with fully compatible devices and services. Due to the implementation of standard technology such as HTTP, JWT and the predefined vocabulary, devices and services already fulfill the first part of the criteria by default. The second case demonstrates the case where a legacy input device is made interoperable by utilizing the architecture's mechanisms for discovering and translating device communication. This

scenario tests the second criteria by making a legacy device accessible to the other parts of the architecture.

6.1.4 Requirement 4: Usability

Fulfillment criteria: The requirement is fulfilled if the EC architecture provides support for standard discovery and integration mechanisms with effort.

Verification result: The actual fulfillment of the criteria is already met with the utilization of mDNS with DNS-SD. However, the key aspect of the usability requirement is the user interaction with the system. Both scenarios in Section 5.8 demonstrate few user interaction. The tradeoff is between security and convenience. Finally, the implementation determines to automatically adopt devices through the discovery process until user input is required for granting access. Especially the discovery and adaption processes in a fully compatible scenario results in low user effort. The integration of legacy devices came with higher effort due to the manual search for suitable gateway scripts. This process could be simplified by applying approaches from the area of automatic reasoning. However, the architecture provides interfaces and opportunities to cover such cases.

In summary, the proposed architecture covers all predefined requirements from Section 3.4 by introducing assisting mechanisms. The service-oriented architecture approach enables a secure, extensible, interoperable, and convenient environment that facilitates the communication of local devices with external service providers in an EC. Especially the aspects in the context of interoperability and integration are covered as desired in Section 1.2. This way, the proposed architecture is assessed as a suitable architecture in the EC context.

6.2 Solution assessment

The previous section verified the implementation according to the predefined architecture requirements. Even if the verification results were mostly positive, the evaluation must be put into context. The first point of this context concerns the physical or underlying situation. First, the architecture is only applicable on the application level of the IP family. In detail, only TCP and UDP is supported. This limits the conceivable scenarios and results in deficits in contrast to the solution proposed by Han et al. [10]. Moreover, the implementation only covered only a minimal set of technology. Consequently, only a few possible scenarios could be verified, and other communication paradigms like publish/subscribe patterns were neglected. Apart from these points, the communication assessment covered essential aspects of a local network. However, the following points of security, overhead, and acceptance have to be discussed in more detail.

6.2.1 Security

The applied security mechanisms protect users, devices, and services from external adversaries. However, there are some cases where security is still an issue. Especially in the local network, there are few security mechanisms to mitigate exploitation. First, a novel device that is introduced into the architecture could forge its identity and fool the user into being another device that the user trusts. If the malicious device requests access to a specific device, the user may unintentionally grant access. However, if the user's network consists of an insecure network, security is anyways at risk. Therefore, the local network is assumed to be a secure environment, and the user must care about an appropriate level of security. Possible ways would be to utilize proven security schemes such as IEEE 802.1X network access control [122]. Another mitigation is to put some unique hardware identification on the device that other devices cannot know. The user could check the respective request and compare it with the hardware identification.

Another point is the automatic processes like discovery, adoption, and network passthrough. These concerns also concern the local level. If an adversary somehow gets access to the local network, it could mimic another device by assigning its IP address. The Network Service may think that the device is secure due to the TD of the mimicked device and passes the request. Because the firewall only concerns IP level traffic, the original device would be disclosed in the network even if it does not provide any security mechanisms. However, this scenario is relatively unlikely. First, an external instance of the adversary must participate in the Network Service's virtual network. Additionally, it must somehow mimic a local instance. The question is why an adversary wants to achieve the automatic passthrough if local access is achieved anyways. In other words, this aspect also depends on access to the local network and concerns the intrusion of adversaries into devices and services.

The third security aspect describes a possible way so that external parties get access to the local network. The respective security vulnerability concerns the remote execution of scripts. Due to the provision of a full-fledged runtime, it is essential to trust the executed software, like gateway scripts. Even trusted code execution could lead to a security breach that enables the adversary access to the local network. In other words, even if the local network is secured from external adversaries, the Runtime Service is already in the local network and could cause malicious behavior like described in the previously mentioned scenarios.

6.2.2 Overhead

Another aspect of the architecture is the overhead of the introduced components and the associated expense for the user. More precisely, EC users have to provide runtimes for the required services. However, standard runtimes are suitable in most cases due to the limited runtime design. Conceivable hardware is single-board computers such as Raspberry Pis or management boxes from existing smart home systems. These systems may lead to little introduction effort and space consumption. Moreover, the implementation of such a

solution is also feasible for small budgets. However, the architecture causes additional overhead. First, the required service software must be installed on the respective systems. Afterward, every device needs to trust the Security Service's certificate for validating JWTs and needs to know its IP address. Constrained devices are at risk of neither supporting the secure implementation of HTTP nor the certificate validation of the security scheme. Additionally, the search for suitable gateway solutions and code requires the user to invest even more time and knowledge into the adoption process of legacy devices. This aspect could discourage people even if the architecture took extraordinary measures to reduce this overhead.

6.2.3 Acceptance

The initial architecture requirements consist of recent aspects in the context of IoT. These requirements represent one possible view of the respective EC scenarios. In particular, this means that even the few user interactions in the presented scenarios may be too much for broad acceptance. Furthermore, users may not want to care about service structures or are unwilling to invest in new technology that follows the proposed architectural design. Ready-for-use alternatives such as directly connected cloud devices could be favored. In this case, various beneficial properties of the architecture would be traded-in for convenience. However, the involvement of third-party providers in the architecture allows the emergence of various service structures. The actual third-party providers could develop novel business models that motivate the user to accept the service landscape of the architecture. Also, business models that pay the EC users are conceivable. In any case, the architecture is ready for these cases and supports various possible scenarios.

CHAPTER 7

Conclusion and outlook

This final chapter summarizes the thesis and repeats key aspects of the research process. Moreover, an outlook of promising approaches reveals ideas for extending and stabilizing the architecture approach.

7.1 Summary

In short, this thesis aimed to design and implement a suitable architecture for ECs. For this purpose, the research process started with assessing conceivable energy-related scenarios. The outcome of this step was the description of so-called interaction scenarios that represent abstract ways of how users and external parties can jointly interact inside of ECs. Furthermore, this step led to the specification of four architecture requirements regarding interoperability, usability, confidentiality and access control, and extensibility. To cover these requirements, Section 4.2 identified eight key aspects that represent guidance for conceiving the architecture design. On this basis, the architecture design came up with a service-oriented approach that consists of five services. The underlying design strategy was to outsource essential concerns into individual services that all are compatible with a preselected subset of novel communication technology. This process led to a modular architecture that was described in Section 4.3. Furthermore, the structure and behavior of every service was outlined in individual sections.

Subsequently, the proposed architecture was demonstrated in a proof-of-concept implementation. The underlying testbed consisted of a Docker environment where every service and the underlying network situation was simulated. In detail, the implementation scenario was built around a prosumer scenario that consists of two local energy-related devices and an external optimizer that wants to access these devices. The implementation of this scenario was started by following the key aspects of the proposed architecture in Section 4.2. Consequently, the implementation started with the selection of a suitable subset of technology such as HTTP, mDNS and JWT. Further, the implementation

explained how the technology subset is applied to the architecture components and outlined the functionalities and service descriptions of the five architecture services.

For demonstration purposes, the implementation was evaluated by demonstrating two different scenarios. The first scenario was about integrating a technology-compatible EV charging station. By utilizing parts of the WoT standard, the implementation showed a fully compatible onboarding process and described the associated assumptions. As a result, the interaction of a potential user with a device was minimal. In fact, the proposed scenario only required user interaction once for granting device access. The second scenario of the implementation demonstrated a legacy device integration. For this purpose, a legacy inverter that implements the Modbus SunSpec protocol was adopted. This scenario demonstrated how the gateway approach could be used to facilitate interoperability by utilizing the architecture's assistance mechanisms. In this case, users had to interact with the architecture twice. First, users had to recognize, find and deploy an interoperable gateway script, and second, they had to configure the associated access rights.

The proof-of-concept revealed that the architecture is a suitable approach to initially set up the EC communication structure. A critical assessment came to the result that the architecture provides components for assisting integration and interoperability problems. However, the findings of this thesis mainly concern the overall architecture. Especially the concrete details of knowledge representation, discovery, and protocol translation are only covered rudimentary. For this purpose, the following section explains ideas and promising approaches for future work.

7.2 Future work

One of the further topics of this thesis is to define a standard vocabulary that all participating parties understand. In other words, it is necessary to carefully define a vocabulary that covers multitudes of possible use cases and reaches a high level of acceptance. Alternatively, the presented approach could be extended by utilizing gateways or vocabulary mappers that translate labels or knowledge graphs from one vocabulary into another. The related research approaches for ontology matching and alignment look promising, and their application must be tested. Additionally, the representation of a standard DD such as WoT's TD format could be utilized to model the whole communication and description vocabulary on a higher level where knowledge-based systems can understand the processes and feed the information into AI systems.

Another topic is interoperability and integration in the context of security. When writing this thesis, there are several interesting security mechanisms in an early adoption stage. First, it is interesting how blockchain technology suits the architecture for authorization purposes. A public blockchain like IOTA that is also verifiable by IoT devices could enable the decentralized management of access control. Second, security interoperability may be facilitated if the applied mechanisms are described in a standard format. The DID document specification from W3C addresses this issue and aims to develop a common

format to represent various kinds of authorization or access control mechanisms [58]. Both approaches seem suitable for the proposed architecture and may increase the level of interoperability.

The final point in this outlook is the execution of third-party scripts. On the one hand, these scripts should be able to process complex tasks. On the other hand, resource intensity and security are central topics of IoT. In other words, the right trade-off between these properties in the context of gateways and IoT devices must be investigated in more detail. The WoT Scripting API brought up the idea of running lightweight scripts directly on IoT devices [48]. However, the runtime of the first Scripting API implementations is built on JS runtimes that could possibly be too extensive for constrained devices. WebAssembly reveals new opportunities for providing a lightweight and secure runtime concept that can execute portable source code from the server to the web. Moreover, the interplay of related runtimes and DD such as TD has to be investigated in more detail. Maybe there is a way to represent the library and runtime environment of the respective scripting engine in a description format. The interesting aspect is the modeling of specific runtime capabilities and restrictions that all participants understand.

APPENDIX A

Vocabularies

A.1 Energy Community namespace

TD A.1: Vocabulary of essential EC related terms for the proof-of-concept implementation

```
// Services
ec:directoryService
ec:discoveryService
ec:networkService
ec:runtimeService
ec:securityService

// Security
ec:scopes
ec:scope
ec:pendingAccessRequests
ec:newAccessRequestArrived
ec:grantedAccess
ec:grantAccess
ec:requestAccess
ec:refuseAccess
ec:accessToken
ec:issueToken
ec:accessEntry
ec:sourceId
ec:target

// Abstract types
ec:gateway
ec:energyDevice
ec:inverter
ec:EVChargingStation

```

```
// Runtime Service
ec:runningScripts
ec:scriptId
ec:scriptName
ec:startScript
ec:stopScript
ec:jsCode

// Miscellaneous
ec:state
ec:allowed

// Discovery Service
ec:startDiscovery
ec:stopDiscovery

// EV charging station
ec:stateOfCharge
ec:startCharging
ec:stopCharging

// Network Service
ec:firewallRules
ec:firewallRule
ec:setFirewallRule
ec:sourceIP
ec:destinationIP

// Inverter
ec:DCPower
```

A.2 SunSpec namespace

TD A.2: Vocabulary of essential SunSpec related terms for the proof-of-concept implementation

```
sunspec:inverter
sunspec:models
sunspec:value
sunspec:desc
sunspec:label
sunspec:name
sunspec:sf
sunspec:size
sunspec:type
sunspec:units
sunspec:name
```

APPENDIX B

Thing Descriptions

B.1 Directory Service

TD B.1: Directory Service's TD

```
{
  "@context": [
    "http://www.w3.org/ns/td",
    "https://w3c.github.io/wot-discovery/context/discovery-context.jsonld",
    {
        "ec": "http://www.tuwien.ac.at/autosys/leonhard-esterbauer/energy-community-architecture"
    }
  ],
  "@type": [
      "ThingDirectory",
      "ec:directoryService"
  ],
  "title": "Directory Service",
  "description": "Maintains and distributes thing descriptions",
  "id": "50ead19a-30ee-4c1a-bd67-2282c3cf80bf",
  "securityDefinitions": {
      "bearer_sc": {
          "scheme": "bearer",
          "in":"header",
          "format": "jwt",
          "alg": "RS256",
          "authorization":
  "https://10.10.0.8:8080/.well-known/wot-thing-description",
          "ec:scopes": ["observe", "control"]
      }
  },
```

```
26    "security": ["bearer_sc"],
27    "base": "https://10.10.0.11:8110",
28    "properties": {
29      "things": {
30        "description": "Retrieve all Thing Descriptions",
31        "uriVariables": {
32          "offset": {
33            "title": "Number of TDs to skip before the page",
34            "type": "number",
35            "default": 0
36          },
37          "limit": {
38            "title": "Number of TDs in a page",
39            "type": "number"
40          },
41          "sort_by": {
42            "title": "Comparator TD attribute for collection sorting",
43            "type": "string",
44            "default": "id"
45          },
46          "sort_order": {
47            "title": "Sorting order",
48            "type": "string",
49            "enum": [
50              "asc",
51              "desc"
52            ],
53            "default": "asc"
54          }
55        },
56        "forms": [
57          {
58            "href": "/api/things{?offset,limit,sort_by,sort_order}",
59            "htv:methodName": "GET",
60            "response": {
61              "description": "Success response",
62              "htv:statusCodeValue": 200,
63              "contentType": "application/ld+json",
64              "htv:headers": [
65                {
66                  "htv:fieldName": "Link"
67                }
68              ]
69            },
70            "additionalResponses": [
71              {
72                "description": "Invalid query arguments",
73                "contentType": "application/problem+json",
74                "htv:statusCodeValue": 400
```

```
75              }
76            ],
77            "scopes": "readAll"
78          }
79        ]
80      }
81    },
82    "actions": {
83      "createThing": {
84        "description": "Create a Thing Description",
85        "uriVariables": {
86          "id": {
87            "@type": "ThingID",
88            "title": "Thing Description ID",
89            "type": "string",
90            "format": "iri-reference"
91          }
92        },
93        "forms": [
94          {
95            "href": "/api/things/{id}",
96            "htv:methodName": "PUT",
97            "contentType": "application/td+json",
98            "response": {
99              "description": "Success response",
100             "htv:statusCodeValue": 201
101           },
102           "additionalResponses": [
103             {
104               "description": "Invalid serialization or TD",
105               "contentType": "application/problem+json",
106               "htv:statusCodeValue": 400
107             }
108           ],
109           "scopes": "write"
110         }
111       ]
112     },
113     "createAnonymousThing": {
114       "description": "Create an anonymous Thing Description",
115       "forms": [
116         {
117           "href": "/api/things",
118           "htv:methodName": "POST",
119           "contentType": "application/td+json",
120           "response": {
121             "description": "Success response including the
    ↪  system-generated URI",
122             "htv:headers": [
```

```
123                 {
124                   "description": "System-generated URI",
125                   "htv:fieldName": "Location"
126                 }
127               ],
128               "htv:statusCodeValue": 201
129             },
130             "additionalResponses": [
131               {
132                 "description": "Invalid serialization or TD",
133                 "contentType": "application/problem+json",
134                 "htv:statusCodeValue": 400
135               }
136             ],
137             "scopes": "write"
138           }
139         ]
140       },
141       "retrieveThing": {
142         "description": "Retrieve a Thing Description",
143         "uriVariables": {
144           "id": {
145             "@type": "ThingID",
146             "title": "Thing Description ID",
147             "type": "string",
148             "format": "iri-reference"
149           }
150         },
151         "safe": true,
152         "idempotent": true,
153         "forms": [
154           {
155             "href": "/api/things/{id}",
156             "htv:methodName": "GET",
157             "response": {
158               "description": "Success response",
159               "htv:statusCodeValue": 200,
160               "contentType": "application/td+json"
161             },
162             "additionalResponses": [
163               {
164                 "description": "TD with the given id not found",
165                 "contentType": "application/problem+json",
166                 "htv:statusCodeValue": 404
167               }
168             ],
169             "scopes": "read"
170           }
171         ]
```

```
    },
    "updateThing": {
      "description": "Update a Thing Description",
      "uriVariables": {
        "id": {
          "@type": "ThingID",
          "title": "Thing Description ID",
          "type": "string",
          "format": "iri-reference"
        }
      },
      "forms": [
        {
          "href": "/api/things/{id}",
          "htv:methodName": "PUT",
          "contentType": "application/td+json",
          "response": {
            "description": "Success response",
            "htv:statusCodeValue": 204
          },
          "additionalResponses": [
            {
              "description": "Invalid serialization or TD",
              "contentType": "application/problem+json",
              "htv:statusCodeValue": 400
            }
          ],
          "scopes": "write"
        }
      ]
    },
    "partiallyUpdateThing": {
      "description": "Partially update a Thing Description",
      "uriVariables": {
        "id": {
          "@type": "ThingID",
          "title": "Thing Description ID",
          "type": "string",
          "format": "iri-reference"
        }
      },
      "forms": [
        {
          "href": "/api/things/{id}",
          "htv:methodName": "PATCH",
          "contentType": "application/merge-patch+json",
          "response": {
            "description": "Success response",
            "htv:statusCodeValue": 204
```

```
            },
            "additionalResponses": [
              {
                "description": "Invalid serialization or TD",
                "contentType": "application/problem+json",
                "htv:statusCodeValue": 400
              },
              {
                "description": "TD with the given id not found",
                "contentType": "application/problem+json",
                "htv:statusCodeValue": 404
              }
            ],
            "scopes": "write"
          }
        ]
      },
      "deleteThing": {
        "description": "Delete a Thing Description",
        "uriVariables": {
          "id": {
            "@type": "ThingID",
            "title": "Thing Description ID",
            "type": "string",
            "format": "iri-reference"
          }
        },
        "forms": [
          {
            "href": "/api/things/{id}",
            "htv:methodName": "DELETE",
            "response": {
              "description": "Success response",
              "htv:statusCodeValue": 204
            },
            "additionalResponses": [
              {
                "description": "TD with the given id not found",
                "contentType": "application/problem+json",
                "htv:statusCodeValue": 404
              }
            ],
            "scopes": "write"
          }
        ]
      },
      "searchJSONPath": {
        "description": "JSONPath syntactic search",
        "uriVariables": {
```

```
270         "query": {
271           "title": "A valid JSONPath expression",
272           "type": "string"
273         }
274       },
275       "safe": true,
276       "idempotent": true,
277       "forms": [
278         {
279           "href": "/search/jsonpath?query={query}",
280           "htv:methodName": "GET",
281           "response": {
282             "description": "Success response",
283             "contentType": "application/json",
284             "htv:statusCodeValue": 200
285           },
286           "additionalResponses": [
287             {
288               "description": "JSONPath expression not provided or
    ↪  contains syntax errors",
289               "contentType": "application/problem+json",
290               "htv:statusCodeValue": 400
291             }
292           ],
293           "scopes": "search"
294         }
295       ]
296     },
297     "searchXPath": {
298       "description": "XPath syntactic search",
299       "uriVariables": {
300         "query": {
301           "title": "A valid XPath expression",
302           "type": "string"
303         }
304       },
305       "safe": true,
306       "idempotent": true,
307       "forms": [
308         {
309           "href": "/search/xpath?query={query}",
310           "htv:methodName": "GET",
311           "response": {
312             "description": "Success response",
313             "contentType": "application/json",
314             "htv:statusCodeValue": 200
315           },
316           "additionalResponses": [
317             {
```

```
318                 "description": "XPath expression not provided or
    ↪  contains syntax errors",
319                 "contentType": "application/problem+json",
320                 "htv:statusCodeValue": 400
321               }
322             ],
323             "scopes": "search"
324           }
325         ]
326       },
327       "searchSPARQL": {
328         "description": "SPARQL semantic search",
329         "uriVariables": {
330           "query": {
331             "title": "A valid SPARQL 1.1. query",
332             "type": "string"
333           }
334         },
335         "safe": true,
336         "idempotent": true,
337         "forms": [
338           {
339             "href": "/search/sparql?query={query}",
340             "htv:methodName": "GET",
341             "response": {
342               "description": "Success response",
343               "htv:statusCodeValue": 200
344             },
345             "additionalResponses": [
346               {
347                 "description": "SPARQL query not provided or contains
    ↪  syntax errors",
348                 "contentType": "application/problem+json",
349                 "htv:statusCodeValue": 400
350               }
351             ],
352             "scopes": "search"
353           },
354           {
355             "href": "/search/sparql",
356             "htv:methodName": "POST",
357             "response": {
358               "description": "Success response",
359               "contentType": "application/json",
360               "htv:statusCodeValue": 200
361             },
362             "additionalResponses": [
363               {
```

```
364                 "description": "SPARQL query not provided or contains
    ↪  syntax errors",
365                 "contentType": "application/problem+json",
366                 "htv:statusCodeValue": 400
367               }
368             ],
369             "scopes": "search"
370           }
371         ]
372       }
373     },
374     "events": {
375       "thingCreation": {
376         "description": "Registration of Thing Descriptions inside the
    ↪  directory",
377         "uriVariables": {
378           "diff": {
379             "description": "Receive the full created TD as event data",
380             "type": "boolean"
381           }
382         },
383         "data": {
384           "title": "Partial/Full TD",
385           "type": "object"
386         },
387         "forms": [
388           {
389             "op": "subscribeevent",
390             "href": "/events/create{?diff}",
391             "subprotocol": "sse",
392             "contentType": "text/event-stream",
393             "htv:headers": [
394               {
395                 "description": "ID of the last event for reconnection",
396                 "htv:fieldName": "Last-Event-ID"
397               }
398             ],
399             "scopes": "notification"
400           }
401         ]
402       },
403       "thingUpdate": {
404         "description": "Updates to Thing Descriptions within the
    ↪  directory",
405         "uriVariables": {
406           "diff": {
407             "description": "Include TD changes inside event data",
408             "type": "boolean"
409           }
```

```
      },
      "data": {
        "title": "Partial TD",
        "type": "object",
        "contentMediaType": "application/merge-patch+json"
      },
      "forms": [
        {
          "op": "subscribeevent",
          "href": "/events/update{?diff}",
          "subprotocol": "sse",
          "contentType": "text/event-stream",
          "htv:headers": [
            {
              "description": "ID of the last event for reconnection",
              "htv:fieldName": "Last-Event-ID"
            }
          ],
          "scopes": "notification"
        }
      ]
    },
    "thingDeletion": {
      "description": "Deletion of Thing Descriptions from the
  ↪ directory",
      "data": {
        "title": "Partial TD",
        "type": "object"
      },
      "forms": [
        {
          "op": "subscribeevent",
          "href": "/events/delete",
          "subprotocol": "sse",
          "contentType": "text/event-stream",
          "htv:headers": [
            {
              "description": "ID of the last event for reconnection",
              "htv:fieldName": "Last-Event-ID"
            }
          ],
          "scopes": "notification"
        }
      ]
    }
  },
  "@id": "http://localhost:9000/.well-known/wot-thing-description"
}
```

B.2 Discovery Service

TD B.2: Discovery Service's TD

```
{
    "@context": [
        "https://www.w3.org/2019/wot/td/v1",
        {
            "ec": "http://www.tuwien.ac.at/autosys/leonhard-esterbau
↪ er/energy-community-architecture",
        }
    ],
    "@type": [
        "Thing",
        "ec:discoveryService"
    ],
    "title": "Discovery Service",
    "description": "Recognizes the DC Power property of Modbus TCP
↪ devices that implement SunSpec model 101",
    "id": "f4eaea7d-6cdb-49c9-a825-a796bc044bd0",
    "securityDefinitions": {
        "bearer_sc": {
            "scheme": "bearer",
            "in":"header",
            "format": "jwt",
            "alg": "RS256",
            "authorization":
↪ "https://10.10.0.8:8080/.well-known/wot-thing-description",
            "ec:scopes": ["admin"]
        }
    },
    "security": ["bearer_sc"],
    "base": "https://10.10.0.9:8090",
    "properties": {
        "state": {
            "@type": ["ec:state"],
            "type": "string",
            "description": "Discovering state (running, stopped)",
            "readOnly": true,
            "enum": [
                "running",
                "stopped"
            ],
            "forms": [{
                "href": "/properties/state",
                "contentType": "application/json",
                "op": ["readproperty"],
                "ec:scopes": ["admin"]
            }]
        }
```

```
44      },
45      "actions": {
46          "startDiscovery": {
47              "@type": ["ec:startDiscovery"],
48              "description": "Start discovering",
49              "forms": [{
50                  "href": "/actions/startDiscovery",
51                  "contentType": "application/json",
52                  "op": ["invokeaction"],
53                  "ec:scopes": ["admin"]
54              }]
55          },
56          "stopDiscovery": {
57              "@type": ["ec:stopDiscovery"],
58              "description": "Stop discovering",
59              "forms": [{
60                  "href": "/actions/stopDiscovery",
61                  "contentType": "application/json",
62                  "op": ["invokeaction"],
63                  "ec:scopes": ["admin"]
64              }]
65          }
66      }
67  }
```

B.3 Network Service

TD B.3: Network Service's TD

```
1  {
2      "@context": [
3          "https://www.w3.org/2019/wot/td/v1",
4          {
5              "ec": "http://www.tuwien.ac.at/autosys/leonhard-esterbau
   ↪  er/energy-community-architecture"
6          }
7      ],
8      "@type": [
9          "Thing",
10         "ec:networkService"
11     ],
12     "title": "Network Service",
13     "description": "Service that is responsible for connecting
   ↪  external parties.",
14     "id": "21da2b3f-2afa-4d2c-9b09-daba90eb75c7",
15     "securityDefinitions": {
16         "bearer_sc": {
17             "scheme": "bearer",
18             "in":"header",
```

```
19              "format": "jwt",
20              "alg": "RS256",
21              "authorization":
    ↪  "https://10.10.0.8:8080/.well-known/wot-thing-description",
22              "ec:scopes": ["observe", "control"]
23          }
24      },
25      "security": ["bearer_sc"],
26      "base": "https://10.10.0.2:8020",
27      "properties": {
28          "firewallRules": {
29              "@type": ["ec:firewallRules"],
30              "type": "array",
31              "items": {
32                  "@type": ["ec:firewallRule"],
33                  "type": "object",
34                  "properties": {
35                      "sourceIP": {
36                          "type": "string",
37                          "@type": ["ec:sourceIP"]
38                      },
39                      "destinationIP": {
40                          "type": "string",
41                          "@type": ["ec:destinationIP"]
42                      },
43                      "allowed": {
44                          "type": "boolean",
45                          "@type": ["ec:allowed"]
46                      }
47                  }
48              },
49              "readOnly": true,
50              "forms": [{
51                  "href": "/properties/firewallRules",
52                  "contentType": "application/json",
53                  "op": ["readproperty"],
54                  "ec:scopes": ["admin"]
55              }]
56          }
57      },
58      "actions": {
59          "setFirewallRule": {
60              "@type": ["ec:setFirewallRule"],
61              "description": "Endpoint for allowing or denying access
    ↪  to a thing's network interface",
62              "input": {
63                  "@type": ["ec:firewallRule"],
64                  "type": "object",
65                  "properties": {
```

```
                    "sourceIP": {
                        "type": "string",
                        "@type": ["ec:sourceIP"]
                    },
                    "destinationIP": {
                        "type": "string",
                        "@type": ["ec:destinationIP"]
                    },
                    "allowed": {
                        "type": "boolean",
                        "@type": ["ec:allowed"]
                    }
                }
            },
            "forms": [{
                "href": "/actions/setFirewallRule",
                "contentType": "application/json",
                "op": ["invokeaction"],
                "ec:scopes": ["admin"]
            }]
        }
    }
}
```

B.4 Runtime Service

TD B.4: Runtime Service's TD

```
{
    "@context": [
        "https://www.w3.org/2019/wot/td/v1",
        {
            "ec": "http://www.tuwien.ac.at/autosys/leonhard-esterbau
er/energy-community-architecture"
        }
    ],
    "@type": [
        "Thing",
        "ec:runtimeService"
    ],
    "title": "Runtime Service",
    "description": "Service that provides a NodeJS Runtime for
executing JavaScript code",
    "id": "c0daf98c-3311-4464-b4c7-a589a4d3e0ae",
    "securityDefinitions": {
        "bearer_sc": {
            "scheme": "bearer",
            "in":"header",
            "format": "jwt",
```

```
20              "alg": "RS256",
21              "authorization":
    ↪  "https://10.10.0.8:8080/.well-known/wot-thing-description",
22              "ec:scopes": ["admin"]
23          }
24      },
25      "security": ["bearer_sc"],
26      "base": "https://10.10.0.10:8100",
27      "properties": {
28          "runningScripts": {
29              "@type": ["ec:runningScripts"],
30              "type": "array",
31              "items": {
32                  "type": "object",
33                  "properties": {
34                      "scriptId": {
35                          "type": "string",
36                          "@type": ["ec:scriptId"]
37                      },
38                      "scriptName": {
39                          "type": "string",
40                          "@type": ["ec:scriptName"]
41                      }
42                  }
43              },
44              "description": "Lists running scripts and related
    ↪  properties such as name and id",
45              "readOnly": true,
46              "forms": [{
47                  "href": "/properties/runningScripts",
48                  "contentType": "application/json",
49                  "op": ["readproperty"],
50                  "ec:scopes": ["admin"]
51              }]
52          }
53      },
54      "actions": {
55          "startScript": {
56              "@type": ["ec:startScript"],
57              "description": "Runs JS code in a virtual environment",
58              "input": {
59                  "type": "object",
60                  "properties": {
61                      "script": {
62                          "type": "string",
63                          "@type": ["ec:jsCode"]
64                      },
65                      "scriptName": {
66                          "type": "string",
```

```
                        "@type": ["ec:scriptName"]
                    }
                }
            },
            "output": {
                "type": "object",
                "properties": {
                    "scriptId": {
                        "type": "string",
                        "@type": "ec:scriptId"
                    },
                    "scriptName": {
                        "type": "string",
                        "@type": "ec:scriptName"
                    }
                }
            },
            "forms": [{
                "href": "/actions/startScript",
                "contentType": "application/json",
                "op": ["invokeaction"],
                "ec:scopes": ["admin"]
            }]
        },
        "stopScript": {
            "@type": ["ec:stopScript"],
            "description": "Stops and removes a virtual environment",
            "input": {
                "@type": ["ec:scriptId"],
                "type": "string"
            },
            "forms": [{
                "href": "/actions/stopScript",
                "contentType": "application/json",
                "op": ["invokeaction"],
                "ec:scopes": ["admin"]
            }]
        }
    }
}
```

B.5 Security Service

TD B.5: Security Service's TD

```
{
    "@context": [
        "https://www.w3.org/2019/wot/td/v1",
        {
```

```
5             "ec": "http://www.tuwien.ac.at/autosys/leonhard-esterbau
  ↪ er/energy-community-architecture"
6         }
7     ],
8     "@type": [
9         "Thing",
10        "ec:securityService"
11    ],
12    "title": "Security Service",
13    "description": "Service that is responsible for device and
  ↪ service authorization",
14    "id": "6daf93df-25f0-414c-9adc-acccf640916a",
15    "securityDefinitions": {
16        "basic_sc": {
17            "scheme": "basic",
18            "in": "header",
19            "ec:scopes": ["admin"]
20        },
21        "bearer_sc": {
22            "scheme": "bearer",
23            "in":"header",
24            "format": "jwt",
25            "alg": "RS256",
26            "authorization":
  ↪ "https://10.10.0.8:8080/.well-known/wot-thing-description",
27            "ec:scopes": ["admin"]
28        }
29    },
30    "security": ["basic_sc", "bearer_sc"],
31    "base": "https://10.10.0.8:8080",
32    "properties": {
33        "pendingAccessRequests": {
34            "@type": ["ec:pendingAccessRequests"],
35            "type": "array",
36            "items": {
37                "@type": ["ec:accessEntry"],
38                "type": "object",
39                "properties": {
40                    "sourceId": {
41                        "type": "string",
42                        "@type": ["ec:sourceId"]
43                    },
44                    "targetId": {
45                        "type": "string",
46                        "@type": ["ec:targetId"]
47                    },
48                    "scopes": {
49                        "type": "array",
50                        "@type": ["ec:scopes"],
```

```
                            "items": {
                                "type": "string",
                                "@type": ["ec:scope"]
                            }
                        }
                    }
                },
                "readOnly": true,
                "forms": [{
                    "href": "/properties/pendingAccessRequests",
                    "contentType": "application/json",
                    "op": ["readproperty"],
                    "ec:scopes": ["admin"]
                }]
            },
            "grantedAccess": {
                "@type": ["ec:grantedAccess"],
                "type": "array",
                "items": {
                    "@type": ["ec:accessEntry"],
                    "type": "object",
                    "properties": {
                        "sourceId": {
                            "type": "string",
                            "@type": ["ec:sourceId"]
                        },
                        "targetId": {
                            "type": "string",
                            "@type": ["ec:targetId"]
                        },
                        "scopes": {
                            "type": "array",
                            "@type": ["ec:scopes"],
                            "items": {
                                "type": "string",
                                "@type": ["ec:scope"]
                            }
                        }
                    }
                },
                "readOnly": true,
                "forms": [{
                    "href": "/properties/grantedAccess",
                    "contentType": "application/json",
                    "op": ["readproperty"],
                    "ec:scopes": ["admin"]
                }]
            }
        },
```

```
100     "actions": {
101         "refuseAccess": {
102             "@type": ["ec:refuseAccess"],
103             "description": "Endpoint for refusing access to a
    ↪  thing's scope",
104             "input": {
105                 "@type": ["ec:accessEntry"],
106                 "type": "object",
107                 "properties": {
108                     "sourceId": {
109                         "type": "string",
110                         "@type": ["ec:sourceId"]
111                     },
112                     "targetId": {
113                         "type": "string",
114                         "@type": ["ec:targetId"]
115                     },
116                     "scopes": {
117                         "type": "array",
118                         "@type": ["ec:scopes"],
119                         "items": {
120                             "type": "string",
121                             "@type": ["ec:scope"]
122                         }
123                     }
124                 }
125             },
126             "forms": [{
127                 "href": "/actions/refuseAccess",
128                 "contentType": "application/json",
129                 "op": ["invokeaction"],
130                 "ec:scopes": ["admin"]
131             }]
132         },
133         "grantAccess": {
134             "@type": ["ec:grantAccess"],
135             "description": "Endpoint for granting access to a
    ↪  thing's scope",
136             "input": {
137                 "@type": ["ec:accessEntry"],
138                 "type": "object",
139                 "properties": {
140                     "sourceId": {
141                         "type": "string",
142                         "@type": ["ec:sourceId"]
143                     },
144                     "targetId": {
145                         "type": "string",
146                         "@type": ["ec:targetId"]
```

```
                    },
                    "scopes": {
                        "type": "array",
                        "@type": ["ec:scopes"],
                        "items": {
                            "type": "string",
                            "@type": ["ec:scope"]
                        }
                    }
                }
            },
            "forms": [{
                "href": "/actions/grantAccess",
                "contentType": "application/json",
                "op": ["invokeaction"],
                "ec:scopes": ["admin"]
            }]
        },
        "requestAccess": {
            "@type": ["ec:requestAccess"],
            "description": "Endpoint for requesting the user for access to a thing's scope",
            "input": {
                "@type": ["ec:accessEntry"],
                "type": "object",
                "properties": {
                    "sourceId": {
                        "type": "string",
                        "@type": ["ec:sourceId"]
                    },
                    "targetId": {
                        "type": "string",
                        "@type": ["ec:targetId"]
                    },
                    "scopes": {
                        "type": "array",
                        "@type": ["ec:scopes"],
                        "items": {
                            "type": "string",
                            "@type": ["ec:scope"]
                        }
                    }
                }
            },
            "output": {
                "type": "string",
                "@type": ["ec:AccessToken", "ec:JWT"]
            },
            "forms": [{
```

```
195                 "href": "/actions/requestAccess",
196                 "contentType": "application/json",
197                 "op": ["invokeaction"]
198             }]
199         },
200         "issueToken": {
201             "@type": ["ec:issueToken"],
202            "description": "Issues a JWT token that is valid for
    ↪  10min",
203             "input": {
204                 "type": "string",
205                 "@type": ["ec:AccessToken", "ec:JWT"]
206             },
207             "output": {
208                 "type": "string",
209                 "@type": ["ec:AccessToken", "ec:JWT"]
210             },
211             "forms": [{
212                 "href": "/actions/issueToken",
213                 "contentType": "application/json",
214                 "op": ["invokeaction"]
215             }]
216         }
217     },
218     "events": {
219         "newAccessRequest": {
220             "@type": ["ec:newAccessRequest"],
221             "data": {
222                 "@type": ["ec:accessEntry"],
223                 "type": "object",
224                 "properties": {
225                     "sourceId": {
226                         "type": "string",
227                         "@type": ["ec:sourceId"]
228                     },
229                     "targetId": {
230                         "type": "string",
231                         "@type": ["ec:targetId"]
232                     },
233                     "scopes": {
234                         "type": "array",
235                         "@type": ["ec:scopes"],
236                         "items": {
237                             "type": "string",
238                             "@type": ["ec:scope"]
239                         }
240                     }
241                 }
242             },
```

```
            "forms": [{
                "href": "/subscriptions/newAccessRequest",
                "subprotocol": "longpoll",
                "op": ["subscribeevent"],
                "ec:scopes": ["admin"]
            }]
        }
    }
}
```

B.6 Electric Vehicle charging station

TD B.6: EV charging station's TD

```
{
    "@context": [
        "https://www.w3.org/2019/wot/td/v1",
        {
            "ec": "http://www.tuwien.ac.at/autosys/leonhard-esterbau
↪ er/energy-community-architecture"
        }
    ],
    "@type": [
        "Thing",
        "ec:energyDevice",
        "ec:EVChargingStation"
    ],
    "title": "EV charging station",
    "description": "Simulated charging station for electric
↪ vehicles",
    "id": "2972af70-7059-41c7-b417-0be91ffb2d5f",
    "securityDefinitions": {
        "bearer_sc": {
            "scheme": "bearer",
            "in":"header",
            "format": "jwt",
            "alg": "RS256",
            "authorization":
↪ "https://10.10.0.8:8080/.well-known/wot-thing-description",
            "ec:scopes": ["observe", "control"]
        }
    },
    "security": ["bearer_sc"],
    "base": "https://10.10.0.6:8060",
    "properties": {
        "stateOfCharge": {
            "@type": ["ec:stateOfCharge"],
            "type": "number",
```

```
32              "description": "If an EV is connected, this property
    ↪  represents the current state of charge in % (0 ... 100%)",
33              "readOnly": true,
34              "minimum": 0,
35              "maximum": 100,
36              "unit": "%",
37              "forms": [{
38                  "href": "/properties/stateOfCharge",
39                  "contentType": "application/json",
40                  "op": ["readproperty"],
41                  "ec:scopes": ["observe"]
42              }]
43          },
44          "state": {
45              "@type": ["ec:state"],
46              "type": "string",
47              "description": "Charging status (readyToCharge,
    ↪  charging, disconnected)",
48              "readOnly": true,
49              "enum": [
50                  "readyToCharge",
51                  "charging",
52                  "disconnected"
53              ],
54              "forms": [{
55                  "href": "/properties/state",
56                  "contentType": "application/json",
57                  "op": ["readproperty"],
58                  "ec:scopes": ["observe"]
59              }]
60          }
61      },
62      "actions": {
63          "startCharging": {
64              "@type": ["ec:startCharging"],
65              "description": "Start charging",
66              "forms": [{
67                  "href": "/actions/startCharging",
68                  "contentType": "application/json",
69                  "op": ["invokeaction"],
70                  "ec:scopes": ["control"]
71              }]
72          },
73          "stopCharging": {
74              "@type": ["ec:stopCharging"],
75              "description": "Stop charging",
76              "forms": [{
77                  "href": "/actions/stopCharging",
78                  "contentType": "application/json",
```

```
79                  "op": ["invokeaction"],
80                  "ec:scopes": ["control"]
81              }]
82          }
83      }
84  }
```

B.7 Legacy inverter

TD B.7: Inverter's TD

```
1  {
2      "@context": [
3          "https://www.w3.org/2019/wot/td/v1",
4          {
5              "ec": "http://www.tuwien.ac.at/autosys/leonhard-esterbau
   ↪  er/energy-community-architecture",
6              "sunspec": "http://www.sunspec.org"
7          }
8      ],
9      "@type": [
10         "Thing",
11         "sunspec:inverter"
12     ],
13     "title": "SunSpec Modbus inverter",
14     "description": "Inverter providing a Modbus TCP interface that
   ↪  implements the SunSpec specification",
15     "id": "6f1d8fa6-c327-4a2a-8784-51b3da260a39",
16     "securityDefinitions": {
17         "nosec_sc": {
18             "scheme": "nosec"
19         }
20     },
21     "security": "nosec_sc",
22     "base": "modbus+tcp://10.10.0.7:502",
23     "sunspec:models": ["101"],
24     "properties": {
25         "DCPower": {
26             "@type": ["sunspec:value"],
27             "type": "int",
28             "forms": [
29                 {
30                     "href": "/1",
31                     "op": ["readproperty"],
32                     "modbus:function": "readCoil",
33                     "modbus:address": 40102
34                 }
35             ],
36             "sunspec:desc": "DC Power",
```

```
            "sunspec:label": "DC Watts",
            "sunspec:name": "DCW",
            "sunspec:sf": "DCW_SF",
            "sunspec:size": 1,
            "sunspec:type": "int16",
            "sunspec:units": "W"
        },
        "DCScalingFactor": {
            "@type": ["sunspec:value"],
            "type": "int",
            "forms": [
                {
                    "href": "/1",
                    "op": ["readproperty"],
                    "modbus:function": "readCoil",
                    "modbus:address": 40103
                }
            ],
            "sunspec:name": "DCW_SF",
            "sunspec:size": 1,
            "sunspec:type": "sunssf"
        }
    }
}
```

B.8 SunSpec Modbus gateway

TD B.8: SunSpec Modbus Gateway's TD

```
{
    "@context": [
        "https://www.w3.org/2019/wot/td/v1",
        {
            "ec": "http://www.tuwien.ac.at/autosys/leonhard-esterbau
↪ er/energy-community-architecture"
        }
    ],
    "@type": [
        "Thing",
        "ec:gateway",
        "ec:energyDevice",
        "ec:inverter"
    ],
    "title": "SunSpec inverter gateway",
    "description": "Gateway for transforming actual SunSpec DC power
↪ from Modbus to HTTP and performs protocol specific adjustments",
    "id": "f02b05c3-a7f9-40cb-9c5a-8621a027e6e7",
    "securityDefinitions": {
        "bearer_sc": {
```

```
19              "scheme": "bearer",
20              "in":"header",
21              "format": "jwt",
22              "alg": "RS256",
23              "authorization":
    ↪  "https://10.10.0.8:8080/.well-known/wot-thing-description",
24              "ec:scopes": ["observe"]
25          }
26      },
27      "security": ["bearer_sc"],
28      "base": "https://10.10.0.10:8101",
29      "properties": {
30          "DCPower": {
31              "@type": ["ec:DCPower"],
32              "type": "number",
33              "description": "Actual DC power of the inverter in
    ↪  Watts",
34              "readOnly": true,
35              "unit": "W",
36              "forms": [{
37                  "href": "/properties/DCPower",
38                  "contentType": "application/json",
39                  "op": ["readproperty"],
40                  "ec:scopes": ["observe"]
41              }]
42          }
43      }
44  }
```

List of Figures

List of Listings

Acronyms

ADU Application Data Unit. 20

AI Artificial Intelligence. 26, 82

API Application Programming Interface. 13, 18, 28, 49, 57, 64, 68, 69, 70, 76, 83

CBOR Concise Binary Object Representation. 58

CDL Contract Description Language. 28

CEC Citizens Energy Community. 10, 11

CoAP Constrained Application Protocol. 3, 22, 23, 25, 26, 57

CoRE Constrained RESTful Environments. 18, 22

DC Direct Current. 61, 72

DD Device Description. 40, 41, 44, 48, 50, 51, 52, 53, 54, 55, 56, 57, 76, 82, 83

DER Distributed Energy Resources. 20

DHCP Dynamic Host Configuration Protocol. 37, 50, 52

DID Decentralized Identifiers. 18, 82

DMZ Demilitarized Zone. 40

DNS Domain Name System. 46

DNS-SD Domain Name System Service Discovery. 58, 67, 77

DPWS Devices Profile for Web Services. 26

DR Demand Response. 2, 12, 26

DSM Demand Side Management. 2, 26, 27

DSRM Design Science Research Methodology. 5, 6, 113

EAG *Erneuerbaren-Ausbau-Gesetz*. 1, 9, 10, 29

EC Energy Community. 1, 3, 4, 5, 6, 9, 11, 19, 28, 29, 30, 31, 32, 33, 34, 35, 37, 39, 45, 47, 61, 66, 75, 76, 77, 78, 79, 81, 82, 85, 115

EIA Electronic Industries Alliance. 20

ElWOG *Elektrizitätswirtschafts- und -organisationsgesetz*. 1, 9, 29

EMS Energy Management System. 2, 29, 30, 33, 34, 35

EU European Union. 9, 11

EV Electric Vehicle. 12, 37, 39, 40, 48, 49, 52, 55, 64, 66, 70, 71, 75, 82, 108, 113, 115

GDPR General Data Protection Regulation. 33

GHG Greenhouse Gas. 2, 9

HEMS Home Energy Management System. 27

HTTP Hypertext Transfer Protocol. 3, 19, 21, 22, 23, 26, 28, 48, 49, 57, 58, 62, 66, 67, 68, 72, 76, 79, 81, 113

HTTPS HyperText Transfer Protocol Secure. 63

ICT Information and Communication Technology. 3

ID Identifier. 52, 64

IETF Internet Engineering Task Force. 21, 22, 58, 59

IFTT If This Then That. 26

IoT Internet of Things. 3, 6, 12, 13, 15, 17, 18, 19, 21, 22, 24, 25, 26, 27, 46, 57, 58, 60, 75, 79, 82, 83

IP Internet Protocol. 21, 24, 37, 38, 46, 47, 50, 52, 55, 56, 61, 62, 63, 76, 77, 78, 79

ISO International Organization for Standardization. 24

ISP Internet Service Provider. 27, 37, 44

JS JavaScript. 60, 61, 69, 83

JSON JavaScript Object Notation. 19, 21, 23, 25, 58, 64, 72

JSON-LD JavaScript Object Notation for Linked Data. 16, 25

JWT JSON Web Token. 59, 62, 64, 66, 67, 70, 76, 79, 81

LAN Local Area Network. 24, 45

LoRaWAN Long Range Wide Area Network. 37

LwM2M OMA Lightweight M2M. 56

MBAP Modbus Application Protocol. 20

MCU Microcontroller. 60

mDNS Multicast DNS. 18, 58, 67, 77, 81

MQTT Message Queuing Telemetry Transport. 3, 4, 19, 22, 23, 25, 26, 28, 57

NAT Network Address Translation. 37

NFV Network Function Virtualization. 27

oneM2M one Machine to Machine. 57

OSGi Open Services Gateway initiative. 60

OSI Open Systems Interconnection model. 24, 113

OWL Web Ontology Language. 23

P2P Peer-to-Peer. 1, 2, 3, 4, 59

PDU Protocol Data Unit. 20

PLC Power Line Communication. 27

PV Photovoltaic. 12

QoS Quality of Service. 19

RDF Resource Description Framework. 25, 57

REC Renewable Energy Community. 10, 11

REST Representational State Transfer. 18, 21, 22, 25, 26

RFC Request for Comments. 21

RL Reinforcement Learning. 12

SAREF Smart Applications Reference. 57

SDN Software Defined Networking. 27

SensorML Sensor Model Language. 56

SG Smart Grid. 4, 19, 25

SME Small and Medium Enterprise. 46

SOA Service Oriented Architecture. 4, 28

SOAP Simple Object Access Protocol. 21

SPARQL SPARQL Protocol and RDF Query Language. 18, 26, 69

SSDP Simple Service Discovery Protocol. 58

SSN Semantic Sensor Network. 57

SWoT Semantic Web of Things. 13

TCP Transmission Control Protocol. 37, 38, 46, 67, 77

TD Thing Description. 16, 17, 18, 19, 57, 58, 59, 62, 63, 64, 66, 67, 68, 69, 70, 72, 73, 76, 78, 82, 83, 85, 86, 87, 97, 98, 100, 102, 108, 110, 111, 115

TDD Thing Description Directory. 18

TIA Telecommunication Industry Association. 20

TLS Transport Layer Security. 57

TPSP Third Party Service Provider. 33, 34, 38

UDP User Datagram Protocol. 22, 37, 38, 46, 77

UI User Interface. 44, 62

UML Unified Modeling Language. 30, 31, 113

UPnP Universal Plug and Play. 46, 58

URI Uniform Resource Identifier. 18, 22, 48, 58, 64, 67

VLAN Virtual Local Area Network. 25

VM Virtual Machine. 60

VPN Virtual Private Network. 25, 59

W3C World Wide Web Consortium. 13, 14, 18, 19, 21, 57, 61, 69, 82

WAN Wide Area Network. 45, 46

WASI WebAssembly System Interface. 60

WG Working Group. 14, 22

WIDL Web Interface Definition Language. 18

WoE Web of Energy. 25

WoT Web of Things. 9, 13, 14, 15, 16, 17, 18, 19, 25, 38, 39, 41, 56, 57, 58, 60, 61, 68, 69, 70, 82, 83, 113

WSDL Web Services Description Language. 21

XML Extensible Markup Language. 21, 23, 25, 28, 58

Bibliography

[1] National Council of Austria, *Bundesgesetz, mit dem die Organisation auf dem Gebiet der Elektrizitätswirtschaft neu geregelt wird (Elektrizitätswirtschafts- und -organisationsgesetz 2010 – ElWOG 2010)*. [Online]. Available: https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=Bundesnormen&Gesetzesnummer=20007045 (visited on 02/24/2022).

[2] National Council of Austria, *Bundesgesetz über den Ausbau von Energie aus erneuerbaren Quellen (Erneuerbaren-Ausbau-Gesetz – EAG)*. [Online]. Available: https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=Bundesnormen&Gesetzesnummer=20011619 (visited on 02/24/2022).

[3] C. Long, J. Wu, C. Zhang, M. Cheng, and A. Al-Wakeel, "Feasibility of Peer-to-Peer Energy Trading in Low Voltage Electrical Distribution Networks", *Energy Procedia*, vol. 105, pp. 2227–2232, 2017, 8th International Conference on Applied Energy, ICAE2016, 8-11 October 2016, Beijing, China. DOI: https://doi.org/10.1016/j.egypro.2017.03.632.

[4] C. Long, J. Wu, C. Zhang, L. Thomas, M. Cheng, and N. Jenkins, "Peer-to-peer energy trading in a community microgrid", in *2017 IEEE Power Energy Society General Meeting*, 2017, pp. 1–5. DOI: 10.1109/PESGM.2017.8274546.

[5] *Electricity prices for household consumers - bi-annual data (from 2007 onwards)*, Eurostat-Code: nrg_pc_204. [Online]. Available: http://appsso.eurostat.ec.europa.eu/nui/show.do?dataset=nrg_pc_204 (visited on 03/16/2022).

[6] H. Ritchie and M. Roser, "CO_2 and Greenhouse Gas Emissions", *Our World in Data*, 2020, https://ourworldindata.org/co2-and-other-greenhouse-gas-emissions.

[7] M. R. via Our World in Data, *The world's energy problem*, Dec. 10, 2020. [Online]. Available: https://ourworldindata.org/worlds-energy-problem (visited on 03/16/2022).

[8] T. Logenthiran, D. Srinivasan, and T. Z. Shun, "Demand Side Management in Smart Grid Using Heuristic Optimization", *IEEE Transactions on Smart Grid*, vol. 3, no. 3, pp. 1244–1252, 2012. DOI: 10.1109/TSG.2012.2195686.

[9] P. Siano, G. Graditi, M. Atrigna, and A. Piccolo, "Designing and testing decision support and energy management systems for smart homes", *Journal of Ambient Intelligence and Humanized Computing*, vol. 4, no. 6, pp. 651–661, 2013. DOI: 10.1007/s12652-013-0176-9.

[10] J. Han, C.-s. Choi, W.-k. Park, I. Lee, and S.-h. Kim, "Smart home energy management system including renewable energy based on ZigBee and PLC", *IEEE Transactions on Consumer Electronics*, vol. 60, no. 2, pp. 198–202, 2014. DOI: 10.1109/TCE.2014.6851994.

[11] B. Celik, R. Roche, D. Bouquain, and A. Miraoui, "Decentralized Neighborhood Energy Management With Coordinated Smart Home Energy Sharing", *IEEE Transactions on Smart Grid*, vol. 9, no. 6, pp. 6387–6397, 2018. DOI: 10.1109/TSG.2017.2710358.

[12] C. Zhang, J. Wu, C. Long, and M. Cheng, "Review of Existing Peer-to-Peer Energy Trading Projects", *Energy Procedia*, vol. 105, pp. 2563–2568, 2017, 8th International Conference on Applied Energy, ICAE2016, 8-11 October 2016, Beijing, China, ISSN: 1876-6102. DOI: https://doi.org/10.1016/j.egypro.2017.03.737.

[13] B. P. Esther and K. S. Kumar, "A survey on residential Demand Side Management architecture, approaches, optimization models and methods", *Renewable and Sustainable Energy Reviews*, vol. 59, pp. 342–351, 2016, ISSN: 1364-0321. DOI: https://doi.org/10.1016/j.rser.2015.12.282.

[14] P. Siano, "Demand response and smart grids—A survey", *Renewable and Sustainable Energy Reviews*, vol. 30, pp. 461–478, 2014, ISSN: 1364-0321. DOI: https://doi.org/10.1016/j.rser.2013.10.022.

[15] P. Pawar and P. Vittal K, "Design and development of advanced smart energy management system integrated with IoT framework in smart grid environment", *Journal of Energy Storage*, vol. 25, p. 100 846, 2019, ISSN: 2352-152X. DOI: https://doi.org/10.1016/j.est.2019.100846.

[16] F. Luo, G. Ranzi, C. Wan, Z. Xu, and Z. Y. Dong, "A Multistage Home Energy Management System With Residential Photovoltaic Penetration", *IEEE Transactions on Industrial Informatics*, vol. 15, no. 1, pp. 116–126, 2019. DOI: 10.1109/TII.2018.2871159.

[17] S. Li, J. Yang, W. Song, and A. Chen, "A Real-Time Electricity Scheduling for Residential Home Energy Management", *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2602–2611, 2019. DOI: 10.1109/JIOT.2018.2872463.

[18] X. Xu, Y. Jia, Y. Xu, Z. Xu, S. Chai, and C. S. Lai, "A Multi-Agent Reinforcement Learning-Based Data-Driven Method for Home Energy Management", *IEEE Transactions on Smart Grid*, vol. 11, no. 4, pp. 3201–3211, 2020. DOI: 10.1109/TSG.2020.2971427.

[19] K. Mahmud, M. J. Hossain, and G. E. Town, "Peak-Load Reduction by Coordinated Response of Photovoltaics, Battery Storage, and Electric Vehicles", *IEEE Access*, vol. 6, pp. 29 353–29 365, 2018. DOI: 10.1109/ACCESS.2018.2837144.

[20] E. Mocanu, D. C. Mocanu, P. H. Nguyen, *et al.*, "On-Line Building Energy Optimization Using Deep Reinforcement Learning", *IEEE Transactions on Smart Grid*, vol. 10, no. 4, pp. 3698–3708, 2019. DOI: 10.1109/TSG.2018.2834219.

[21] L. Zhang, J. Wen, Y. Li, *et al.*, "A review of machine learning in building load prediction", *Applied Energy*, vol. 285, p. 116 452, 2021, ISSN: 0306-2619. DOI: https://doi.org/10.1016/j.apenergy.2021.116452.

[22] A.-D. Pham, N.-T. Ngo, T. T. Ha Truong, N.-T. Huynh, and N.-S. Truong, "Predicting energy consumption in multiple buildings using machine learning for improving energy efficiency and sustainability", *Journal of Cleaner Production*, vol. 260, p. 121 082, 2020, ISSN: 0959-6526. DOI: https://doi.org/10.1016/j.jclepro.2020.121082.

[23] C.-J. Huang and P.-H. Kuo, "Multiple-Input Deep Convolutional Neural Network Model for Short-Term Photovoltaic Power Forecasting", *IEEE Access*, vol. 7, pp. 74 822–74 834, 2019. DOI: 10.1109/ACCESS.2019.2921238.

[24] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey", *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2010.05.010.

[25] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications", *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1125–1142, 2017. DOI: 10.1109/JIOT.2017.2683200.

[26] J. Lloret, J. Tomas, A. Canovas, and L. Parra, "An Integrated IoT Architecture for Smart Metering", *IEEE Communications Magazine*, vol. 54, no. 12, pp. 50–57, 2016. DOI: 10.1109/MCOM.2016.1600647CM.

[27] *Loxone homepage*. [Online]. Available: https://www.loxone.com (visited on 03/21/2022).

[28] *Homematic IP homepage*. [Online]. Available: https://homematic-ip.com (visited on 03/21/2022).

[29] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications", *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015. DOI: 10.1109/COMST.2015.2444095.

[30] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A Design Science Research Methodology for Information Systems Research", *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45–77, 2007. DOI: 10.2753/MIS0742-1222240302. eprint: https://doi.org/10.2753/MIS0742-1222240302.

[31] M. Cohn, *User Stories Applied: For Agile Software Development*. USA: Addison Wesley Longman Publishing Co., Inc., 2004, ISBN: 0321205685.

[32] *FFG Project RES² Community*. [Online]. Available: https://projekte.ffg.at/projekt/4014638 (visited on 04/09/2022).

[33] E. Parliament, *EU directive 2018/2001 of the European Parliament and the council of 11 December 2018 on the on the promotion of the use of energy from renewable sources*. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32018L2001 (visited on 02/22/2022).

[34] E. Parliament, *EU directive 2019/944 of the European Parliament and the council of 5 June 2019 on common rules for the internal market for electricity and amending Directive 2012/27/EU*. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32019L0944 (visited on 02/22/2022).

[35] E-Control Austria, *Erläuterungen*. [Online]. Available: https://www.e-control.at/documents/1785851/1811582/SNE-V_2te-Novelle_2021_Erlaeuterungen.pdf (visited on 02/24/2022).

[36] Y.-Y. Chen, M.-H. Chen, C.-M. Chang, F.-S. Chang, and Y.-H. Lin, "A Smart Home Energy Management System Using Two-Stage Non-Intrusive Appliance Load Monitoring over Fog-Cloud Analytics Based on Tridium's Niagara Framework for Residential Demand-Side Management", *Sensors*, vol. 21, no. 8, 2021, ISSN: 1424-8220. DOI: 10.3390/s21082883.

[37] Y.-H. Lin, "Design and Implementation of an IoT-Oriented Energy Management System Based on Non-Intrusive and Self-Organizing Neuro-Fuzzy Classification as an Electrical Energy Audit in Smart Homes", *Applied Sciences*, vol. 8, no. 12, 2018, ISSN: 2076-3417. DOI: 10.3390/app8122337.

[38] *ntuity*. [Online]. Available: https://ntuity.io/en/ (visited on 03/26/2022).

[39] *GreenCom Networks*. [Online]. Available: https://www.greencom-networks.com/en (visited on 03/26/2022).

[40] A. J. Jara, A. Olivieri, Y. Bocchi, M. Jung, W. Kastner, and A. Skarmeta, "Semantic Web of Things: an analysis of the application semantics for the IoT moving towards the IoT convergence", *International Journal of Web and Grid Services*, vol. 10, pp. 244–272, Apr. 2014. DOI: 10.1504/IJWGS.2014.060260.

[41] D. Guinard and V. Trifa, "Towards the Web of Things: Web Mashups for Embedded Devices", in *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences)*, Madrid, Spain, Apr. 2009.

[42] D. Guinard, "A Web of Things Application Architecture – Integrating the Real-World into the Web", Ph.D. dissertation, ETH Zurich, Zurich, Switzerland, Aug. 2011.

[43] V. Trifa, D. Guinard, and D. Carrera, "Web Thing Model - W3C Member Submission 24 August 2015", W3C, Tech. Rep. [Online]. Available: https://www.w3.org/Submission/wot-model/ (visited on 02/22/2022).

[44] *W3C Begins Standards Work on Web of Things to Reduce IoT Fragmentation.* [Online]. Available: https://www.w3.org/2017/02/media-advisory-wot-wg.html.en (visited on 01/29/2022).

[45] M. Kovatsch, R. Matsukura, M. Lagally, T. Kawaguchi, K. Toumura, and K. Kajimoto, "Web of Things (WoT) Architecture - W3C Recommendation 9 April 2020", W3C, Standard. [Online]. Available: https://www.w3.org/TR/wot-architecture/ (visited on 02/22/2022).

[46] S. Kaebisch, T. Kamiya, M. McCool, V. Charpenay, and M. Kovatsch, "Web of Things (WoT) Thing Description - W3C Recommendation 9 April 2020", W3C, Standard. [Online]. Available: https://www.w3.org/TR/wot-thing-description/ (visited on 02/22/2022).

[47] M. Koster and E. Korkan, "Web of Things (WoT) Binding Templates - W3C Working Group Note 30 January 2020", W3C, Standard. [Online]. Available: https://www.w3.org/TR/wot-binding-templates/ (visited on 02/22/2022).

[48] Z. Kis, D. Peintner, C. Aguzzi, J. Hund, and K. Nimura, "Web of Things (WoT) Scripting API - W3C Working Group Note 24 November 2020", W3C, Standard. [Online]. Available: https://www.w3.org/TR/wot-scripting-api/ (visited on 02/22/2022).

[49] "Web Interface Definition Language (WIDL)", W3C, Standard. [Online]. Available: https://www.w3.org/TR/widl/ (visited on 05/05/2022).

[50] E. Reshetova and M. McCool, "Web of Things (WoT) Security and Privacy Guidelines - W3C Working Group Note 6 November 2019", W3C, Standard. [Online]. Available: https://www.w3.org/TR/wot-security/ (visited on 02/22/2022).

[51] A. Cimmino, M. McCool, F. Tavakolizadeh, and K. Toumura, "Web of Things (WoT) Discovery - W3C Working Draft 2 June 2021", W3C, Standard. [Online]. Available: https://www.w3.org/TR/wot-discovery/ (visited on 02/22/2022).

[52] G. Normington, E. Surov, M. Mikulicic, and S. Gössner, "JavaScript Object Notation (JSON) Path", IETF, Standard. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-jsonpath-base-00 (visited on 05/16/2022).

[53] J. Robie, M. Dyck, and J. Spiegel, "XML Path Language (XPath) 3.1", W3C, Standard. [Online]. Available: https://www.w3.org/TR/xpath-31/ (visited on 05/16/2022).

[54] The W3C SPARQL Working Group, "SPARQL 1.1 Overview", W3C, Standard. [Online]. Available: https://www.w3.org/TR/sparql11-overview/ (visited on 05/16/2022).

[55] S. Cheshire and M. Krochmal, "Multicast DNS", IETF, Standard. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6762 (visited on 05/05/2022).

[56] Z. Shelby, "Constrained RESTful Environments (CoRE) Link Format", IETF, Standard. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6690 (visited on 05/16/2022).

[57] M. Nottingham, "Well-Known Uniform Resource Identifiers (URIs)", IETF, Standard. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8615 (visited on 05/05/2022).

[58] M. Sporny, D. Longley, M. Sabadello, D. Reed, O. Steele, and C. Allen, "Decentralized Identifiers (DIDs) v1.0", W3C, Standard. [Online]. Available: https://www.w3.org/TR/did-core/ (visited on 05/03/2022).

[59] M. Lagally, M. McCool, R. Matsukura, S. Kaebisch, and T. Mizushima, "Web of Things (WoT) Profile - W3C First Public Working Draft 24 November 2020", W3C, Standard. [Online]. Available: https://www.w3.org/TR/wot-profile/ (visited on 02/22/2022).

[60] M. Faheem, S. Shah, R. Butt, *et al.*, "Smart grid communication and information technologies in the perspective of Industry 4.0: Opportunities and challenges", *Computer Science Review*, vol. 30, pp. 1–30, 2018. DOI: https://doi.org/10.1016/j.cosrev.2018.08.001.

[61] IEC 61850–7-1:2011, "Communication networks and systems for power utility automation - Part 7-1: Basic communication structure - Principles and models", IEC, Standard.

[62] IEEE2030.5-2018, "IEEE standard for smart energy profile application protocol", Standard, 2018, pp. 1–361. DOI: 10.1109/IEEESTD.2018.8608044.

[63] "SunSpec Technology Overview", Spec Alliance, Tech. Rep., Mar. 2015.

[64] "MODBUS Application Protocol Specification V1.1b3", Modbus Organization, Standard, Apr. 2012. [Online]. Available: https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf (visited on 08/05/2022).

[65] "MODBUS Messaging on TCP/IP Implementation Guide V1.0b ", Modbus Organization, Tech. Rep., Oct. 2006. [Online]. Available: https://modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf (visited on 08/05/2022).

[66] IEEE1547-2018, "Ieee standard for interconnection and interoperability of distributed energy resources with associated electric power systems interfaces", Standard, 2018, pp. 1–138. DOI: 10.1109/IEEESTD.2018.8332112.

[67] TIA/EIA-232-F, "Interface between data terminal equipment and data circuit-terminating equipment employing serial binary data interchange", Electronic Industries Alliance and Telecommunication Industry Association, Standard, Oct. 1997.

[68] TIA/EIA-485-A, "Electrical characteristics of generators and receivers for use in balanced digital multipoint systems", Electronic Industries Alliance and Telecommunication Industry Association, Standard, Mar. 1998.

[69] R. Fielding, U. Irvine, J. Gettys, *et al.*, "Hypertext Transfer Protocol – HTTP/1.1", Network Working Group, Standard. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2616 (visited on 05/05/2022).

[70] M. Gudgin, M. Hadley, N. Mendelsohn, *et al.*, "SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)", W3C, Standard. [Online]. Available: https://www.w3.org/TR/soap12/ (visited on 02/22/2022).

[71] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana, "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language", W3C, Standard. [Online]. Available: https://www.w3.org/TR/wsdl/ (visited on 02/22/2022).

[72] R. T. Fielding, "REST: architecture Styles and the Design of Network-based Software Architectures", Ph.D. dissertation, University of California, Irvine, 2000. [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[73] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)", IETF, Standard. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7252 (visited on 05/05/2022).

[74] A. Banks and R. Gupta, "MQTT Version 3.1.1", OASIS, Standard. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html (visited on 05/05/2022).

[75] C. Paniagua, J. Eliasson, and J. Delsing, "Interoperability Mismatch Challenges in Heterogeneous SOA-based Systems", in *2019 IEEE International Conference on Industrial Technology (ICIT)*, 2019, pp. 788–793. DOI: 10.1109/ICIT.2019.8754991.

[76] M. Botts, A. Robin, and E. Hirschorn, "OGC SensorML: Model and XML Encoding Standard", Open Geospatial Consortium, Standard. [Online]. Available: https://docs.ogc.org/is/12-000r2/12-000r2.html (visited on 05/05/2022).

[77] D. L. McGuinness and F. van Harmelen, "OWL Web Ontology Language Overview", W3C, Standard. [Online]. Available: https://www.w3.org/TR/owl-features/ (visited on 02/22/2022).

[78] OSI model, "Information technology – Open Systems Interconnection – Basic Reference Model: The basic model", Internatioinal Telecommunication Union, Tech. Rep., Jul. 1994. [Online]. Available: https://handle.itu.int/11.1002/1000/2820 (visited on 02/22/2022).

[79] V. Caballero, D. Vernet, A. Zaballos, and G. Corral, "Prototyping a Web-of-Energy Architecture for Smart Integration of Sensor Networks in Smart Grids Domain", *Sensors*, vol. 18, no. 2, 2018. DOI: 10.3390/s18020400.

[80] P. Desai, A. Sheth, and P. Anantharam, "Semantic Gateway as a Service Architecture for IoT Interoperability", in *2015 IEEE International Conference on Mobile Services*, IEEE, Jun. 2015. DOI: 10.1109/mobserv.2015.51.

[81] S.-M. Kim, H.-S. Choi, and W.-S. Rhee, "IoT home gateway for auto-configuration and management of MQTT devices", in *2015 IEEE Conference on Wireless Sensors (ICWiSe)*, 2015, pp. 12–17. DOI: 10.1109/ICWISE.2015.7380346.

[82] Y.-Y. Chen, Y.-H. Lin, C.-C. Kung, M.-H. Chung, and I.-H. Yen, "Design and Implementation of Cloud Analytics-Assisted Smart Power Meters Considering Advanced Artificial Intelligence as Edge Analytics in Demand-Side Management for Smart Homes", *Sensors*, vol. 19, no. 9, 2019. DOI: 10.3390/s19092047.

[83] M. H. Y. Moghaddam and A. Leon-Garcia, "A fog-based internet of energy architecture for transactive energy management systems", *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1055–1069, Apr. 2018. DOI: 10.1109/jiot.2018.2805899.

[84] L. Atzori, J. Bellido, R. Bolla, *et al.*, "SDN&NFV contribution to IoT objects virtualization", *Computer Networks*, vol. 149, pp. 200–212, 2019, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2018.11.030.

[85] F. G. Brundu, E. Patti, A. Osello, *et al.*, "IoT Software Infrastructure for Energy Management and Simulation in Smart Cities", *IEEE Transactions on Industrial Informatics*, vol. 13, no. 2, pp. 832–840, 2017. DOI: 10.1109/TII.2016.2627479.

[86] H. Derhamy, J. Eliasson, and J. Delsing, "IoT Interoperability—On-Demand and Low Latency Transparent Multiprotocol Translator", *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1754–1763, 2017. DOI: 10.1109/JIOT.2017.2697718.

[87] E. Parliament, *EU regulation 2016/679 of the European Parliament and the council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)*. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679 (visited on 03/18/2022).

[88] P. Wegner, "Interoperability", *ACM Comput. Surv.*, vol. 28, no. 1, pp. 285–287, Mar. 1996. DOI: 10.1145/234313.234424.

[89] "Requirements for internet hosts – communication layers", Tech. Rep. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc1122 (visited on 04/09/2022).

[90] O. Novo and M. Francesco, "Semantic Interoperability in the IoT: Extending the Web of Things Architecture", *ACM Transactions on Internet of Things*, vol. 1, pp. 1–25, Mar. 2020. DOI: 10.1145/3375838.

[91] I. Fette and A. Melnikov, "The WebSocket Protocol", IETF, Standard. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6455 (visited on 05/11/2022).

[92] E. D. Hardt, "The OAuth 2.0 Authorization Framework", IETF, Standard. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6749 (visited on 05/05/2022).

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

[94] "Lightweight Machine to Machine Technical Specification:Core", Open Mobile Alliance, Standard. [Online]. Available: https://www.openmobilealliance.org/release/LightweightM2M/V1_2-20201110-A/OMA-TS-LightweightM2M_Core-V1_2-20201110-A.pdf (visited on 05/05/2022).

[95] "Functional Architecture", oneM2M, Standard. [Online]. Available: https://www.onem2m.org/technical/published-specifications/release-1 (visited on 05/05/2022).

[96] O. Lassila and R. R. Robin Swick, "Resource Description Framework (RDF) Model and Syntax", W3C, Standard. [Online]. Available: https://www.w3.org/TR/WD-rdf-syntax-971002/ (visited on 05/05/2022).

[97] L. Daniele, R. Garcia-Castro, M. Lefrançois, and M. Poveda-Villalon, "SAREF: the Smart Applications REFerence ontology", ETSI, Standard. [Online]. Available: https://saref.etsi.org/core/v3.1.1/ (visited on 05/05/2022).

[98] A. Haller, K. Janowicz, S. Cox, D. Le Phuoc, K. Taylor, and M. Lefrançois, "Semantic Sensor Network Ontology", W3C, Standard. [Online]. Available: https://www.w3.org/TR/vocab-ssn/ (visited on 05/05/2022).

[99] "ECMA-404: The JSON Data Interchange Syntax", Ecma international, Standard. [Online]. Available: https://www.ecma-international.org/wp-content/uploads/ECMA-404_2nd_edition_december_2017.pdf (visited on 05/05/2022).

[100] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", W3C, Standard. [Online]. Available: https://www.w3.org/TR/xml (visited on 05/05/2022).

[101] C. Bormann and P. Hoffman, "RFC 8949 Concise Binary Object Representation (CBOR)", IETF, Standard. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8949.html (visited on 05/05/2022).

[102] A. Enterprises, "GZIP file format specification version 4.3", Network Working Group, Standard. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc1952 (visited on 05/05/2022).

[103] J. Alakuijala and Z. Szabadka, "Brotli Compressed Data Format", IETF, Standard. [Online]. Available: https://www.ietf.org/rfc/rfc7932.txt (visited on 05/05/2022).

[104] A. Donoho, B. Roe, M. Bodlaender, *et al.*, "UPnP Device Architecture 2.0", Open Connectivity Foundation, Standard. [Online]. Available: https://openconnectivity.org/upnp-specs/UPnP-arch-DeviceArchitecture-v2.0-20200417.pdf (visited on 05/05/2022).

[105] Y. Y. Goland, T. Cai, P. Leach, Y. Gu, and S. Albright, "Simple Service Discovery Protocol/1.0 Operating without an Arbiter", IETF, Standard. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-cai-ssdp-v1-03 (visited on 05/05/2022).

[106] S. Cheshire and M. Krochmal, "DNS-Based Service Discovery", IETF, Standard. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6763 (visited on 05/05/2022).

[107] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)", IETF, Standard. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7519 (visited on 05/05/2022).

[108] *OpenVPN – A Secure tunneling daemon*, https://github.com/OpenVPN/openvpn.

[109] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", IETF, Standard. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7296 (visited on 05/05/2022).

[110] J. A. Donenfeld, "WireGuard: Next Generation Kernel Network Tunnel", Tech. Rep. [Online]. Available: https://www.wireguard.com/papers/wireguard.pdf (visited on 05/05/2022).

[111] D. J. Bernstein, "ChaCha, a variant of Salsa20", Tech. Rep. [Online]. Available: https://cr.yp.to/chacha/chacha-20080128.pdf (visited on 05/05/2022).

[112] *Tailscale*. [Online]. Available: https://tailscale.com/ (visited on 05/05/2022).

[113] *Tailscale clients*, https://github.com/tailscale/tailscale.

[114] "OSGi Core", OSGi Alliance, Standard. [Online]. Available: http://docs.osgi.org/download/r8/osgi.core-8.0.0.pdf (visited on 05/05/2022).

[115] *OpenEMS - Open Source Energy Management System*, https://github.com/OpenEMS/openems. DOI: 10.5281/zenodo.6510190.

[116] A. Rossberg, "WebAssembly Core Specification", W3C, Standard. [Online]. Available: https://www.w3.org/TR/wasm-core-2/ (visited on 05/05/2022).

[117] *WebAssembly System Interface*, https://github.com/WebAssembly/WASI.

[118] *lowjs*, https://github.com/neonious/lowjs.

[119] *Espruino*, https://github.com/espruino/Espruino.

[120] *WoT Hive*, https://github.com/oeg-upm/wot-hive.

[121] *vm2*, https://github.com/patriksimek/vm2.

[122] 802.1X-2020, "IEEE Standard for Local and Metropolitan Area Networks–Port-Based Network Access Control", Standard, 2020, pp. 1–289. DOI: 10.1109/IEEESTD.2020.9018454.