Institute of Software Engineering
Software Quality and Architecture

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

A Digital Twin for Bidirectional Charging to Reduce Electricity Cost

Raphael Weber

Course of Study:	Software Engineering
Examiner:	Prof. Dr. Steffen Becker
Supervisor:	Patrick Spaney, M.Sc., Floriment Klinaku, M.Sc.
Commenced:	April 2, 2024
Completed:	October 2, 2024

Abstract

The ongoing interest in renewable energy drives efforts to continuously improve these systems. Photovoltaic (PV) battery systems could be optimized by integrating a Digital Twin (DT) in combination with bidirectional charging. The problem with existing PV systems lies in the constantly changing weather conditions and fluctuating electricity prices, which are not effectively utilized, leading to inefficient and costly solutions. We developed an optimization algorithm based on a DT that reduces electricity costs through intelligent charging and discharging of the PV battery system for apartments and residential homes, focusing on balcony power plants. To achieve this, a DT is created that digitally mirrors the physical PV battery system and continuously adapts to the current conditions using consumption, production, and price data in real-time and for forecasting. In comparison with two other self-developed algorithms, one computationally intensive and one very simple, our intelligent algorithm presents a middle ground between fast computation and cost efficiency. The results show a cost reduction of up to 300 euros or 35% for a typical German household.

Kurzfassung

Das anhaltende Interesse an erneuerbaren Energien treibt die Bemühungen um eine kontinuierliche Verbesserung dieser Systeme voran. PV-Batteriesysteme könnten durch die Integration eines DT in Kombination mit bidirektionalem Laden optimiert werden. Das Problem bei bestehenden PV-Systemen liegt in den sich ständig ändernden Wetterbedingungen und schwankenden Strompreisen, die nicht effektiv genutzt werden, was zu ineffizienten und kostspieligen Lösungen führt. Wir haben einen auf einem DT basierenden Optimierungsalgorithmus entwickelt, der die Stromkosten durch intelligentes Laden und Entladen des PV-Batteriesystems für Wohnungen und Wohnhäuser reduziert, wobei der Schwerpunkt auf Balkonkraftwerken liegt. Dazu wird ein DT geschaffen, das das physische PV-Batteriesystem digital abbildet und sich anhand von Verbrauchs-, Produktions- und Preisdaten in Echtzeit und zur Prognose kontinuierlich an die aktuellen Gegebenheiten anpasst. Im Vergleich zu zwei anderen selbstentwickelten Algorithmen, einem rechenintensiven und einem sehr einfachen, stellt unser intelligenter Algorithmus einen Mittelweg zwischen schneller Berechnung und Kosteneffizienz dar. Die Ergebnisse zeigen eine Kostenreduktion von bis zu 300 Euro oder 35% für einen typischen deutschen Haushalt.

Contents

1 Introduction **1**

2 Foundations and Related Work **5**
2.1 Foundations 5
2.2 Related Work 9

3 Collaborations **13**

4 Architecture and Implementation **15**
4.1 Digital Twin Architecture 15
4.2 Algorithm 21

5 Evaluation **31**
5.1 Algorithm evaluation 31
5.2 Importance of the Adaptations 32
5.3 Discussion 33

6 Conclusion **35**
6.1 Summary 35
6.2 Benefits 36
6.3 Limitations 36
6.4 Lessons Learned 36
6.5 Future Work 37

Bibliography **41**

List of Figures

2.1 Concepts of a Digital Model, Digital Shadow and Digital Twin 5

4.1 Architecture of the used Digital Twin based on Digital Shadows 16
4.2 Overview over the architectural cycle 18
4.3 Standardized load profile: Summer 20
4.4 Concept of the Algorithm logic 22
4.5 Concept of handling incoming real-time data 28

5.1 Price comparison of the three algorithms 32

List of Algorithms

4.1 Simulation of battery activity over 24 hours 23
4.2 Identification of optimal time periods 24
4.3 Evaluation of additional charge windows for upcoming expensive times 26
4.4 Determine optimal charging amount 27
4.5 Evaluation if the next cheapest time can be reached 27

Acronyms

API Application Programming Interface. 7

AWS Amazon Web Services. 2

DM Digital Model. 5

DS Digital Shadow. 5

DT Digital Twin. iii

EV electric vehicle. 7

IAM Identity and Access Management. 9

IoT Internet of Things. 8

MAPE-K Monitor-Analyze-Plan-Execute over a shared Knowledge. 6

MOE MOdular Energy. 13

MQTT Message Queuing Telemetry Transport. 8

PV photovoltaic. iii

RDS Relational Database Service. 8

S3 Simple Storage Service. 7

SoC state of charge. 15

SQL Structured Query Language. 8

1 Introduction

The growing awareness of climate change and the urgent need to reduce carbon emissions have led to an increasing demand for sustainable and climate-neutral energy solutions. As governments and industries worldwide strive to meet ambitious climate goals, renewable energy sources such as solar, wind, and hydropower are becoming more prevalent. However, due to their intermittent nature, such as solar energy not being available at night, wind being unpredictable, and water flow being seasonal, renewable energy must be efficiently stored to ensure a stable and reliable power supply. This shift towards greener energy production is driven not only by environmental concerns but also by economic benefits, including lower operating costs and independence from fluctuating fossil fuel prices [CG09; SS04].

In recent years, PV systems have gained significant popularity, especially in residential areas. In densely populated urban settings where space is limited balcony power plants have become a preferred option. These small-scale PV systems are designed to fit on balconies, making them accessible for individuals who do not have access to rooftops and, therefore, cannot install traditional PV systems. In Germany alone over 1.2 million balcony power plants were installed by 2023, highlighting their growing acceptance [WS15].

The primary goal of these systems is to cover current household consumption with the solar energy that is being generated at that moment. However, a major limitation of this approach is the handling of surplus energy which is produced when generation exceeds demand. A logical step to address this issue is to utilize this excess energy by incorporating battery storage into the system [CG09; WS15].

Grid-connected PV systems are a key part of this setup. A grid-connected PV system utilizes solar panels as the main source of power generation while being integrated into the public electricity grid. This configuration allows the system to work alongside the grid, providing solar power to the household when available and drawing electricity from the grid when solar generation is insufficient, thereby ensuring a balanced and flexible energy supply [UL06].

Building on this approach, additional factors such as household electricity consumption, the actual energy production of the PV system, and electricity prices, particularly when dynamic tariffs are being used, can be integrated into the energy management strategy [BMD13]. Considering these factors is crucial when aiming to reduce electricity costs as they directly influence decisions about when and how the battery should be charged or discharged [UL06]. Taking this concept further bidirectional charging can be introduced. Bidirectional charging not only allows the stored energy to be used when needed but also enables surplus energy stored in the battery to be fed back into the grid and sold at current market prices. This approach not only maximizes the use of stored energy but also provides an opportunity to generate additional income from excess electricity, adding a new dimension to energy management [JDNB18; PMG+13].

However, the complexity of managing bidirectional charging systems extends beyond simple charge and discharge cycles. Factors such as time-varying electricity prices, weather forecasts, household consumption patterns, and battery health must all be considered [UL06]. To address this complexity, DTs have emerged as powerful tools. A DT is a virtual representation of a physical system that uses real-time data to simulate and predict the behavior of its physical counterpart [NFM17]. In the context of bidirectional charging, a DT can continuously monitor and adapt the charging strategy based on real-world data, making it an ideal solution for optimizing energy usage in residential PV systems with battery storage [SFM20].

This thesis aims to develop a DT for a bidirectional charging system that reduces electricity costs by optimizing the charge and discharge cycles of a balcony PV-battery system and therefore answer the following research questions.

RQ1 How efficient is the algorithm in terms of computational speed and effectiveness in optimizing energy usage?

RQ2 How much money can an average German household save by using this DT-based approach?

RQ3 How effective is the self-adaptation of the DT in optimizing energy management compared to a non-adaptive approach?

A few prerequisites were defined for this thesis. The university specified the use of a DT platform to create the DT in order to gain knowledge about the creation process. As a result, Amazon Web Services (AWS) was chosen as it was already in use by nineti GmbH.

The contributions of this thesis are:

- The design of a DT architecture that mirrors the bidirectional charging system using AWS services, allowing real-time simulation and optimization.
- The development of an optimization algorithm acting within the DT, that dynamically adjusts charging and discharging cycles based on various data inputs including electricity prices and consumption patterns.
- A comparative evaluation of the developed algorithm against simpler and more complex approaches, demonstrating its balance between computational efficiency and cost savings.
- Insights into the adaptability of the DT, showing its ability to handle unexpected changes in energy production and consumption, which enhances its practical application in energy management.

Thesis Structure

Chapter 2 – Foundations and Related Work: Covers the foundations of DTs and bidirectional charging, providing a background on the related work in these fields.

Chapter 3 – Collaborations: Describes the collaborations involved in developing the DT and the practical applications of the system.

Chapter 4 – Architecture and Implementation: Explores the architecture of the DT and the specifics of the developed algorithm.

Chapter 5 – Evaluation: Presents the evaluation of the algorithm, including its performance compared to other approaches and the importance of adaptability.

Chapter 6 – Conclusion: Concludes the thesis with a summary of the findings, discussions on the benefits and limitations, and suggestions for future work.

2 Foundations and Related Work

In this section, the foundations as well as work related to this thesis are presented. A distinction is made between foundations and related work with the former including the concept of DTs and bidirectional charging, as well as an introduction to the used system AWS. These concepts are later used or built upon, which is why they have to be explained properly.

2.1 Foundations

2.1.1 Digital Twin

A DT is a virtual copy of a physical system that reproduces the behavior of its counterpart using real-time data [NFM17]. Due to the diverse application areas of the DT concept particularly in industrial, manufacturing, and production environments, there is no single, standardized definition as each context has led to the development of various interpretations [DJR+22; NFM17]. In their paper, Kritzinger et al. [KKT+18] provide a widely used definition that classifies the system of a DT into three subcategories. These are the Digital Model (DM), the Digital Shadow (DS), and the DT.

A DM represents a physical object, but there is no automated data exchange between the DM and the physical object it represents. The digital representation typically includes a detailed description of the physical object, for example through simulation models for planned factories or mathematical models for new products. Any data exchange is manual and changes to the physical object have no direct impact on the DM and vice versa. This can be seen in Figure 2.1 in the very left illustration.

The middle concept of Figure 2.1 shows that a DS is defined when there is an automated one-way data flow between the state of a physical object and a digital object. This means that changes to the physical object lead to changes in the digital object, but not vice versa.

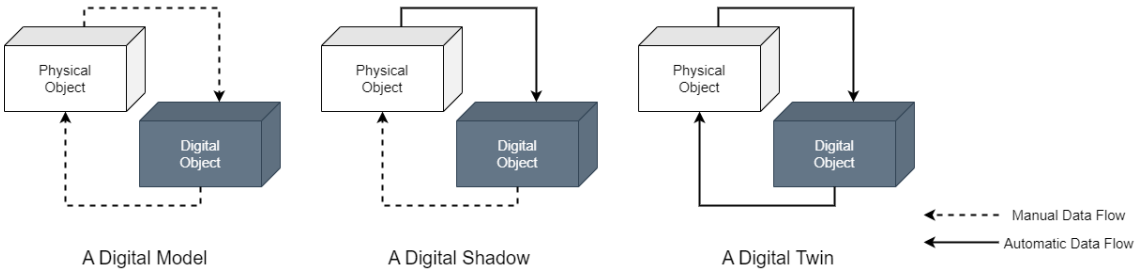

Figure 2.1: Concepts of a Digital Model, Digital Shadow and Digital Twin
Recreation based on [KKT+18]

Lastly, a DT describes the complete integration of the data flow in both directions between a physical and a digital object. In this constellation, the digital object can also have a controlling effect on the physical object. As depicted in the right design in Figure 2.1, changes in the state of the physical object directly cause changes in the digital object and vice versa.

DTs have their origins in the aerospace industry [KKT+18] but have found their way to Industry 4.0 over time [NFM17]. Given the large number of application fields [QTH+21], DTs have a correspondingly large variety of functionalities. Among other things, these include monitoring, simulation, prediction, and verification [LPT+21; QTH+21].

One of the most advanced functionalities of DTs is their ability to be self-adaptive. Self-adaptive systems can autonomously adjust their behavior in response to changes in their environment, objectives, or internal state, without human intervention. This capability is described by Bibow et al. [BDH+20] in their paper, where a DT in combination with its physical counterpart continuously processes data, collects, analyzes, and optimizes machine parameters autonomously. The system utilizes a multi-layered architecture to manage data flows and decisions, allowing it to adapt in real-time to any changes in the process environment.

For instance, the DT automatically generates and executes experiments to determine optimal machine settings evaluates the results, and adjusts operational parameters without operator input. This not only enhances the efficiency and quality of production but also reduces manual intervention, showcasing the full potential of self-adaptive systems in modern manufacturing environments [BDH+20].

Bibow et al. [BDH+20] base their model on the Monitor-Analyze-Plan-Execute over a shared Knowledge (MAPE-K) concept. This concept describes a feedback loop, which is a prominent model in autonomic and self-adapting systems. It is used for modeling, validating, and verifying these systems. The term MAPE-K is made up of several important terms that are closely linked. The *M* stands for *Monitor*, which monitors the system and through that collects input data via various sources such as sensors. The *A* stands for *Analyze*, which refers to the analysis of the data to check whether an adaption is needed. If that is the case, the *P*, which stands for *Plan* conceptualizes a way to fulfill these adaptions to achieve the system's goal. The last term of the main four is the *E* and stands for *Execute*. This concept takes the created plan and executes it through actuators of the system. Finally, the *K* stands for *Knowledge*, which represents the central knowledge of the system. The other four processes all have access to the knowledge and communicate through that. Through these concepts, the system adjusts itself according to predefined goals [ARS15].

Due to the growing number of application fields and the resulting increase in demand for DTs [NFM17], concepts such as DT platforms are also becoming increasingly popular. This is because the creation of DTs is cost and time-intensive while DT platforms such as Amazon AWS [Ser23] or Microsoft Azure [Cor24] help to facilitate this. These platforms serve as comprehensive tools for the development and operation of DTs thus resulting in a way to develop DTs efficiently [LPT+21].

2.1.2 Bidirectional charging

The concept of bidirectional charging is already well established in the electric vehicle (EV) industry, where many vehicles can store energy in their batteries and discharge it as needed. For instance, Bertoluzzo et al. [BGK21] illustrate how this bidirectional flow of electricity between the car and the grid can be practically implemented enhancing the overall management of power systems in terms of efficiency, economy, sustainability, and reliability [CM13]. The optimization of buying and selling electricity to the grid through dynamic pricing schemes is explored by Esfahani et al. [ELS22], demonstrating how strategic charging and discharging of EVs can minimize system costs.

Bidirectional energy use also plays a vital role in improving grid stability and resilience, particularly by addressing challenges like power outages and grid overloads. Discharging stored energy into the home's own grid provides a backup power source during outages, which is especially valuable in areas with unstable grid connections or no reliable access to the central grid. Furthermore, feeding energy back into the grid during peak demand can reduce strain on the power network, lowering the risk of overloading and enhancing grid reliability, thus contributing to both individual and broader energy security [PMG+13; SS04].

Beyond the EV sector, bidirectional charging is also applied in renewable energy systems. Jadhav et al. [JDNB18] discuss how PV systems combined with batteries can store solar energy and discharge it when needed to tackle issues such as supply shortages and grid instability. This approach is increasingly important as energy demand grows, fossil fuel prices fluctuate, and the intermittent nature of renewables creates challenges in maintaining a stable energy supply. By allowing stored solar energy to be used during high-demand periods or when the grid is under pressure, bidirectional charging supports a more resilient and reliable energy system reducing dependency on conventional power sources and enhancing overall stability [JDNB18; PMG+13; SS04].

2.1.3 AWS Services

Cloud services are essential in the development and operation of DTs, offering scalable and versatile platforms that support data integration, real-time analytics, and virtual representations of physical systems [LPT+21]. In this context, AWS was selected due to its broad array of services that support diverse industrial and technological solutions. This choice was influenced by the fact that many existing processes and workflows within the company were already utilizing AWS services at the start of this work. Below is an explanation of the key AWS services used in the architecture.

AWS Lambda is a serverless computing service that automatically runs code in response to specific events, such as data uploads, Application Programming Interface (API) requests, or scheduled tasks, without requiring server management. It supports various programming languages and integrates with other AWS services, making it suitable for building microservices and processing data in real-time. Lambda's flexibility and scalability enable cost-effective solutions as it only runs code when triggered, reducing costs and the need for dedicated servers [Clo11; Lam22; MV14; SK17].

Amazon Simple Storage Service (S3) is a highly scalable, durable, and secure object storage service used for data backup, archiving, content distribution, and data lake formation. It supports lifecycle management, versioning, and various encryption options, making it suitable for handling large volumes of data efficiently. S3 is often integrated with other AWS services for data transfer, content

delivery, and big data analytics [Clo11; MV14; Reh23]. S3's global availability and support for multiple access protocols make it ideal for scalable storage needs in cloud-native applications [Reh23].

Amazon DynamoDB is a fully managed NoStructured Query Language (SQL) database service that offers fast and predictable performance with seamless scalability. NoSQL databases are non-relational and support flexible data models, which allow for dynamic, schema-less data structures. This makes them ideal for applications that need to handle rapidly changing data without the constraints of a fixed schema. DynamoDB is designed for applications that require consistent, single-digit millisecond response times at any scale. Its flexible schema design allows it to manage diverse and rapidly changing datasets, making it ideal for real-time data processing and Internet of Things (IoT) applications. The service also integrates well with AWS Lambda to automate workflows based on data changes [Clo11; Lam22; MV14].

AWS IoT Core is a managed cloud service that enables secure interaction between connected devices and cloud applications using protocols like Message Queuing Telemetry Transport (MQTT), WebSocket, and HTTPS. The rule engine within IoT Core allows data to be processed in real-time using SQL-like queries to filter, transform, and route messages based on defined conditions. For example, incoming data from sensors is sent to AWS IoT Core via MQTT. The rule engine processes these messages by running SQL-like queries such as:

```
SELECT * FROM 'sensor/topic' WHERE condition
```

This query filters and transforms the data based on specific criteria and can route the processed data to other AWS services, such as DynamoDB for storage or AWS Lambda for further computation. This path allows data to flow efficiently from IoT devices to cloud applications enabling real-time processing and analysis in DT architectures [Clo11; JH18; MV14].

AWS IoT SiteWise is a managed service for collecting, organizing, and analyzing data from industrial equipment. It uses asset models to represent the physical structure of industrial systems capturing their properties, measurements, and relationships. SiteWise helps monitor performance, identify inefficiencies, and optimize operations through structured data analysis both in the cloud and at the edge, enhancing visibility and decision-making in industrial environments [Clo11; MV14; Reh23].

AWS IoT TwinMaker helps create DTs by integrating data from multiple sources such as sensors, video feeds, and enterprise systems. It allows the development of detailed 3D models and simulations that visualize physical systems, providing insights into operational performance and enabling predictive maintenance and optimization of complex environments [Clo11; MV14; Reh23].

Amazon Relational Database Service (RDS) is a managed service that simplifies setting up, operating, and scaling relational databases in the cloud. It supports multiple database engines, including MySQL, PostgreSQL, MariaDB, Oracle, and SQL Server. RDS automates tasks such as backups, patching, and scaling, which reduces the administrative burden. RDS provides high availability and fault tolerance through Multi-AZ (Availability Zone) deployments, where data is automatically replicated to a standby instance in a different Availability Zone. This setup ensures minimal downtime during maintenance or failover scenarios, enhancing data reliability and system resilience. RDS is suitable for applications that require robust and reliable data management, making it a strong choice for supporting DTs with transactional and historical data storage needs [Clo11; MV14].

AWS Identity and Access Management (IAM) provides granular control over AWS resources allowing the definition of detailed permissions for users, groups, and services. IAM is critical for maintaining security and compliance ensuring that only authorized entities have access to sensitive data and operations within AWS environments. Its integration with other AWS services helps enforce security best practices across cloud applications [Clo11; Lam22; MV14].

These AWS services provide a comprehensive framework that supports the creation, operation, and optimization of DTs. This framework was essential for developing and executing the DT as well as the algorithm presented in this work.

2.2 Related Work

There is limited literature on this specific topic, as many studies on PV systems often exclude battery integration and research on battery systems tends to focus on bidirectional charging within the automotive industry. Existing studies that address this field generally overlook the role of DTs and, consequently, the aspect of self-adaptivity, highlighting a gap in current research. This review will outline key areas explored so far emphasizing where existing approaches fall short and identifying the potential contributions of integrating DTs in this context.

In their research on user-centric residential PV battery systems, Basmadjian [Bas20] focus on algorithms to optimize self-consumption and cost efficiency. This work explores how PV battery systems can be used to maximize self-consumption rates and reduce dependence on the grid. Through the use of real-time pricing and other dynamic pricing strategies, two algorithms have been developed that schedule battery charging during low-cost periods and discharge energy during expensive peak consumption periods. These approaches help to minimize the total cost of ownership while improving the energy autonomy of households. Using heuristic methods and linear programming models, the study developed these algorithms to identify optimal charging and discharging times that maximize the economic benefits of residential PV battery systems. It was found that significant savings in electricity costs can be achieved in various situations increasing the financial attractiveness of such systems for end users. However, this article is limited to the hours between 5 p.m. and 7 a.m., since no electricity is generated by the PV system there. That is something, which also has to be taken into account in this work's approach and the algorithm is also not used for bidirectional charging, which should sell other electricity at high price times.

A similar approach is taken in the research by Boström [Bos16], where a cost reduction is also achieved through selective charging and discharging. For this purpose, the author used weather forecasts and a PV system to simulate the predicted PV production and used this as well as the consumption and price data to create a battery optimization algorithm. Excel and linear programming were used to find global extremes in order to charge or discharge at these extremes. Although this approach is very similar in principle to the approach in this paper, there are a few differences. Among other things, the price data is only queried for two hours and must therefore be interpolated to obtain the missing values. The battery capacities considered are in the range between 5 and 20 kWh, which is significantly higher than the values considered in this paper. Similarly, the household consumption of 12,580 kWh per year and the PV sizes of 3 - 20 kW are not in a range that is relevant for this work. This approach also does not include bidirectional charging and therefore does not sell the surplus electricity at peak times, but at the respective time when charging

is taking place and nothing more fits into the battery. Neither this nor the previous approach uses a DT to keep the algorithm running continuously and self-adapting to unexpected events allowing the system to adjust autonomously in real-time.

Another viewpoint explored in recent studies is the use of Balcony-PV systems combined with battery storage to enhance energy efficiency in residential settings. Both Eum and Kim [EK20] and Kim et al. [KEK20] investigate the effectiveness of integrating balcony PV systems with batteries in apartment complexes. In these studies, PV panels are installed on multiple apartments, and excess solar energy that is not immediately needed is stored in a battery for later use. When the sun goes down, this stored energy is used to cover a portion of the household's consumption.

Both articles demonstrate the advantages of using battery storage as they show that a significant portion of energy demand can be met by the stored solar power, thereby reducing the overall dependence on the grid. However, what sets these studies apart from the approach in this work is the absence of DTs and the lack of integration of dynamic electricity tariffs. Neither study considers the possibility of purchasing additional electricity at optimal times nor do they utilize adaptive algorithms to manage the charging and discharging cycles in real-time, which are crucial for minimizing costs. Consequently, their focus is not on achieving the most cost-effective operation, highlighting a key area where the application of DTs and dynamic pricing strategies could provide substantial improvements.

A paper with a similar approach is the one conducted by Saad et al. [SFM20], which investigates the use of DT technology in managing energy systems that include energy sources, storage units, and consumer loads within a power distribution network. The authors propose a framework that combines cloud computing and IoT technologies to create a digital replica of both the physical infrastructure and control systems. This DT serves as a real-time monitoring and control tool enabling continuous adjustments to optimize energy distribution and system performance. The study presents a DT model that integrates real-time data from sensors and controllers ensuring that the digital replica accurately reflects the current state of the physical system. This allows for automated, real-time decision-making processes that can improve the efficiency and reliability of the energy system. Saad et al. [SFM20] also explore various applications of the DT including its potential for system monitoring, fault detection, and predictive maintenance.

While the paper offers valuable insights into the deployment of DTs for real-time energy management, it does not delve into the integration of dynamic pricing models or bidirectional charging strategies. These elements, which are central to the approach outlined in this thesis, are not addressed, leaving room for further exploration in these areas.

2.2.1 Literature Research Methodology

For the collection of relevant literature, Google Scholar was utilized. The search strategies involved entering various keywords that were pertinent to the research topic. Frequently used keywords included terms such as “AWS,” “Digital Twin,” “Bidirectional Charging,” “Bidirectional Loading,” “PV-Battery,” as well as “AWS Services.” These terms among others were used both individually and in combination in the search engines.

Given that the literature on this topic is somewhat limited, especially concerning the specific combination of technologies and concepts under investigation, the selection process was particularly careful. The search of the term combination “aws digital twin bidirectional charging” results in under 900 results with most of them not being cited even once in other papers. Another interesting outcome of this search is, that every checked paper on this topic was published in 2020 and later. There are even a lot of publications from 2024 that suggest recent interest in this topic. When certain searches, such as “IoT Core” did not yield the desired results, the advanced search feature of Google Scholar was employed. This allowed for more targeted searches by looking for specific terms within titles or other specific areas to improve the relevance of the results.

The selection of literature was based on quality criteria focusing on relevance, recency, and the scientific reputation of the sources. This means, that if there were several papers containing the same information, the papers which already had more citations were chosen. With the exception of a few PV papers most of the other publications are from 2018 and later. We did not have any strict policy about discarding papers that were too outdated but still tried to use newer papers while still utilizing those that were already in regular use. Papers were included in the collection if they had a direct connection to the central topics of the research and were methodologically sound.

Furthermore, additional relevant literature was identified through forward and backward searches of the previously identified articles. This approach allowed for a deeper exploration of the research field and ensured that no significant studies were overlooked.

3 Collaborations

This bachelor thesis deals with the development of an optimization algorithm for the charging and discharging cycles of the MOdular Energy (MOE) system of the nineti GmbH, a start-up company based in Stuttgart, Germany. The nineti GmbH creates, produces, and distributes innovative solutions for energy generation and storage, focusing on both physical and digital products. Their flagship product, MOE, represents a sustainable energy technology solution with its modular balcony power plant and integrated storage system.

A balcony power plant offers a practical solution for green energy production, especially in urban environments. It is designed to be user-friendly, allowing easy installation and operation directly from the balconies of residential buildings. This adaptability makes it highly suitable for urban dwellers who may not have access to traditional rooftop installations. When energy production exceeds the own household consumption the excess power is given back to the grid without any compensation. [BP17]

That is where the inclusion of a storage system comes into play. A balcony power plant including a battery can store the overproduced power for later use, ensuring that users can maximize their energy savings and reduce their dependence on the grid. This is what nineti's MOE system constitutes, a balcony power plant with a storage system.

At the heart of the MOE system is a 1.5 kWh battery for energy storage coupled with a bi-directional inverter which has an output of 1.5 kW and a solar panel capable of producing up to 800 watts of power. The bi-directional inverter is crucial as it converts direct current into alternating current power and vice versa. This flexibility is key to maximizing the use of the energy generated and adapting to changing household energy needs.

MOE has a modular design that allows multiple batteries to be stacked, significantly increasing storage capacity according to the user's needs. The scalability of the system is beneficial for adapting to different energy needs and conditions. The efficiency of the solar panel is affected by its placement, orientation, and potential shading, but additional panels can be added if space permits increasing the overall energy production.

The core of this thesis is the creation of an algorithm that utilizes data from a DT of the MOE system. This DT serves as an advanced simulation tool that replicates the physical and operational characteristics of the MOE enabling accurate scenario analysis and performance evaluation under different environmental conditions. The algorithm is designed to adjust the battery's charging and discharging strategies based on a number of factors including expected energy demand, weather forecasts, and current electricity prices.

This approach is a way to effectively integrate renewable energy sources, which are often variable in nature. By optimizing the storage and use of energy the algorithm not only improves the operational efficiency of the MOE system but also extends the life of the batteries. This improves both the economic and environmental performance of the system, positioning MOE as an attractive option for sustainable home energy solutions.

4 Architecture and Implementation

The following chapter explores the development and implementation of the DT system. It begins by providing an architectural overview of the created procedure. The second section focuses on the development of the algorithm explaining key design decisions and used features.

4.1 Digital Twin Architecture

The architectural structure of the used DT is based on the one from Bibow et al. [BDH+20] and has been extended with additional layers including the IoT Gateway Layer, Data Handling Layer, and Visualization Layer. These layers were developed to meet specific requirements identified in AWS architecture, and their names were chosen accordingly to reflect their functions effectively. The Figure 4.1 depicts the used architecture based on AWS and its different layers: cyber-physical layer, IoT Gateway layer, data handling layer, connection layer, application layer, and visualization layer.

Cyber-Physical Layer. The cyber-physical layer provides the DT with data. From the real-time data of the production and consumption to the state of charge (SoC) of the battery which is provided by the MOE system. On the other hand, multiple forecasts of price and production data are being provided by different APIs. For the price data, there are day-ahead forecasts, providing data for the next day. Among the available APIs that provide similar data, such as Nord Pool [Poo20] and SMARD [Bun24], the Energy-Charts API [Cha24] was utilized in this context. The consumption data is retrieved from the Stadtwerke Kiel website [Gmb24], where the H0 profile specifically represents household consumption. As these are standardized data they are consistently being provided by multiple sources. The Stadtwerke Kiel was selected due to its well-structured and easily accessible data. There are nine different data columns where it is being differentiated between three seasons (summer, winter, and transition) as well as three different days (Saturday, Sunday, and weekdays in general). Each data column provides quarter-hourly power values for an annual consumption of 1,000 kWh. Finally, the production data is received from the Solcast API [Jam16], which provides PV production forecasts for the next three days. These forecasts contain data in half-hourly intervals to estimate how much energy is being produced.

IoT Gateway Layer The cyber-physical system of the previous layer needs a way to upload its data. That's where this layer comes into play. Here the AWS IoT Core receives the data through a MQTT protocol. The system uses predefined MQTT topics to send the messages. The IoT Core uses a rule engine with SQL queries to process incoming messages. The query in our case is formulated in such a way that if messages are received on the correct topic, they are forwarded directly to DynamoDB.

Data handling Layer As the name already suggests, this layer receives data from the previous layers and sends them to the correct spots. This is done by AWS Lambda, a serverless computing service that operates with an event-driven execution. This means that these AWS Lambda functions are

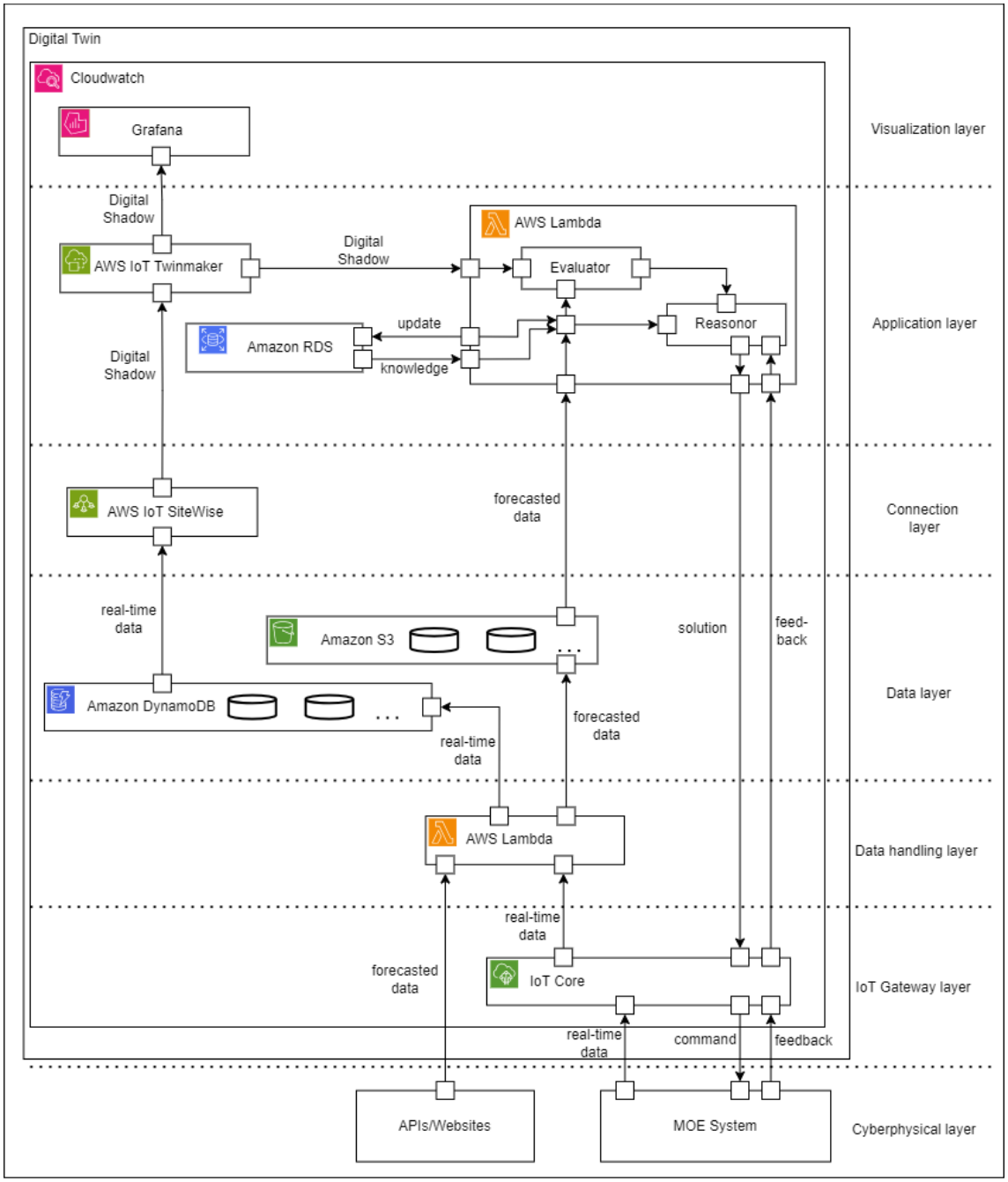

Figure 4.1: Architecture of the used Digital Twin based on Digital Shadows Extension based on [BDH+20] in consideration of the AWS services

triggered by events. Those events can come from a variety of AWS services such as the previously mentioned IoT Core. But it also triggers at predefined times. This is the case with the price and the production data, which gets updated every day. The AWS Lambda can execute the code in different supported programming languages, such as Node.js, Java, and more, or in our case, Python.

Data Layer From the AWS Lambda, the data meaning either the real-time production/consumption which is labeled as *real-time data* in Figure 4.1 or the forecasted price, production, and consumption, labeled *data*, gets sent to different databases regarding the different use cases. When it comes to real-time data the Amazon DynamoDB is preferred since it is a NoSQL database, meaning it stores data in a schemeless format and can therefore handle flexible and dynamic data structures better. Data in DynamoDB is stored in tables, which consist of items (rows) and attributes (columns). On the other hand, there is Amazon S3 which stores data as objects within buckets. The Amazon S3 is being used for the daily updated data, as well as the storage of the consumption data as it does not need to be updated in real-time.

Connection Layer AWS IoT SiteWise, which is located in this layer, uses asset models to represent the physical structure and hierarchy of the physical system. These models define the properties, measurements, and relationships between assets allowing to organize and understand the data in the context of the system's environment. Here, as part of the measurements, real-time data gets modeled. Since this is the location where the digital representation of the physical system receives automatic data flow from its physical counterpart, from here on out the data can be specified as DS. Up to this point, the automation only works in one way.

Application Layer In this layer, the main part of the DT takes place. There are different services playing a role here. AWS IoT Twinmaker can create models that represent the structure, components, and relationships of the physical system. For visualization 3D models of the system's components can be integrated to improve the already created models. With the DS it receives from the AWS IoT SiteWise service the AWS IoT Twinmaker can combine the models and the assets to connect the data with the corresponding model. Another big part of this layer is the AWS Lambda, where the logic of our algorithm is provided. This is where the intelligence of the DT lies. It consists of an evaluator, which analyzes the DS as well as the production and price data it receives from the Amazon S3 and evaluates system states. In our case, this would be the charging plan. Based on this evaluation, the evaluator creates goals, in this circumstance the charging times, that are passed to the reasoner. The reasoner uses the knowledge contained in the knowledge database (Amazon RDS) to develop solutions that achieve these goals. This means that the charging times can be achieved even if the real-time data varies from the predicted forecasts. Amazon RDS is being used since, compared to Amazon S3, it is not for long-term data storage. Former charging plans are stored here so that they can be used as a basis for future adjustments. This ensures the continuous adaptation of the system to changing conditions. The AWS Lambda then provides a solution to the system. It receives this solution through the IoT Core which commands the system to charge and discharge during those selected hours. Feedback is sent back to the reasoner in the exact same way to ensure that everything went as planned.

Visualization Layer The Grafana service gets the DS provided and visualizes the 3D models with the associated real-time data. Grafana allows the creation of custom dashboards, and as new data is ingested these dashboards automatically update, reflecting the most recent information. The other service in this layer is Cloudwatch. It is a monitoring and observability service provided by AWS that enables to collect, monitor, and manage operational data. That is why this service extends

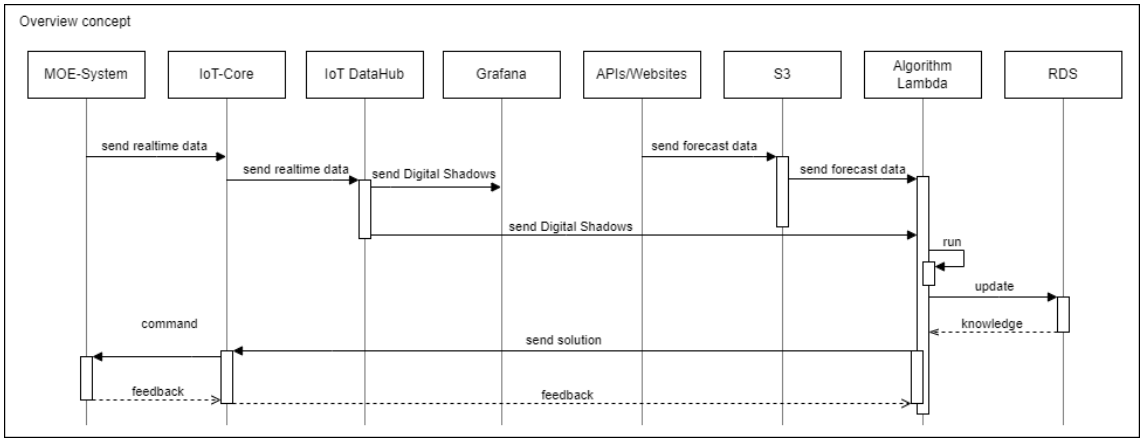

Figure 4.2: Overview over the architectural cycle

across the entire digital system. It monitors every service, provides a dashboard and graphs to visualize, and sends alarms as well as notifications if something does not go as planned or exceeds any previously set limits.

All the MAPE-K elements can also be seen in our architecture. The “Monitor” phase is reflected in the collection of real-time data from the MOE system, handled by the Cyber-Physical Layer and AWS IoT SiteWise, where this data is continuously gathered and stored in Amazon DynamoDB. The “Analyze” phase is carried out by the evaluator within AWS Lambda, which processes this data along with historical and forecasted information to assess system states. The “Plan” phase is represented by the reasoner, which uses the insights from the knowledge database stored in Amazon RDS to formulate actionable plans such as optimizing charging schedules. The “Knowledge” itself is as the action already suggests the Amazon RDS database. Finally, the “Execute” phase is managed by AWS IoT Core, which implements the reasoner’s decisions ensuring that the system adapts in real-time. Feedback from these actions is fed back into the knowledge base, enabling continuous learning and adaptation, thereby completing the MAPE-K loop.

Figure 4.2 shows a cycle of the previously discussed architecture. The lifelines in the graphic depict the temporal dependencies of the concept. Clearly visible is the data transfer from the MOE system as well as the APIs/websites through the different IoT and data management services from AWS. Some services such as DynamoDB and the IoT services SiteWise and Twinmaker were combined into the IoT DataHub for a better overview.

The process begins with the continuous transmission of real-time data from the MOE system to IoT-Core. This data is sent persistently ensuring that the system remains updated with the latest information regarding energy production, consumption, and battery SoC. This real-time data is then forwarded to the IoT DataHub, which consolidates and processes it before sending the DS to Grafana for visualization purposes enabling real-time monitoring and analysis.

In parallel, forecast data from external APIs and websites is periodically retrieved, currently set to be updated every 24 hours, as indicated by the planned schedule for API calls. This forecast data includes predictions on energy prices and production, which are essential for the algorithm’s decision-making process. The forecast data is sent to Amazon S3 for storage and then forwarded to the Lambda function.

What is shown here is that all the data, in this case, the DS and the forecast data, must first be loaded into the lambda before it can start the algorithm, which is illustrated with the run call. During this phase, the algorithm evaluates the system state and uses the gathered data to create an optimal load plan.

The continuous improvement that the DT undergoes can be seen by updating and retrieving the information from the RDS. The Lambda function updates the knowledge stored in Amazon RDS, which contains historical data and learned adjustments to improve future decisions. This continuous update process ensures that the DT improves over time, becoming more accurate and adaptive to changes in the environment.

Finally, the feedback loop sends a load plan solution to the system and receives confirmation on whether the execution was successful as planned. The system then executes the plan and sends confirmation back to the Lambda function to verify that the execution was successful. This feedback mechanism is crucial for ensuring that any deviations from the expected performance are captured and used to refine future operations.

4.1.1 System Design

There are three main components in the system used. Those would be a PV system, an inverter, and a battery. The PV system consists of two panels each of which can generate up to 400W peak. They have an inclination of 30 degrees, are south-facing, and create no shade in order to maximize the possible output. The inverter has an output of 1500W meaning it can either feed 1500W into the battery in one hour or retrieve it. MOE is working with a battery, with a capacity of 1500Wh. This means that in combination with the inverter the battery can be fully charged from 0 to 100 within one hour.

4.1.2 Prerequisites

In order to test our algorithm a few preconditions must be met. Apart from the AWS account, we need the prediction data. As already mentioned, the consumption data is obtained on the basis of standard load profiles, which come from the Stadtwerke Kiel website [Gmb24]. The website differentiates between profiles but the only one of interest to us is H0 since it reflects household consumption. Information is available in nine different variants, where a distinction is made between three different seasons (summer, winter, and transitional period) and three different days (Saturday, Sunday, and weekdays in general). The data are per entry in quarter-hour performance values for the annual consumption of 1,000 kWh/a. This means that there are 96 values per entry. Three of these can be seen in Figure 4.3, where the three different days for summer profiles are depicted. It is visible, that during weekdays the consumption in the early hours is higher whereas the overall consumption during the weekend is higher. This data is loaded into AWS and thus serves as the basis for the algorithm. The energy charts API [**enery-charts**] is used to obtain the price data. These are specified day-ahead, which means that data can only be received for tomorrow and with an hourly interval. After observing we have found out that these are updated daily around 3 pm. This means that until then only the prices for the same day are known and from then on the prices are given up to and including 11 pm the following day. This is also the reason why our price inquiry runs around 4 pm, which ensures that we always get the updated prices immediately. We also

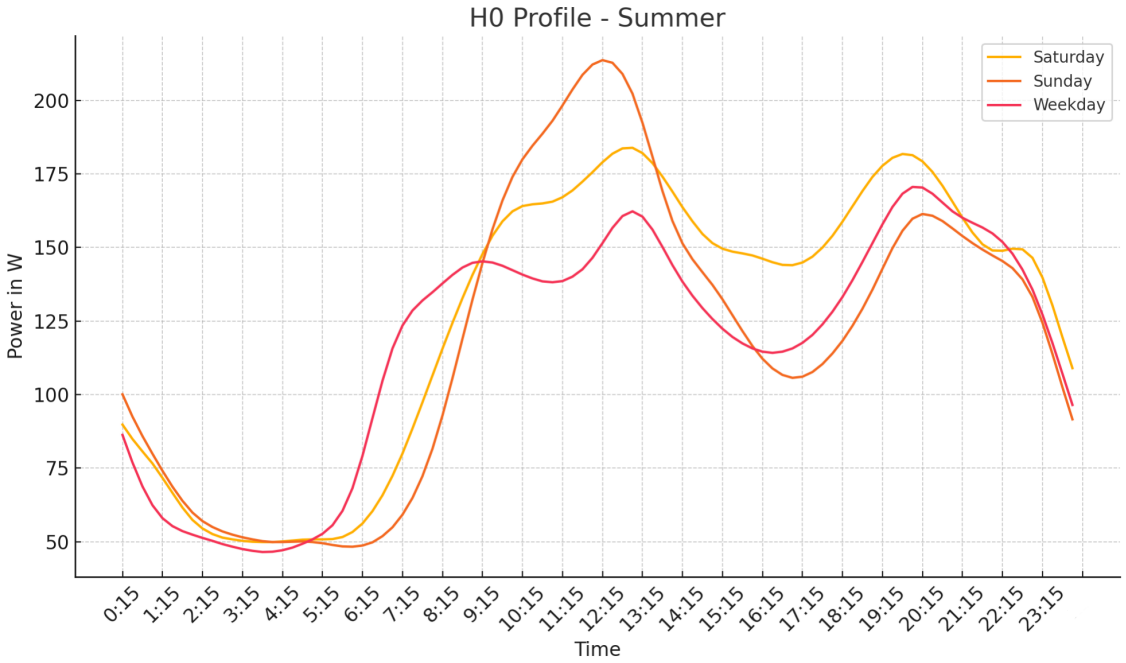

Figure 4.3: Standardized load profile: Summer
Creation based on the data of [Gmb24]

receive the production data in watts for the next three days at half-hourly intervals via an API. We can specify our PV system in the Solcast API [Jam16], as described above. The issue is that the minimum PV specification on this page is set at 1500W for a panel. Although this limitation exists, it is later managed within the environment variables. Additionally, despite the data covering a three-day span, they are updated daily to ensure more accurate and up-to-date information.

Environment Variables

In order to adapt the data and the system to the algorithm and determine predefined values, one can define environment variables for an AWS Lambda, which are entered separately. These include a divider, which is used to adjust the production data. The values arrive for 1500W but we only have 400W on the roof twice therefore we need a divider of 1.875 to get the correct values. Also required is a multiplier which adjusts each entry of the consumption data table. The whole load profile is designed for an annual consumption of 1000kW but we assume an annual consumption of 2400kWh, which is why our multiplier is 2.4. The battery capacity and the charging rate are specified by the system and are both 1500Wh. We have also incorporated a few additional measures to extend the life of the battery. First of all, there is a strict charging limit per day of 3000Wh, which means that it is only possible to discharge and recharge twice per day. But this charging limit can be split into several charges. The only important point is, that after a whole day not more than two times the battery capacity was charged. Another one is the minimal and maximal capacity limit. Both of them are at 10%. With the addition that it is no longer possible to load or unload too much and that a daily limit has been set, the life expectancy of the battery should increase.

4.2 Algorithm

In this section, the charging algorithm is explained. To achieve that the first part will show a basic concept of the functionality of the algorithm while the second part depicts several pseudo codes of the actual code for a better overview and visualization.

Figure 4.4 shows a simplified activity diagram that does not go into all the exact details of the actual code but rather provides a rough overview. It starts with the collection of data. This shows both the price prediction data plus the real-time and forecast data for consumption and production. The next two steps are important for deciding later whether to charge or discharge. By limiting the periods in which the price extremes are then searched for, all important points in time are found. Dividing the periods differently may result in some high or low points not being labeled as such and therefore being overlooked.

This brings us to the first important loop of the algorithm, which divides the total time period into individual hours. Within this outer loop, the first query is whether the current hour is an hour marked as favorable. If this is the case, a decision is made as to whether the loading capacity should be adjusted if there is an even more favorable time outside the current period. This means that the battery only needs to be charged until this point is reached in order to charge only the cheapest energy into the battery.

At this point, the decision is made. If we are in one of the favorable hours or if it has been determined in the previous query that the battery should be charged, we reach the branch where it is determined that the battery should be charged. Charging continues until the specified limit is reached. However, if we are in one of the expensive hours the battery is discharged to keep costs low. Instead of using the expensive electricity at the current time, the cheaper electricity charged in the battery is being used. If neither of the two options is the case the third scenario is considered, where a distinction is made based on the remaining battery charge. If sufficient charge is available it is used, otherwise the required energy is drawn directly from the grid. In the last case, the costs incurred are also calculated. The two loops are then only queried. If the current hour still contains further quarter hours the process continues within the same hour. If this is not the case, the outer loop increments by one moving to the next hour. This continues until the specified period is completed.

Algorithm 4.1 is the centerpiece of the algorithm. It simulates the charging and discharging behavior of a battery over a longer period of time in order to minimize operating costs, taking into account electricity prices, consumption, and energy production. Firstly, the algorithm receives various input data, including electricity price data, consumption data, production data, the current battery capacity, and the current date and time. The output of the algorithm consists of the simulation results and the total operating costs incurred. These simulation results now include detailed predictions on how much SoC is to be expected in the battery at specific times additionally to information on when the battery should be charged or discharged to optimize cost efficiency for the next 24 hours.

Two variables are initialized at the start of the simulation: an empty list called *results*, which stores the results of the simulation, and *total_cost*, a variable for calculating the total costs. In addition, the cheapest and most expensive hours for using the battery are determined using the electricity price data and the current date by calling the *FindOptimalTimes* function. The results of the function are saved in two different lists. The core of the algorithm consists of a loop that runs through the time hour by hour until the specified end time is reached. This is usually 24 hours after the start time. An exception would be if the price data or the production data do not reach this point in time. Within

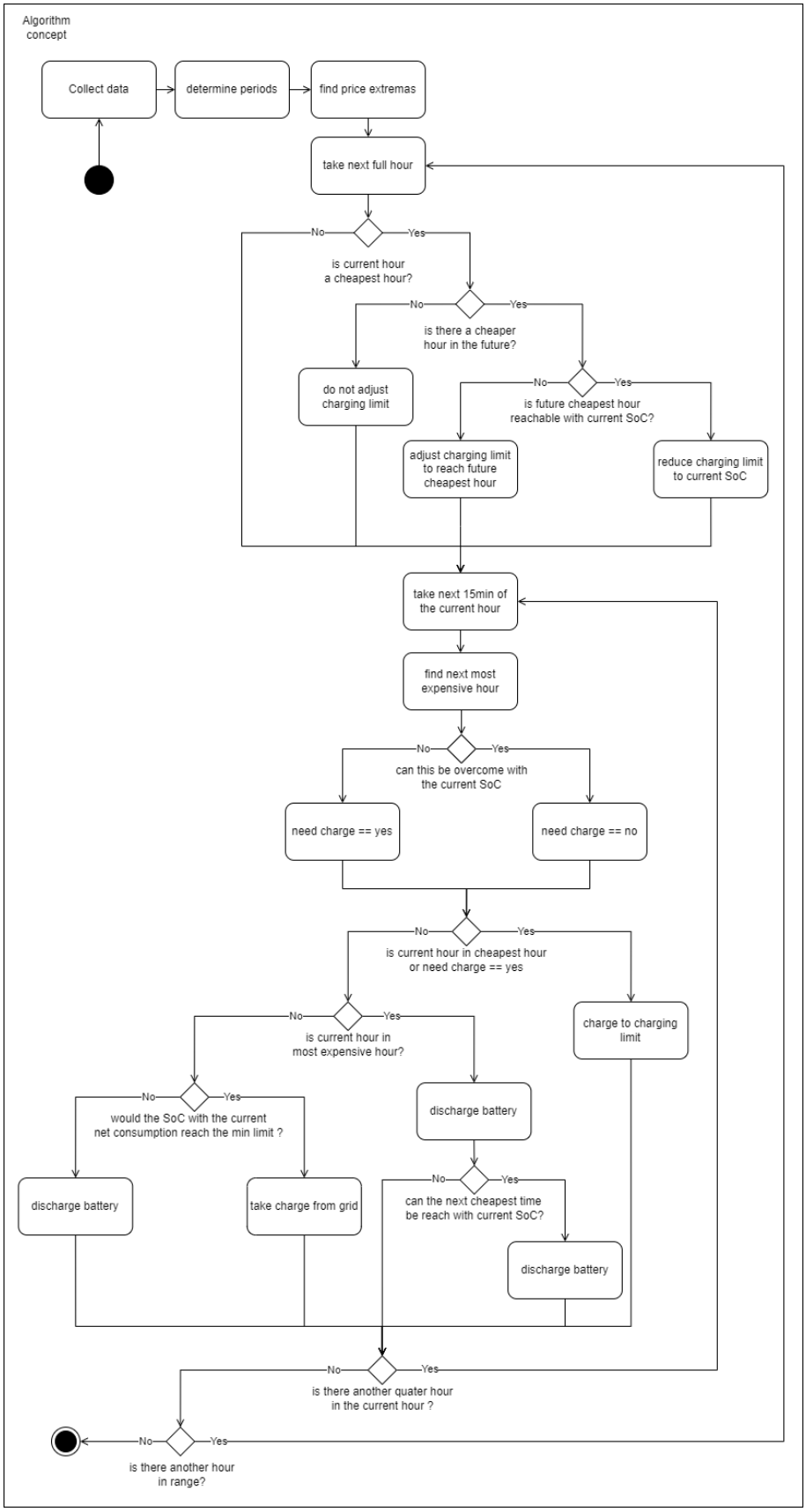

Figure 4.4: Concept of the Algorithm logic

Algorithm 4.1 Simulation of battery activity over 24 hours

```
1: procedure SimulateBatteryLife
2:     Input: price_data, consumption_data, production_data, current_battery_capacity, current_date
3:     Output: simulation results, total cost
4:     results ← [ ]
5:     total_cost ← 0
6:     cheapest_hours, most_expensive_hours ← FindOptimalTimes(price_data, current_date)
7:     while next_full_hour ≤ end_time do
8:         if next_full_hour ∈ cheapest_hours then
9:             current_limit ← DetermineOptimalChargingAmount(current_battery_capacity, price_data, consumption_data, production_data)
10:        end if
11:        for minute ∈ {0, 15, 30, 45} do
12:            current_period ← next_full_hour + timedelta(minutes=minute)
13:            net_production ← production - consumption
14:            should_charge ← EvaluateAdditionalChargeWindow(current_period, battery_status, price_data, consumption_data, production_data, most_expensive_hours)
15:            if current_period ∈ most_expensive_hours then
16:                current_battery_capacity ← Discharge(current_battery_capacity)
17:                if CanReachNextCheapest(current_period, current_battery_capacity, consumption_data, production_data, cheapest_hours) then
18:                    current_battery_capacity ← Discharge(current_battery_capacity)
19:                    current_cost ← CalculateCost                    // these costs are negative
20:                    total_cost + = current_cost
21:                end if
22:            else if current_period ∈ cheapest_hours or should_charge then
23:                charge_amount ← min(current_limit, CHARGING_RATE)
24:                current_battery_capacity ← Charge(current_battery_capacity)
25:                current_cost ← CalculateCost
26:                total_cost + = current_cost
27:            else
28:                if current_battery_capacity + net_production > MIN_CAPACITY_LIMIT then
29:                    current_battery_capacity ← Discharge(current_battery_capacity)
30:                else
31:                    current_cost ← CalculateCost
32:                    total_cost + = current_cost
33:                end if
34:            end if
35:            results.append(current_period, current_battery_capacity, net_production, current_cost)
36:        end for
37:        next_full_hour ← next_full_hour + 1 hour
38:    end while
39:    return results, total_cost
40: end procedure
```

Algorithm 4.2 Identification of optimal time periods

```
 1: procedure FindOptimalTimes
 2:     Input: price_data, start_time
 3:     Output: cheapest_times_list, most_expensive_times_list
 4:     periods ← FindPeriods                          // predefined periods from 9 to 9
 5:     cheapest_times_list ← [ ]
 6:     most_expensive_times_list ← [ ]
 7:     for each period in periods do
 8:         period_data ← price_data(period)
 9:         if period_data is not empty then
10:             cheapest_times ← min(period_data)
11:             cheapest_times_list.extend(cheapest_times)
12:             expensive_times ← max(period_data)
13:             most_expensive_times_list.extend(expensive_times)
14:         end if
15:     end for
16:     return cheapest_times_list, most_expensive_times_list
17: end procedure
```

each hour, if it is a favorable hour, the optimum charging amount is determined. A function named *DetermineOptimalChargingAmount* is called for this purpose. The result of this call specifies a possible limit to the charging amount.

The simulation is then carried out for every quarter of an hour within this hour. During each quarter of an hour, the net energy production is calculated by subtracting the consumption from the production. The algorithm then decides in line 14 whether additional energy should be charged into the battery during this time period. This added decision is intended to guarantee that the battery charge is used for the most expensive hour of the respective period. Lines 15 to 33 are used to decide what is to be done in the respective quarter of an hour. If the quarter of an hour is one of the most expensive hours, the battery is discharged to avoid costs.

A special item can be found in lines 17 to 20. This is where the bidirectionality of the system comes into play. The function *CanReachNextCheapest* is used to query whether the next most favorable time can still be reached with the current charge of the battery in order to determine whether and how much electricity can be sold. The whole thing is only done within the *most_expensive_times* to make as much profit as possible. However, if the quarter of an hour is in the most favorable hours or if an additional charge from calling up the function is considered useful, the battery is charged taking into account the maximum possible charging limit or the maximum permissible charging speed. Any costs incurred when charging are calculated and added to the total costs.

If none of the previous conditions apply, the battery is discharged if the battery capacity is above a certain minimum value. If this is not the case, the energy is taken directly from the line. In this case, the battery charge is not changed and the costs for drawing external power are calculated and also added to the total costs. After every quarter of an hour, the relevant data such as the current time, the battery capacity, the net energy production, and the costs incurred are saved in the results list. At the end of a full hour, the system moves on to the next hour and the simulation continues. Finally, the algorithm outputs the collected simulation results and the calculated total costs.

Algorithm 4.2 is used to identify the cheapest and most expensive time periods within a certain time frame based on electricity price data. This information is used to determine the optimum time for charging or discharging a battery. Firstly, the algorithm receives the electricity price data and a start time as inputs. The outputs of the algorithm are two lists: a list of the cheapest times and a list of the most expensive times. The algorithm begins by identifying predefined time periods. Price data in Germany generally follows a very similar pattern [Cha24]. They become expensive in the morning and evening when a lot of energy is needed and fall again at midday and at night when a lot of energy is produced by solar and when little energy is needed. In order to capture all possible extremes of price data, we have opted for a 12-hour period from 9 to 9. This ensures that all extreme times are captured with the highest probability. For each of these time periods, we check whether price data is available for this period. A mask filter (*period_mask*) is created for each time period found, which marks all points in time within the period under consideration. The price data that falls within this time period is stored in *period_data*. If there is relevant price data for a time period, the cheapest point in time within this time period is determined and added to the list of cheapest time periods. Similarly, the most expensive point in time within the period is identified and added to the list of the most expensive periods. After all time periods have been run through, the algorithm returns the two lists: *cheapest_times_list*, which contains the cheapest time periods, and *most_expensive_times_list*, which shows the most expensive time periods.

Algorithm 4.3 is used to determine whether an additional charge of the battery is required in a certain time window to keep the battery sufficiently charged in order to guarantee that the most expensive hours of battery charging are covered. Future extremes in electricity prices as well as current and forecast energy production and consumption are taken into account. First, the algorithm receives as input the current time, the current battery capacity, electricity price data, consumption data, production data, and a list of the times at which electricity prices reach their extreme values (*extreme_times*). The output of the algorithm is a Boolean value that indicates whether additional charging is required (True) or not (False). The algorithm begins by finding the next time in the future at which an extreme value in the electricity price will be reached (*next_extreme_time*). If no such future extreme value is found, the algorithm immediately returns False as no additional charge is required. However, if there is a future extreme value, the algorithm checks whether this is an expensive time period. For that, the function *isExpensive* is called with the *next_exteme_time* as input. If the next extreme value is a favorable period, False is also returned as it is not necessary to charge the battery prematurely. However, if the next extreme value is an expensive period, a simulation is carried out to check whether the battery capacity could fall to a critical level by this time. This simulation runs in 15-minute increments and takes into account the net energy (production minus consumption). If during the simulation it is determined that the battery capacity falls below a specified minimum value (*MIN_CAPACITY_LIMIT*), the algorithm returns True, indicating that additional charging is required. If the simulation is successfully completed without falling below the minimum battery capacity, the algorithm returns False, indicating that no additional charging is required. This algorithm helps to proactively decide whether additional charging of the battery is necessary to avoid high electricity costs.

Algorithm 4.4 calculates the optimal amount of energy that should be charged into the battery in order to minimize cost. This algorithm is only called in favorable hours and takes into account current and future electricity prices as well as energy consumption and energy production. Firstly, the algorithm receives various input data including the current time, electricity price data, consumption data, production data, the current battery status, the maximum battery capacity, and optionally the maximum number of hours for which the calculation is carried out - 12 hours by default. The output

Algorithm 4.3 Evaluation of additional charge windows for upcoming expensive times

```
1: procedure EvaluateAdditionalChargeWindow
2:     Input: current_time, current_battery_capacity, price_data, consumption_data, production_data, extreme_times
3:     Output: True (if additional charge is needed), False (otherwise)
4:     next_extreme_time ← None
5:     for each extreme_time in extreme_times do
6:         if extreme_time > current_time then
7:             next_extreme_time ← extreme_time
8:             break
9:         end if
10:    end for
11:    if next_extreme_time = None then
12:        return False                          // No future extreme found, no need to charge
13:    end if                                    // find out if the next extreme is an expensive one
14:    if isExpensive(next_extreme_time) then
15:        return False                          // Next extreme is cheap, no need to charge the battery
16:    else
17:        end_time ← next_extreme_time + 1 hour
18:        time_step ← 15 minutes
19:        while current_time < end_time do
20:            net_energy ← production - consumption
21:            current_battery_capacity + = net_energy
22:            if current_battery_capacity < MIN_CAPACITY_LIMIT then
23:                return True        // Battery falls below minimum capacity, charging is needed
24:            end if
25:            current_time + = time_step
26:        end while
27:    end if
28:    return False                              // No need to charge if the simulation is successful
29: end procedure
```

is the limit up to which the battery should be charged. The first step is to check whether there is a point in time in the future when the electricity price will be more favorable than the current price. If such a point in time exists, the net energy is calculated by subtracting the energy consumption from the energy production. This net energy indicates the amount of load required to cover the energy demand until the next more favorable price period. If no future time with a more favorable price is found, the required charging quantity is calculated as the difference between the maximum battery capacity and the current battery status. This quantity is also limited by a fixed limit, which represents the maximum capacity of the battery. In the end, the algorithm delivers the calculated amount of energy required to optimally charge the battery. The algorithm ensures that the battery is charged as efficiently as possible by identifying favorable charging times and adjusting the current capacity limits of the battery in order to charge more of the cheaper electricity, if possible.

Algorithm 4.4 Determine optimal charging amount

```
1: procedure DetermineOptimalChargingAmount
2:     Input: current_time, price_data, consumption_data, production_data, battery_status, max_battery_capacity, max_hours (default 12)
3:     Output: required_charge
4:                                                          // find cheaper prices in the future
5:     if next_cheaper_price exists then
6:         net_energy ← production_data − consumption_data      // just the data until the next cheaper price
7:         required_charge ← net_energy
8:     else
9:         required_charge ← min(max_battery_capacity − battery_status, MAX_CAPACITY_LIMIT − battery_status)
10:    end if
11: end procedure
```

Algorithm 4.5 Evaluation if the next cheapest time can be reached

```
1: procedure CanReachNextCheapest
2:     Input: current_time, current_battery_capacity, consumption_data, production_data, cheapest_times
3:     Output: True (if next cheapest time can be reached), False (otherwise)
4:     next_cheapest_time ← None
5:     for each cheapest_time in cheapest_times do
6:         if cheapest_time > current_time then
7:             next_cheapest_time ← cheapest_time
8:             break
9:         end if
10:    end for
11:    if next_cheapest_time = None then
12:        return False                                     // No future cheapest time
13:    end if
14:    end_time ← next_cheap_time
15:    time_step ← 15 minutes
16:    while current_time < end_time do
17:        net_energy ← production_data[current_time] - consumption_data[current_time]
18:        current_battery_capacity + = net_energy
19:        if current_battery_capacity < MIN_CAPACITY_LIMIT then
20:            return False    // Battery falls below minimum capacity, can't reach next cheapest time
21:        end if
22:        current_time + = time_step
23:    end while
24:    return True                          // Sufficient battery capacity to reach next cheapest time
25: end procedure
```

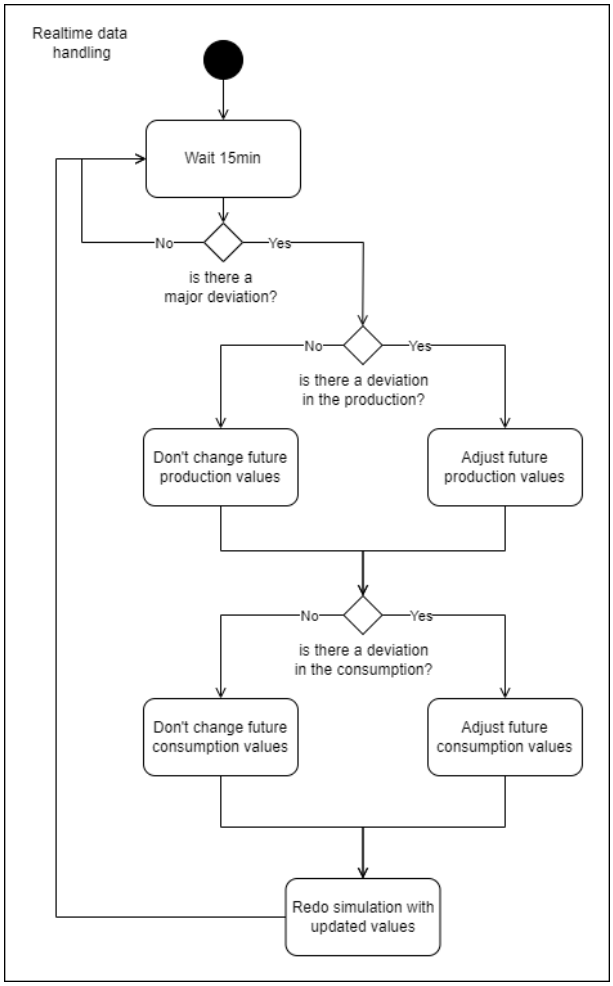

Figure 4.5: Concept of handling incoming real-time data

Algorithm 4.5 is a boolean function that queries whether the next most favorable time can be reached under current conditions. Both the current time and the current SoC are required as input as well as the production and consumption data along with the list of the cheapest times. The output corresponds to a boolean value that indicates whether the next most favorable time can be reached (True) or not (False). It starts by looking at whether there is still a most favorable time in the future from the current point in time. If this is not the case, False is still output as the charging plan should not simply sell everything left in the battery at the end. After all, the whole process starts all over again the next day and it would be disadvantageous to start the day with an empty battery. From here on, the process follows the same steps as in most of the previous algorithms: it is divided into 15-minute blocks, the net energy is determined and then it is simulated step by step how far the battery charge can be sustained. This is done until the next most favorable time. If this is achieved with a battery charge greater than the set limit, True is output and energy can be made available for sale, otherwise, it is not.

Figure 4.5 presents the continuous improvement of the algorithm based on incoming real-time data. This data is sent by the MOE system every 30 seconds and then summarized into 15-minute intervals. This is done because the algorithm processes data in 15-minute steps, which simplifies the comparison.

After a 15-minute block is processed, it is checked whether the real-time data deviates significantly from the algorithm's predictions, with a distinction made between production and consumption. If no major deviations are found, the simulation is not rerun and the process simply continues with the next 15-minute block.

However, if at least one of these two aspects shows major changes, a more detailed check is performed, and the respective values are adjusted for future iterations of the algorithm. Once the simulation has been rerun with the new, adjusted data, the same process is repeated from the beginning but now the adjusted load plan is used to compare with the real-time data. This is to ensure that potential unforeseeable events, that could affect consumption or production, are also accounted for.

5 Evaluation

In this chapter, a distinction is made between two basic types of experiments. Firstly, there is the evaluation of the algorithm, where a comparison with other algorithms is used to see how effective it is. Further, the verification of the importance of the feedback loop of the DT is shown.

5.1 Algorithm evaluation

In this section, the created algorithm is compared to two other ones, which simulate the battery life and further the energy costs over the same period. When looking at a single day it is not possible to determine what is normal and what is just a special exception, which is why the comparison period should be extended to a longer period. A 30-day period was selected, specifically from 12^{th} of April to 12^{th} of May, as all relevant data had already been collected during this time frame. This choice was made because obtaining retrospective data, especially for production, can be challenging, making it practical to utilize the data already gathered by the company for this period. The algorithms being compared include one that was developed, a simple one, and a computationally complex one.

All three algorithms were specifically created for this comparison. The basic algorithm charges the battery at the cheapest times and then discharges it until the battery runs out of charge. When this is the case the required energy is drawn directly from the grid. The complex algorithm, like the others, was designed to optimize loading during favorable times within the specified period. The intervals between these favorable times are divided into 15-minute blocks with each block assigned a state: 'charge,' 'discharge,' or 'grid.' Each block then performs the corresponding action either charging, discharging, or drawing energy directly from the grid. Further, every possible combination is tested always taking the most favorable one per period. The problem with this was that it was far too computationally complex, making the number of possible combinations far too large. No hardware systems with the necessary computing power were available. As a result, the size of the blocks was increased to hourly intervals. When doing that, it reduces the accuracy. Now despite still having a large number of possible combinations it is possible to do it in a reasonable time.

Figure 5.1 shows all three algorithms in comparison. Here the red line represents the simple basic algorithm. It is clearly visible that, most of the time, this line indicates higher costs compared to the other two, signifying that the prices for those days are generally more expensive. On some days the lines are nearly the same but on others, the gap is much wider. There are even some occasions, especially in the beginning where the basic algorithm is cheaper than one of or even both the other ones. The complex algorithm is represented by the blue line, whereas our own has the color green. The two stand out a little from the other one, as they often compete for the lower price, which is usually a little below the red line. There are days or hours when the complex algorithm results in lower prices per kWh but this advantage is typically short-lived, as quickly afterward, the green line

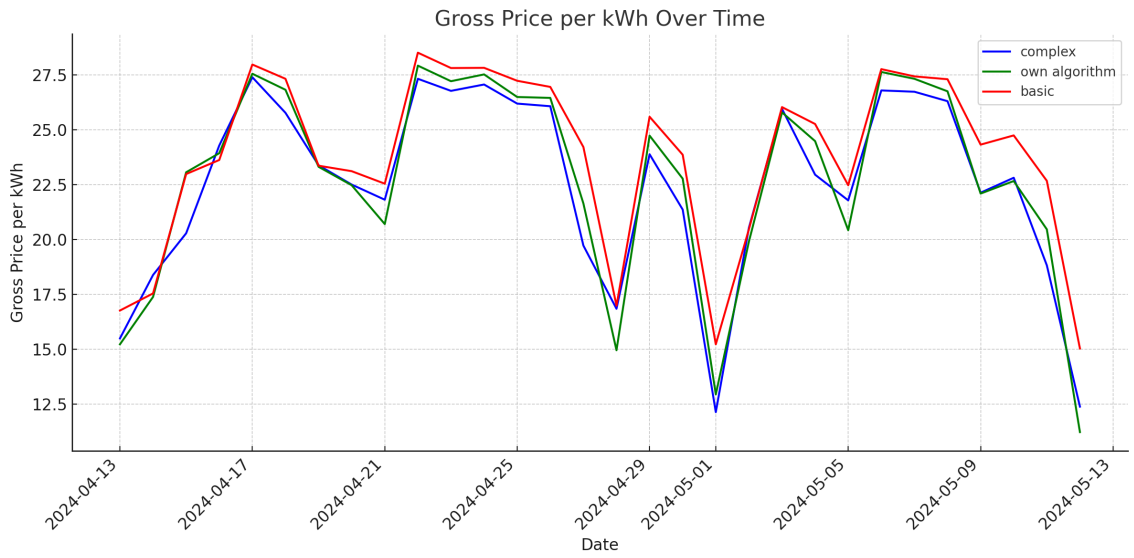

Figure 5.1: Price comparison of the three algorithms

undercuts it, indicating even lower prices per kWh or vice versa. If you look at the average gross prices per kWh of the three algorithms you can also see similar results. The basic algorithm has an average price of 23.77 ct/kWh whereas ours has 22.74. The most complex one is also the cheapest with an average of 22.47 ct/kWh. This is reflected in the graphic 5.1.

5.2 Importance of the Adaptations

To test the effectiveness of the system's self-adaptation, the key factor here is not how low the price is but how close the actual price is to the price of the algorithm's simulation. Since our primary interest lies in reducing costs we compare the two versions, one with adaptation and one without, by evaluating the net electricity costs incurred during the period. For this experiment, the self-adapting algorithm was run for 17 hours at a time. Additionally, the version without customization was executed for comparison. The period starts at 7 a.m. and lasts until midnight. These are the hours where production has the most influence on the entire process.

Determining the effectiveness of the adaptations can be challenging, which is why we decided to simulate both scenarios and compare them with the actual outcome, where the battery is charged and discharged according to the plan over the specified period using actual consumption and production data. The differences between the two variants were not very significant in our example so none of the quarter-hour slots were given a different status as was the case with the other algorithm.

Over the 17 hours, the real-time production and consumption data was either loaded from the battery, fully charged, or simply noted, depending on what the DT specified at the respective time. The same data was then also used to execute the previously created charging plan of the non-adaptable algorithm to compare which version more accurately reflects the real-time data. The algorithm that results in a net cost closer to the actual cost of using the energy, including charging the battery, is considered the better one.

The previously created charging plan has a net cost, i.e., the energy required when charging or using the line, multiplied by the current hourly prices, of 29.67 ct. Whereas the adaptive one calculated a cost of 30.13 ct, which is close to the actual cost of 30.03 ct.

5.3 Discussion

In this section, we discuss our produced results and possible explanations.

5.3.1 Evaluating the Digital Twin Development

If someone completely inexperienced is using the AWS system for the first time it can trigger a completely overwhelming reaction. It might also take some time and dedication to familiarize oneself with the system but once you get the hang of it a lot of things work relatively automatically and the system itself often leads the right way. Once it is clear what services are needed to build a DT the implementation is the easier part. All the different services are connected to one another making it easy to automatically send the incoming real-time data to the correct database or the IoT SiteWise, where it is needed.

That is also the service that handles real-time data collection, processing, and monitoring, tasks that would otherwise require custom-built pipelines and monitoring systems. The respective service indicates directly if any information is missing or if some information is not mandatory. Similarly, AWS IoT TwinMaker simplifies the integration of data sources and the visualization of physical systems, reducing the need for custom development in these areas.

Sometimes, however, when something does not work it is unclear exactly what the problem is and it is not always easy to find the right service. This can make troubleshooting more challenging as the complexity of the AWS ecosystem may obscure the source of an issue, particularly when trying to adapt your architecture to fit AWS's specific workflows.

5.3.2 Answering the research question

RQ1 How efficient is the algorithm in terms of computational speed and effectiveness in optimizing energy usage?

As shown in Figure 5.1, the algorithm performs well in comparison to other algorithms. The developed algorithm is less computationally demanding than the complex one and, on average, cheaper than the simple one making it both effective and efficient. This indicates that the algorithm successfully balances speed and optimization making it a practical choice for managing energy usage without requiring extensive computational resources.

RQ2 How much money can an average German household save by using this DT-based approach?

With our algorithm, the average price is about 23 ct/kWh. Even if the price difference between the algorithms themselves does not seem especially significant it can lead to a bigger difference over a longer period of time. The comparison between our algorithm and the basic algorithm shows a price difference of one cent, which amounts to a price difference of 24 € for an annual consumption of 2400 kWh, which is what has been calculated. Only when compared with the static electricity tariff, which has a price of 35.9 ct/kWh for long-term customers [Hol13], the difference of over 13 ct results in annual savings of over 300 €. For an average German household consuming 2400 kWh per year which is our baseline consumption, this DT-based approach results in annual costs of approximately 550 € whereas the version using the static tariff is around 860 €. This means that using our approach an average German household could save approximately 300 €, which amounts to around 35 % in cost reductions.

RQ3 How effective is the self-adaptation of the DT in optimizing energy management compared to a non-adaptive approach?

When it comes to the self-adaptive part of the algorithm, the results do not show any major difference between the various loading plans generated. The reason for this is probably the short experiment period. However, even with only 17 hours of experimentation, the results were closer to the real events after adjustments. The effects could possibly also increase with a longer test period. As soon as the loading plan after the adjustments differs from the one before the differences should also become clearly visible. These findings suggest that the self-adaptive capabilities of the DT are promising, particularly over extended periods, where the ability to adjust based on real-time data can offer a significant advantage over non-adaptive methods.

6 Conclusion

6.1 Summary

The developed architecture of a DT for bidirectional charging demonstrates an innovative approach to reduce electricity costs by optimizing charging and discharging cycles. The DT accurately replicates the physical system, utilizing real-time and forecast data to simulate various scenarios and evaluate performance under different conditions. The optimization algorithm dynamically adjusts the battery system's charging and discharging strategies based on variable electricity prices, projected energy demand, and production forecasts. This is achieved through continuous adjustments of charging plans that respond to current consumption and production data.

The algorithm was compared with a simpler and a more complex algorithm to assess its efficiency. Results show that the developed algorithm strikes a good balance between computational effort and cost savings. It is less computationally intensive than the complex algorithm while demonstrating higher efficiency than the simple approach. This makes the developed algorithm not only more practical but also nearly as effective as the more resource-intensive alternative. A significant advantage of the developed algorithm is its adaptability to real-time data, allowing it to respond more effectively to changes in energy production and consumption. This adaptability enables closer alignment with predicted outcomes contributing to the optimization of energy costs. The combination of the DT and the algorithm offers valuable insights for software developers and architects in the field of energy management systems, demonstrating a practical method for integrating DTs into renewable energy applications.

However, there are limitations. The approach does not perform equally well in all settings, especially in scenarios with highly variable data or where real-time adjustments become too complex. Additionally, there are constraints regarding the scalability and the types of energy systems that can be accurately modeled. Several important lessons were learned during the implementation and evaluation of the system, particularly regarding the integration of various systems and the management of real-time data flow. Future work could focus on extending the adaptability of the algorithm, improving computational efficiency, and applying the methodology to larger or different energy systems. The results of this thesis underscore the practical and innovative use of DTs to optimize energy costs in bidirectional charging systems, highlighting the potential of such technologies as sustainable solutions in home energy management.

6.2 Benefits

The outcome of this thesis is particularly beneficial for software developers and architects working in the field of energy management. The adaptability of the DT to real-time data provides valuable insights into the operational efficiency of renewable energy systems enabling stakeholders to make informed decisions about energy usage and cost optimization. The approach outlined can be extended to various other applications where real-time adaptive control is crucial.

6.3 Limitations

The developed solution shows its strengths, particularly in scenarios with unpredictable and changing conditions but there are several limitations that must be considered. The algorithm's performance can be significantly impacted if household consumption behavior deviates substantially from the standardized load profiles used in the model. Households with highly irregular or non-typical consumption patterns may experience less accurate predictions, which can reduce the overall effectiveness of the optimization.

Moreover, the algorithm assumes ideal conditions for PV installations such as optimal alignment towards the south and minimal shading. If the PV system is not correctly positioned—e.g., facing a different direction or being partially shaded—energy production forecasts may not accurately reflect real conditions, leading to suboptimal energy management decisions and potentially diminishing the cost-saving benefits.

Another limitation is related to the battery size used in the tests. The analysis was conducted with a single 1500 Wh battery, which may not fully capture the bidirectional functionality of the system. A larger battery or a system with lower consumption would likely enhance the system's ability to store and utilize energy more effectively, thereby reducing the average price per kWh. However, expanding battery capacity would also involve additional costs prompting the need to evaluate whether such investments are economically justified.

6.4 Lessons Learned

Throughout the development and evaluation of the DT for bidirectional charging, several valuable lessons were learned, particularly in the areas of system integration, data management, and algorithm adaptability.

One important lesson was the significance of proper data integration and real-time processing. Working with AWS services revealed both the advantages and complexities of its highly interconnected tools. The platform offers a vast array of services that can greatly facilitate system setup but only when configured correctly. Initially, identifying the right services and setting them up to communicate effectively posed a challenge due to the overwhelming number of options. The configuration process required careful planning and attention to detail. However, once the system architecture was correctly established, the seamless integration between components underscored the strength of a well-structured cloud-based solution.

The adaptability of the algorithm was another significant area of learning. The DT's ability to adjust charging and discharging cycles in response to real-time data was designed to enhance its effectiveness. However, this adaptability is highly dependent on the accuracy and timeliness of the input data. Any delays in data updates or inaccuracies in forecast data could lead to suboptimal decisions. This highlighted the need for robust error handling and fallback strategies within the DT to manage unexpected scenarios such as sudden drops in solar production or unexpected spikes in household consumption. Additionally, implementing continuous feedback loops allowed the system to refine its actions and maintain closer alignment with actual conditions but it also increased the computational complexity of the algorithm.

Moreover, the experience underscored the importance of balancing computational effort with the need for accuracy in optimization. The complex algorithm, while potentially more precise, proved too resource-intensive and computationally expensive to be practical for the targeted application. This realization drove home the need to find a middle ground where the algorithm could operate efficiently without demanding excessive computational resources. The chosen approach managed to strike this balance but further refinement is needed to ensure scalability, especially if applied to larger or more complex energy systems.

Overall, the lessons learned from this thesis provide valuable guidance for future developments in DT-based energy management systems, emphasizing the need for careful planning, precise system integration, and a balanced approach to algorithm design.

6.5 Future Work

Future work could focus on enhancing the consumption forecast component to improve the overall accuracy of the algorithm. One approach could involve developing different user profiles that represent various consumption patterns typical of different household types such as single-person apartments, families, or shared living spaces. By tailoring forecasts to these distinct profiles, the system could better anticipate energy consumption, leading to more precise and effective charge-discharge strategies.

Further improvement could be achieved by incorporating machine learning models trained on collected consumption data from individual households. These models could learn regular patterns and habits in energy usage, such as peak consumption times in the morning or evening, weekend behaviors, or seasonal variations. By analyzing past consumption behaviors, the models could predict future demand more accurately, capturing the unique routines and preferences of each household. This approach would allow the algorithm to make more informed decisions based on precise, real-time data, enabling a more dynamic and adaptive energy management system that continuously learns and adjusts to changes in household behavior. As a result, the algorithm's outcomes would align more closely with real-world conditions, leading to improved accuracy in managing energy consumption and maximizing cost savings.

In addition to improving consumption forecasting, another area for future work could focus on refining how the system handles production forecasts for PV installations. Currently, the algorithm assumes optimal conditions such as proper south-facing alignment and minimal shading. However, if the PV system is not ideally positioned or is partially shaded, the energy production forecasts might not accurately represent the actual output, leading to less effective energy management

decisions. Future work could explore methods to detect reduced production due to such factors and automatically adjust the system's parameters or model to reflect these conditions. This adjustment could also involve informing the user about the change, thereby enhancing user engagement and ensuring informed decision-making.

Additionally, there is potential to refine the algorithm itself by aligning it closer to the more complex models without significantly increasing computational demands. This could involve exploring optimization techniques or integrating machine learning models that assist in decision-making and in doing so enhancing the algorithm's efficiency. Machine learning could help with identifying optimal charge and discharge points more accurately or in simplifying the decision process where traditional methods might be computationally intensive. By combining data-driven approaches with improved algorithmic design, future iterations of the system could achieve higher performance levels, making the DT even more effective in reducing energy costs and optimizing resource use.

Acknowledgment

The authors would like to thank Patrick Spaney and Floriment Klinaku for supervising this thesis as well as their feedback and helpful advice. We also thank Prof. Dr. Steffen Becker for his contributions, help and feedback.

Bibliography

[ARS15] P. Arcaini, E. Riccobene, P. Scandurra. “Modeling and analyzing MAPE-K feedback loops for self-adaptation”. In: *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE. 2015, pp. 13–23 (cit. on p. 6).

[Bas20] R. Basmadjian. “Optimized charging of pv-batteries for households using real-time pricing scheme: A model and heuristics-based implementation”. In: *Electronics* 9.1 (2020), p. 113 (cit. on p. 9).

[BDH+20] P. Bibow, M. Dalibor, C. Hopmann, B. Mainz, B. Rumpe, D. Schmalzing, M. Schmitz, A. Wortmann. “Model-driven development of a digital twin for injection molding”. In: *International Conference on Advanced Information Systems Engineering*. Springer. 2020, pp. 85–100 (cit. on pp. 6, 15, 16).

[BGK21] M. Bertoluzzo, S. Giacomuzzi, A. Kumar. “Design of a bidirectional wireless power transfer system for vehicle-to-home applications”. In: *Vehicles* 3.3 (2021), pp. 406–425 (cit. on p. 7).

[BMD13] S. Breukers, R. Mourik, B. DuneWorks. “The end-users as starting point for designing dynamic pricing approaches to change household energy consumption behaviours”. In: *Report for Netbeheer Nederland, Project group Smart Grids (Pg SG). Arnhem: March* (2013) (cit. on p. 1).

[Bos16] C. Boström. *Optimization of a Household Battery Storage: The Value of Load Shift*. 2016 (cit. on p. 9).

[BP17] L. Burckhardt, M. Pehnt. “Plug-in-Photovoltaik in Deutschland: Eine technische, ökonomische und soziale Analyse”. In: *Energiewirtschaftliche Tagesfragen* 67 (2017), pp. 48–52 (cit. on p. 13).

[Bun24] Bundesnetzagentur. *Smard*. https://www.smard.de/home. Accessed: "01.08.24". 2024 (cit. on p. 15).

[CG09] S. Chen, H. Gooi. “Scheduling of energy storage in a grid-connected PV/battery system via SIMPLORER”. In: *TENCON 2009-2009 IEEE Region 10 Conference*. IEEE. 2009, pp. 1–5 (cit. on p. 1).

[Cha24] E. Charts. *Energy-Charts*. https://api.energy-charts.info/#/prices/day_ahead_price_price_get. Accessed: "01.08.24". 2024 (cit. on pp. 15, 25).

[Clo11] A. E. C. Cloud. “Amazon web services”. In: *Retrieved November* 9.2011 (2011), p. 2011 (cit. on pp. 7–9).

[CM13] U. C. Chukwu, S. M. Mahajan. “Real-time management of power systems with V2G facility for smart-grid applications”. In: *IEEE Transactions on Sustainable Energy* 5.2 (2013), pp. 558–566 (cit. on p. 7).

[Cor24] M. Corporation. *Microsoft Azure*. https://azure.microsoft.com/. Accessed: "14.05.24". 2024 (cit. on p. 6).

[DJR+22] M. Dalibor, N. Jansen, B. Rumpe, D. Schmalzing, L. Wachtmeister, M. Wimmer, A. Wortmann. "A cross-domain systematic mapping study on software engineering for digital twins". In: *Journal of Systems and Software* 193 (2022), p. 111361 (cit. on p. 5).

[EK20] J. Eum, Y. Kim. "Analysis on operation modes of residential BESS with balcony-PV for apartment houses in Korea". In: *Sustainability* 13.1 (2020), p. 311 (cit. on p. 10).

[ELS22] H. N. Esfahani, Z. Liu, Z. Song. "Optimal pricing for bidirectional wireless charging lanes in coupled transportation and power networks". In: *Transportation Research Part C: Emerging Technologies* 135 (2022), p. 103419 (cit. on p. 7).

[Gmb24] S. N. GmbH. *SWKiel*. https://www.swkiel-netz.de/strom/kennzahlen/lastprofile. Accessed: "01.08.24". 2024 (cit. on pp. 15, 19, 20).

[Hol13] L. M. Holm. *strom-report*. https://strom-report.com/strompreise/strompreisentwicklung/. Accessed: "25.09.24". 2013 (cit. on p. 34).

[Jam16] D. N. E. James Luffman. *Solcast*. https://solcast.com/. Accessed: "01.08.24". 2016 (cit. on pp. 15, 20).

[JDNB18] S. Jadhav, N. Devdas, S. Nisar, V. Bajpai. "Bidirectional DC-DC converter in solar PV system for battery charging application". In: *2018 international conference on smart city and emerging technology (ICSCET)*. IEEE. 2018, pp. 1–4 (cit. on pp. 1, 7).

[JH18] N. I. Jaya, M. F. Hossain. "A prototype air flow control system for home automation using mqtt over websocket in aws iot core". In: *2018 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*. IEEE. 2018, pp. 111–1116 (cit. on p. 8).

[KEK20] C.-N. Kim, J.-Y. Eum, Y.-K. Kim. "Analysis of Energy Saving Effect of the Residential BESS Connected to the Balcony-PV in Apartment Houses". In: *Journal of the Korean Solar Energy Society* 40.3 (2020), pp. 21–31 (cit. on p. 10).

[KKT+18] W. Kritzinger, M. Karner, G. Traar, J. Henjes, W. Sihn. "Digital Twin in manufacturing: A categorical literature review and classification". In: *Ifac-PapersOnline* 51.11 (2018), pp. 1016–1022 (cit. on pp. 5, 6).

[Lam22] A. Lambda. *Aws lambda*. 2022 (cit. on pp. 7–9).

[LPT+21] D. Lehner, J. Pfeiffer, E.-F. Tinsel, M. M. Strljic, S. Sint, M. Vierhauser, A. Wortmann, M. Wimmer. "Digital twin platforms: requirements, capabilities, and future prospects". In: *IEEE Software* 39.2 (2021), pp. 53–61 (cit. on pp. 6, 7).

[MV14] S. Mathew, J. Varia. "Overview of amazon web services". In: *Amazon Whitepapers* 105.1 (2014), p. 22 (cit. on pp. 7–9).

[NFM17] E. Negri, L. Fumagalli, M. Macchi. "A review of the roles of digital twin in CPS-based production systems". In: *Procedia manufacturing* 11 (2017), pp. 939–948 (cit. on pp. 2, 5, 6).

[PMG+13] J. Pinto, V. Monteiro, H. Gonçalves, B. Exposto, D. Pedrosa, C. Couto, J. L. Afonso. "Bidirectional battery charger with grid-to-vehicle, vehicle-to-grid and vehicle-to-home technologies". In: *IECON 2013-39th Annual Conference of the IEEE Industrial Electronics Society*. IEEE. 2013, pp. 5934–5939 (cit. on pp. 1, 7).

[Poo20] N. Pool. *Nord Pool*. https://data.nordpoolgroup.com/auction/day-ahead/prices?deliveryDate=latest&currency=EUR&aggregation=Hourly&deliveryAreas=GER. Accessed: "01.08.24". 2020 (cit. on p. 15).

[QTH+21] Q. Qi, F. Tao, T. Hu, N. Anwer, A. Liu, Y. Wei, L. Wang, A. Y. Nee. "Enabling technologies and tools for digital twin". In: *Journal of Manufacturing Systems* 58 (2021), pp. 3–21 (cit. on p. 6).

[Reh23] S. Rehan. "Industrial IoT with AWS IoT". In: *AWS IoT With Edge ML and Cybersecurity: A Hands-On Approach*. Springer, 2023, pp. 177–210 (cit. on p. 8).

[Ser23] A. W. Services. *AWS*. https://aws.amazon.com/. Accessed: "14.05.24". 2023 (cit. on p. 6).

[SFM20] A. Saad, S. Faddel, O. Mohammed. "IoT-based digital twin for energy cyber-physical systems: design and implementation". In: *Energies* 13.18 (2020), p. 4762 (cit. on pp. 2, 10).

[SK17] P. Sbarski, S. Kroonenburg. *Serverless architectures on AWS: with examples using Aws Lambda*. Simon and Schuster, 2017 (cit. on p. 7).

[SS04] M. Shahidehpour, F. Schwarts. "Don't let the sun go down on PV [photovoltaic systems]". In: *IEEE power and energy magazine* 2.3 (2004), pp. 40–48 (cit. on pp. 1, 7).

[UL06] M. Urbina, Z. Li. "A fuzzy optimization approach to PV/battery scheduling with uncertainty in PV generation". In: *2006 38th North American Power Symposium*. IEEE. 2006, pp. 561–566 (cit. on pp. 1, 2).

[WS15] H. Wirth, K. Schneider. "Recent facts about photovoltaics in Germany". In: *Fraunhofer Ise* 92 (2015) (cit. on p. 1).

All links were last followed on September 03, 2024.