Bachelorthesis

Ervin Victor Madaha
Gervais Usha Mckoy

Integration of Energy Service Components into an Open System for Future Energy Markets

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Ervin Victor Madaha
Gervais Usha Mckoy

Integration of Energy Service Components into an Open System for Future Energy Markets

Bachelor thesis based on the study regulations
for the Bachelor of Engineering degree programme
Information Engineering
at the Department of Information and Electrical Engineering
of the Faculty of Engineering and Computer Science
of the Hamburg University of Aplied Sciences

Supervising examiner : Prof. Dr. rer. nat. Wolfgang Renz
Second Examiner : Prof. Dr.Ing. Andre Wenzel

Day of delivery 7. November 2017

Ervin Victor Madaha
Gervais Usha Mckoy

Title of the Bachelorthesis

Integration of Energy Service Components into an Open System for Future Energy Markets

Keywords

Smart Grid, Lab Test, Communication Infrastructure

Abstract

De-centralized renewable energy generation in houses and farms together with their local consumers and a local energy management system (typically a household control box) can act as prosumers already nowadays. The project OS4ES (Open System for Energy Services) provides a registry-based ICT infrastructure that supports the offering and aggregation of local flexibility to larger energy amounts that the aggregator can trade on energy markets. In future smart grids the registry will act as a market plattform where prosumers can offer their energy services to get booked by the aggregators.
The objective of this work was a lab test where real physical devices offer their flexibility to the registry and get operated by an aggregator instance that has booked that flexibility from the registry. A testbed for the prototypical implementation of the OS4ES has been designed and implemented in order to prepare that lab test. Finally the lab test was executed with a simulated PV-battery system and with a real CHP (Combined Heat and Power) in the Smart Grid Lab of the CC4E in the Energy Campus Bergedorf.

Ervin Victor Madaha
Gervais Usha Mckoy

Titel der Arbeit

Integration von Energie-Service-Komponenten in ein offenes System fuer zukuenftige Energiemaerkte

Stichworte

Smart Grid, Lab Test, Communication Infrastructure,

Kurzzusammenfassung

Dezentrale erneuerbare Energieerzeuger in Haeusern und im laendlichen Umfeld bilden zusammen mit den lokalen Verbrauchern und einem lokalen Energiemanagementsystem (typischerweise Haushalt-Steuerboxen) sog. Prosumer. Das Projekt OS4ES (Open System for Energy Services) konstruiert eine registerbasierte IKT-Infrastruktur, durch die der Prosumer seine Flexibilitaet anbieten und zu vermarktungsfaehigen Mengen aggregieren lassen kann. In zukuenftigen Smart Grids bildet das Register die Marktplattform, auf der Prosumer ihre Energiedienstleistungen anbieten und buchen lassen koennen.
Ziel dieser Arbeit ist die Durchfuehrung eines Labortests, in dem reale physikalische Anlagen ihre Flexibilitaet dem Register anbieten und dann vom Aggregator betrieben werden, nachdem er diese gebucht hat. Um die prototypische Implementation des OS4ES fuer den Labortest vorzubereiten wurde ein Testbed designed und implementiert. Abschliessend wurde der Labortest durchgefuehrt, und zwar mit einem simulierten PV-Batterie-System und mit einem realen Blockheizkraftwerk im Smart Grid Labor des CC4E im Energie-Campus Bergedorf.

Acknowledgment

We would like to thank our supervisor and mentor Prof. Dr.rer.nat. Wolfgang Renz from the Faculty of Engineering and Computer Science, Hamburg University of Applied Sciences. His expertise, guidance and encouragement played a vital role in making our thesis a success.

We would like to extend this gratitude to Dipl.-Ing. Petrit Vuthi, for introducing us to the OS4ES project and providing technical support during the development process at the Energy Campus. We also express our appreciation to Lars Ole Behncke for his time and effort spent in counselling and supporting this work. The final token of gratitude and acknowledgement goes to Sebastian Farrenkopf, M.Eng. Tim Dethlefs, and the whole OS4ES project team at the Hamburg University of Applied Sciences, in each and every role they have taken to help us accomplish our work on this project.

Contents

List of Tables

List of Figures

Abbreviations

CHP	Combined Heat and Power (System)
DER	Distributed Energy Resource
HUAS	Hamburg University of Applied Sciences
JPB	Java Python Bridge
LLN0	Logical Node zero
MQTT	Message Queue Telemetry Transport
OS4ES	Open System for Energy Services
OAA	OS4ES Aggregator Application
PV	Photo Voltaic (System)
URCB	Unbuffered Report Control Block
XMPP	Extensible Messaging and Presence Protocol

1. Introduction

1.1. Motivation

From the rapid growth in technology throughout recent years, there has been an insatiable demand for energy which has led to the current status where there is a massive increase in the number of Distributed Energy Resources (DERs). This increase of DERs occurs mainly as a result of the rise of renewable energies which tend to use multiple smaller modular resources (i.e. solar panels) rather than single larger ones (i.e. power plants). Such DERs can also be decentralized producers that through their local EMS(Energy Management System) will get active prosumers[1] participating in the future energy markets. Having such a high number of DERs getting connected to the grid, this causes an adverse impact to the grid's reliability and robustness. As a consequence it complicates the network management that is handled by the Distribution System Operators (DSOs).[7]

With most of these resources emerging from the renewable sector, as reported by REN21, from 2004-2014 the global capacity of solar power grew from 2.4 GW to 139 GW[15], approximately 58 times its original GW; while the total capacity of renewable energy doubled from 800GW into 1560GW[15]. Increasing the number of resources at such a rate causes some difficulties for the grid to manage and distribute all that energy generated. Some of the problems include the lack of flexibility to switch in between resources, the ability to efficiently meet the demand without wasting energy in surplus and mainly its just that the current grid infrastructure cannot manage these changes.

As a solution to this, some concepts have been proposed to improve the grid's infrastructure. The smart grid, is a grid that allows a bidirectional communication and controls between its components, making it is more reliable, flexible and efficient; thus, resolving the aforementioned issues. Stemming from this is the OS4ES[2] project which aims to provide an open system where energy can be readily offered from various resources and easily acquired by the operators; hence, enabling the dynamic DER and DSO cooperation[7]. It's a collaborative project which is funded by the European Commission under the EU's Seventh Framework

[1] Prosumer : A consumer that also produces energy
[2] OS4ES stands for "Open System for Energy Services"

Programme for research, technological development and demonstration. The project commenced on the 1 July 2014 and was set to run for three years, and to be conducted by nine partners (including the Hamburg University of Applied Sciences) with coordination by FGH e.V.[7]

Based on the OS4ES architecture development under the lead of the supervisor of this thesis, the project has deployed separate work packages to be implemented by different partners, to build up the fundamental components of the OS4ES system[18]. These components are the communication interface, generic smart grid applications for the operators as well as an interface for the DERs. And these components are then integrated on a semantic middleware which is based on the IEC 61850 standard[3] to have a common protocol for all the components of the system.

1.2. Problem Statement

The main objective of this thesis is to prepare, test and integrate the software components built from previous work packages and to incorporate real devices as energy resources into the system. Software components under test, are namely the middleware which serves as a communication bridge for the system components and the registry which is the common marketplace for the energy services. This HUAS[4] lab test aims at putting the OS4ES system into a lab environment to observe how it handles test scenarios with the usage of a CHP system which is dependent on a heat storage and a Battery that charges with solar panels.[5] During the process of conducting this lab test, the fundamental software components that have been developed within the project, can be functionally tested to observe if they can achieve the capabilities they're expected to have for this system.

1.3. Thesis Overview

Here is a brief outline of this thesis document, giving a short explanation on what each chapter contains, to grasp the general overview of the entire thesis.

Chapter 2: The "Open System for Energy Services", is an in-depth elaboration on the project context of this work. This provides an insight into the smart grid concept and also gives a description on the components of the system, along with their roles within the project.

[3]It is the standard set for Power Automation Utility by the International Electrotechnical Commission

[4]HUAS: Hamburg University of Applied Sciences

[5] As a preparation for the final lab test, a test-bed had to be prepared that simulated the physical components by using software modules developed from earlier works of the project.

Furthermore, the communication infrastructure, the test methodology and Lab test are also examined.

Chapter 3: Requirement Analysis, this chapter probes the functional and non-functional objectives of this thesis, as some are expected by the project and others are just flexible on how they could be implemented. The software architecture is presented here, defining how the software should be implemented and the structure to be followed.

Chapter 4: Test Environment and Design, this chapter explains the current technical status of the OS4ES system and the prerequisites for setting it up, as well as the design for the next steps to be taken.

Chapter 5: Implementation is the chapter where detailed explanation of the implementation decisions and the code required to complete the integration of the components for the OS4ES system and fully realise the final test. The main work of this thesis is expressed in this chapter.

Chapter 6: Results and Validation, this chapter summarises and displays most of the results from the lab test; basically it shows how the devices responded and the evaluation of the tests when working on a live system.

Chapter 7: Conclusion, presents the outcomes of this work and the relevance of these results to the entire project as well as some factors that can be improved for future work.

2. The Open System for Energy Services

This chapter introduces the OS4ES project and gives an insight into how the basis of this thesis came into being. It gives an explanation about the OS4ES in terms of the basic concept, the objectives it aims to fulfill and the different components used to realise it. There is also a summarised review of the smart grid as the OS4ES aims to create a system that uses the concept.

2.1. The OS4ES Concept

[E. Madaha]

The Open System for Energy Services (OS4ES) is a concept that is about creating a centralised system on which energy services can be offered and acquired in a flexible and efficient manner. The concept aims to achieve a common marketplace where energy is the main commodity; thus allowing a more dynamic approach to which energy can be sold at any time, to any buyer at the given market.

OS4ES intends to build a system where the energy offered by a single resource unit can be added with other offers from other resources to fit a larger energy profile which can feed a sizeable demand, this is called the aggregation of energy resources. A real case scenario would be an offer of energy from the solar panels coming from a single household, added together with offers of 100 more houses into a single combined offer this would virtually be equal to a small solar farm. The concept allows the aggregation of multiple DER-units to form a larger element that could function as a virtual power plant (VPP).[7]

To realise such a concept, there are components of the system that had to be chosen to fit the implementation of the OS4ES system. First, an insight into the smart grid's features and the type of tools used to realize such a system. Then a brief description of the different roles taken in an OS4ES system and the devices used for the final lab test. Along with that, the overview of the main standard used for this implementation and the methodology of the test description.[18]

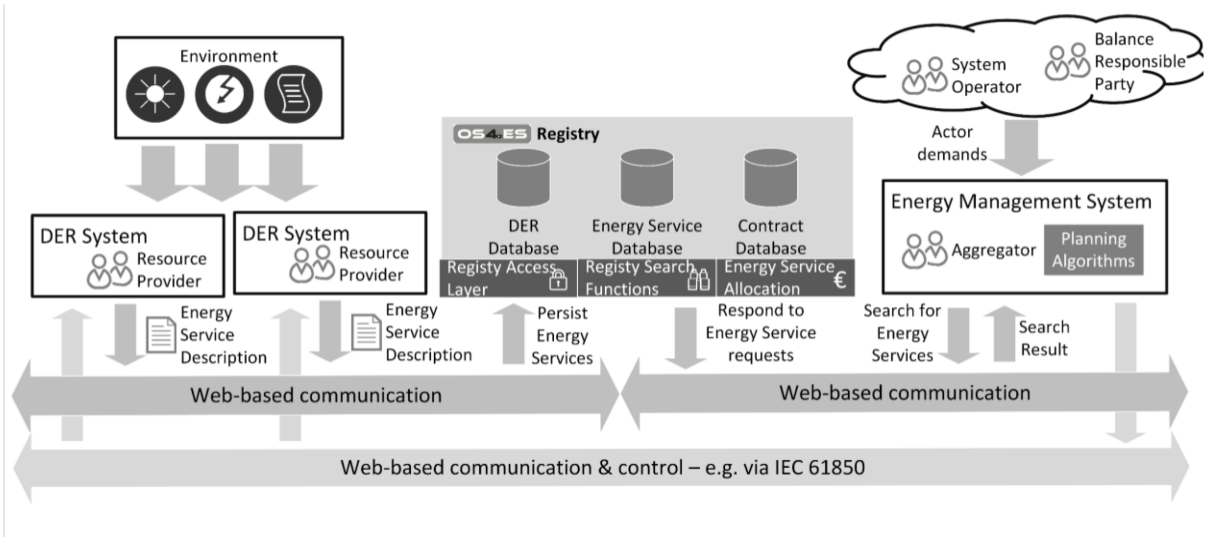


Figure 2.1.: The OS4ES System Architecture[7]

Figure 2.1 describes the entire architecture the an OS4ES system, drawing the workflow from the left, the energy sources, and ending on the right with the energy operators for the end consumers. The whole workflow can be subdivided into three fundamental roles as they will be continuosly referred to throughout this thesis.

Firstly, its the DER-System that is controlled by the Resource Provider, seen on the very left of the diagram this block serves to produce energy from the environment(natural resources) and offers this energy as a service to the OS4ES Registry. The flow goes onto the OS4ES Registry[1] , seen in the middle with a row of databases, this is where most of the information is stored and shared, business-wise it can be seen as the point of exchange, where energy services are booked and sold. Finally, at end the flow is Aggregator, on the far right of the diagram, with this role the operator can take advantage from the availability of the resources at their individual offered prices, to arrive to a suitable energy profile to feed the grid's demand at an optimal cost.

Briefly kept the OS4ES system provides a platform for multiple energy resources (minor and major) to provide their energy to an open market, where different energy distributors can buy their offers based on their availability at the offered price, to feed the demand of the end consumers (i.e. households, offices...).

[1]OS4ES Registry, can also be called Distributed Registry, but will mainly be referred to as simply Registry throughout the rest of the Thesis.

2.2. Smart Grid

[E. Madaha]

This project has its concept emanating from the smart grid theory as it reflects on the similar goals of what the smart grid aims to achieve. Since the OS4ES system fundamentally fits into a smart grid and performs in a way that achieves the features of a smart grid[7], there needs to be an overview of the smart grid concept in order for us to fully fathom the origins of the project and to get an understanding for how the project improves the energy sector as a whole.

To get a full grasp of the term Smart Grid, at a fundamental level, it is required to first split the two words and have the simpler definitions help capture the full meaning of the term. A grid, precisely an electric grid, is a massive network of various electric components (including transmission cables, substations and transformers) that facilitate the transfer of electricity from the power plant to the end users namely homes, offices and other places that run on electricity. The adjective smart on the other hand is used to represent the involvement of enhancing a system or a device to become somewhat intelligent, automated and more independent from direct user control (also usually programmable).[23, 25]

A Smart Grid is an enhanced network of electrical components meant to transfer electrical power from production resources to the consumption side, (electrical grid))while featuring a layer of digital communication which facilitates more flexibility, efficiency and some automation.[23]

From the current boom and shift towards renewable energies, the number of energy sources have also multiplied immensely. With such resources where the supply is not as consistent, being entirely dependent on weather and other natural resources, the supply becomes a lot more dynamic, having the resources complementing each other to satisfy the varying energy demanded on the client side.

This would call for a system which can adjust the supply with an efficient and flexible approach to suit the required demands. The Smart Grid aims to accomplish this by adding some elements to the grid which will enable the system to cope with these demands. Such elements are the ability of the components to communicate with ease, ability to control the resources remotely and to run on an open fluid market.[23]

Relative to this work; the OS4ES system will act as a gateway to different DER-Systems (demand and supply) within a Smart Grid; allowing Smart Grid actors or aggregators to deploy their applications and, through them, to commission and associate with appropriate and necessary energy services[18].

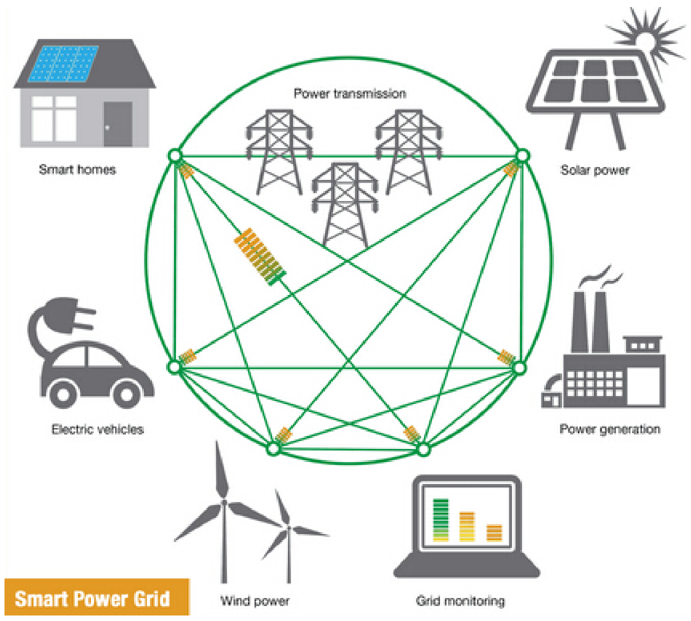


Figure 2.2.: Smart Grid networking[24]

Figure 2.2 shows the networking of the smart grid, exhibiting how interconnected every component is; appearing as a mesh topology of a larger system. This helps to grasp the idea of how the communication would work in this system, given that, traditionally, it is a one way transmission between the different components.

2.2.1. Features of a Smart Grid

[E. Madaha]

The smart grid could be implemented in various ways given the different environments it is based on, but mainly, it should maintain some key features that follow the concept. Below are the main features of a smart grid:

Reliability

[E. Madaha]

The Smart grid has open access to all its vastly spread out components; this allows precautionary vigilance against any forthcoming problems. It improves the ability to detect faults and gives more time to react for such issues. This makes the system more reliable, on the competence to control these issues when they emerge; given that with the normal grid there is usually no time to effectively counter such events. [12]

Flexibility

[E. Madaha]

The traditional supply of energy is normally a one-sided flow, where only the supplier provides electricity to the consumer. However the smart Grid intends to have a bidirectional flow allowing the consumer to feedback some electricity to the grid from a solar panel, battery or any other home energy resource. This makes the grid easily adapt to new providers without having any big challenge or infrastructural changes.[27]

Efficiency

[E. Madaha]

The smart grid is able to manoeuvre through complex situations simply by using intelligent algorithms or by simply offering consumers better alternatives. The computer algorithms can foresee peak times and allow precautionary steps, like keeping more generators in standby or pre-booking sufficient orders. Due to such early planning some costs could be saved and even an overload could be avoided.[4]

It is also more efficient, in that it can inform consumers or consumer smart devices to minimize their consumption when the load is too high on the grid. A good example would be to reschedule the use of less immediate devices such as washing machine or dishwasher. With an openly communicating network this allows a lot more efficiency on the performance of the grid as a whole. Even though not everyone will comply, with more smart homes emerging, this transition will contribute to the efficiency of the grid as a whole.[23]

Sustainability

[E. Madaha]

The system guarantees sustainability due to the easily adjustable infrastructure where renewable elements can be easily plugged in and out without any need to restructure the system as a whole. In the long term, newer and sustainable Energy resources can be added to the system while the old ones are easily removed; thus, sustaining industrial advancements, whilst achieving a cleaner environment.

Market Enabling

[E. Madaha]

The system allows for a competitive market by having flexible prices relative to the demand, which will encourage the producers to have desirable availability for the market. This challenge allows the market to have a competitive edge and keeps the supply sufficiently ready for the forthcoming demand[23]. Consumers will also have more affordable electric bills in that the device will help them prioritize by allowing low priority devices to run when the cost is most affordable. This also helps maintain a more efficient use of energy when the excess consumption is cut down to have a well balanced system.

2.2.2. Technologies of a Smart Grid

[E. Madaha]

To realize the smart grid, some technologies can be extended from the state of the classic electrical grid. The functionality of a smart grid is on the basis of an additional layer of transmission where the communication and control can be eased to allow more flexibility. The following technologies are used to exploit that extra layer and feed in the additional information while also allowing autonomous control of the system.

Sensors, these help track the consumption of utilities, production of various resources and different measurements along the system. In comparison to the traditional grid a lot more sensors and measuring devices have to be used for reading the different points of data at every part of the system. Since the smart grid has to account for everything that occurs within it; the tracking of that information is key to the concept.[23]

Smart meters, are used as regular meters to monitor the normal consumption of the household, although it also works in a two-way communication with the grid. This works as the gateway to each household for the grid, controlling the data coming and going through the system.[23]

Power System Automation, this gives the ability to make quick and precise responses to any faults or interruptions within the system. Additionally, with enough computing power and efficient algorithms, it should be able simulate multiple scenarios to avoid risky or dangerous outcomes for the system.[4]

Distributed Power Flow Control, involves transmission lines with devices attached to them to control the flow of power within. This should help reduce the power lost during transmission, given that a considerable amount is lost during the transmission. The technology would support the better usage of renewables such as wind and solar.[2]

Additionally, an actual adaptation of the wiring would be necessary to accomplish this, as the normal wiring only carries power without any data. Nowadays there are viable alternative solutions to this issue such as the use of power line technologies[14].

Other sections of the grid that need further improvement: substation automation, distribution automation, energy management systems, wireless networking, Power Line communications[14] and fibre optics. With this sections completely integrated it would allow real-time control, asset utilization and security.

2.2.3. Further works on the Smart Grid

[E. Madaha]

Various researches have been set on the concept spanning from academic institutions such as the University of California on launching the Smart Grid Energy Research Centre (SMERC)[26] in 2010, and other universities have conducted their own research.

One of the earliest deployments of the Smart Grid concept was done by Enel in Isernia, Italy, where the company had also produced the meters and the software required for the installation[8]. Other works can be found in Mannheim, Germany where the purpose is to use the Broadband Powerline (BPL) communications in the Model City Mannheim "MoMa" project[16]. Also, the US Department of Energy have launched the ARRA Smart Grid Project[9] with a $ 4.5 billion investment , which should be a driving force for the deployment of the smart grid in multiple locations and eventually nationwide.

2.3. Devices and Roles

[E. Madaha]

The OS4ES system facilitates the networking of multiple elements to conduct an exchange of energy services. To understand the system better there needs to be an elaboration on the nature of these elements. The elements can be grouped into three main categories: the elements that acquire the energy services are called aggregators; the elements providing the energy are the DER-Systems; the element enabling this exchange is the registry. Here the devices used to test the OS4ES system will also be introduced and their functionalities briefly explained.[18]

2.3.1. DER-System

[E. Madaha]

A DER-System is owned by an entity that has the role of a resource provider and may include storage as well. A DER-System can be a single DER-unit or it can be multiple DER-units of various types (generating, consuming and storing).[18, p.24]

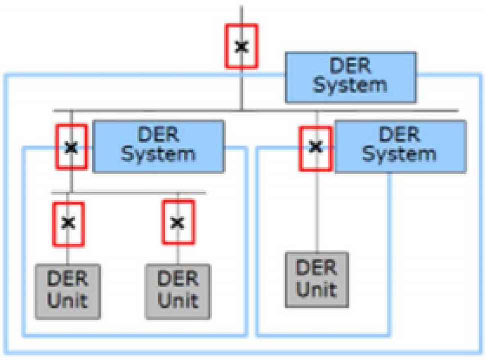


Figure 2.3.: A DER-System consisting of DER-Systems and DER-units [18]

Figure 2.3 represents the inner blocks of a DER-System and that it may be build from smaller DER-Systems or a number of DER-units (even a single unit). This also demonstrates that multiple resources can also be compounded as a single system.[18]

DER (Distributed Energy Resource)

[E. Madaha]

A DER-unit is an individual appliance (DER) or a group of DERs of the same type, for instance an array of PV cells is considered a single DER-unit, that actually produces energy and/or consumes energy and can be accessed through a DER-System[18, p.24]. For the tests conducted during this work the DER-units were a PV, CHP generator and a Battery.

PV

[E. Madaha]

Photo-voltaic cells otherwise known as solar cells are semiconducting materials used to generate electricity from light, by a process called the photoelectric effect. Solar power has been a good alternative in the energy sector, since it is a clean and fuel-less Energy resource.

The PV tends to work along with a converter, since the electricity generated is DC and needs to be converted into AC to be fed into the grid or the household. Alternatively, when the power generated exceeds the consumption or demand from the grid it can then be stored in a battery and preserve the energy for later use. The PV system functioning as the PV-Converter-Battery works well, as it allows greater flexibility for the smart grid's performance and also maintains reserves for night time consumption.

CHP

[E. Madaha]

A combined heat and power plant is a system for electricity production where the on-site heat energy, that would usually be lost, is captured and utilized for heating purposes. Such a system plays a key role in having sustainable and efficient energy resources in the future energy market, as it utilizes a better model which carries over 30% improvement.[10]

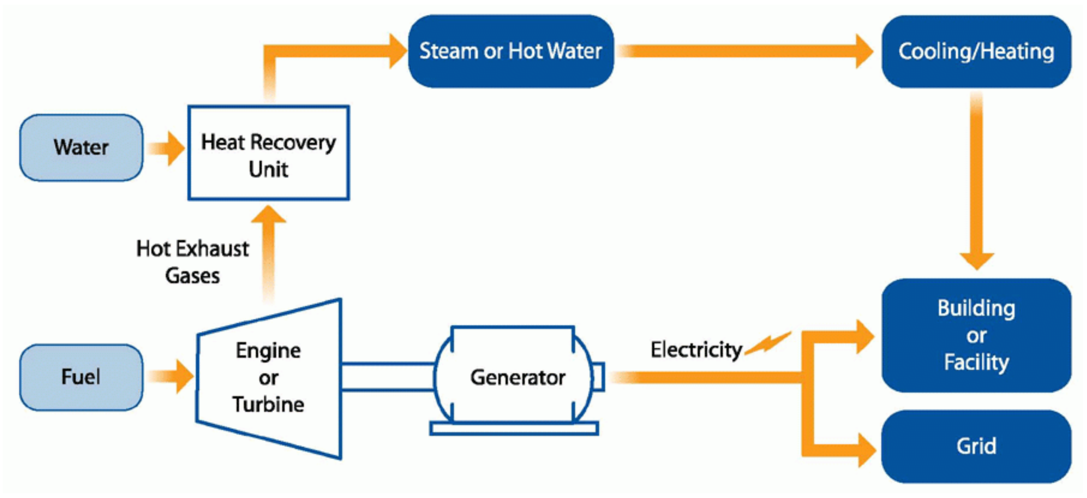


Figure 2.4.: The flow of a CHP setup [10]

Figure 2.4 shows the energy cycle of a CHP system; on how the water and fuel feed the system, and how the output splits into a path for the heating system and the electricity. Finally the end user is either the facility or the supply to the grid, as the flow shows there is no point where the energy is lost, thus, maintaining a high efficiency.[10]

A CHP can be beneficial to multiple Smart Grid concepts such as the micro-grid where the circulation or distribution is highly minimized. Here the heat produced by the generators can be reused and fed into the smaller sized system (i.e. School Campus). Further on, the device could also be utilized with a Heat grid concept which is another upcoming topic in the energy sector.

The final constraint for the device is the balance between the two; when more electricity is demanded than Heating or vice versa, but the use of the heat storage Systems and a battery could act as a buffer to store the surplus energy. Further optimisation can also be done on forecasting and simulating optimal scenarios; where such a problem could be solved under the functionalities of a smart grid.

Battery

[E. Madaha]

Batteries are essential in the shift towards renewables, as they serve as a bridge between non-deterministic resources; wind/solar and the varying demand from the grid. The battery can be used in mainly two ways: one, where it can collect the surplus power from the grid, and two where it can be used as a storage for renewables. Both methods improve the efficiency of the system where the battery acts as an additional buffer that saves the extra energy which would otherwise be lost.

The battery is also used in electric vehicles and in such cases the application can be the same as above. When the car is plugged in, the grid could buy the electricity during peak times and normally charge the car when the load is balanced or low. Assuming the case of multiple cars being plugged in at once, the grid could effectively aggregate them into a virtual power plant functioning similar to a battery storage power station.

Batteries are continuously worked on and improved. New alternatives from the typical Li-ion are also researched and tested, switching towards different elements like using Hydrogen as a form of energy storage; others include hydroelectricity, superconducting magnets and even thermal storage.

2.3.2. Aggregator

[E. Madaha]

Along with the DERs there are the aggregators, that plays the business role on this framework. They are responsible for: analysing the data from the DER portfolios, implementing specific control strategies over the appropriate DER clusters and exploiting the capabilities of these DER clusters.

The aggregator serves the purpose of exploiting the capabilities that OS4ES wants to offer. They have access to a number of energy resources where they can collect all of their capacities combined and effectively feed the grid demand with cheaper, flexible and, if possible, cleaner means.

The aggregator interacts with the OS4ES system via a separate application, in order to perform the role on the grid; they can also be called smart grid actors.[18]

2.3.3. Registry

[E. Madaha]

The distributed registry system, referred to as registry for short, is the system which enables the publishing of energy services and the discovery of DERs. This is also where the aggregator can view and book the energy services offered by different DERs. This is a key component of the OS4ES system, essentially it is the energy marketplace that this project aims to create for the trading of energy services.[18]

The registry's workflow allows a DER-System to register itself and publish the schedule for the available energy. A registered aggregator can then book a schedule and agree on a contract for the energy service.

The registry will have the security features required such as authentication, confidentiality, integrity control and availability. It will incorporate the use of secure and reliable protocols for the communication and data management which establishes a trustworthy system that supports information privacy.[18]

2.3.4. Other roles

[E. Madaha]

Apart from the key roles and actors mentioned above, there are other roles that are also in the system that also deserve to be mentioned:

- Transmission System Operator (TSO): it is the provider of the conventional grid equipment, which is responsible for power transmission; especially when high voltage electricity is transferred.[6]
- Distribution System Operator (DSO): it is the entity that manages medium voltage transmission. It also behaves as the mediating actor between the TSO and the aggregator, selling the energy to the latter. [6]
- BRP, Balance Responsible Party, is responsible for the timely supply of schedules and for balancing the energy; as in the amount of energy scheduled for consumption or production, should match the energy consumed or produced at the end.[18, p. 29]

- "Prosumer": while traditional energy consumers are present in the Smart Grid as well, they are also able to obtain energy from their own equipment (solar panels, small-sized windmills, etc.) thus, becoming an actor able to produce and consume energy; hence, the term "prosumer" (producer + consumer). [18, 6]

These roles are not necessarily separated from the main role since these entities could also perform some actions of the main roles. For instance, a prosumer is basically a DER that is producing and also consuming at the same time from the grid and the DSO and BRP are roles that can also perform aggregation.

2.4. Communication Interface

[E. Madaha]

The OS4ES relies on having a well defined communication interface since the core purpose of the whole system is to enable the flexible networking between energy resources and the grid. To accomplish this, the IEC 61850 standard was chosen to be the protocol, which would run under a semantic middleware of the system.

2.4.1. IEC 61850

[E. Madaha]

The IEC 61850 standard is a Power Utility Automation Standard built in cooperation with manufacturers and users to create a uniform, future-proof basis for the protection, communication and control of substations.[13]

This standard was set to have a single protocol for the complete substation, with defined services to transfer data so that the communication mapping can be robust. The standard aims to promote high inter-operability between systems and different vendors by using a common data format.[13]

This standard is used in the OS4ES system to maintain a common protocol on which all sides can communicate and understand each other with ease. It was chosen because it was already the standard set for power automation and the OS4ES intends to work with automated power systems.[18]

Core Features

[E. Madaha]

- The standard aims to be an object model that describes the information of the various primary equipment of the substation and the automation functions available. This includes abstract definitions of services, data and a common data class independent of underlying protocols.[29]
- Specifies the communication between intelligent electronic devices (IED) with substation automation systems.[29]
- Acts as a configuration language for exchanging configuration information.[29]

Data Model

[E. Madaha]

The data model of the IEC 61850 follows a tree structure that allows the device to have different types of data objects that can hold attribute like schedules, control commands and the status values. [29]

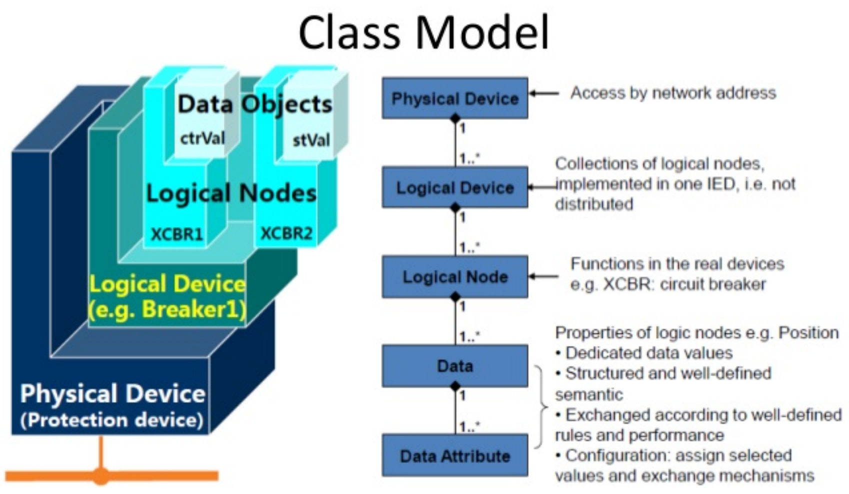

Class Model
Data Objects
ctrVal
stVal
Logical Nodes
XCBR1
XCBR2
Logical Device
(e.g. Breaker1)
Physical Device
(Protection device)
Physical Device
Access by network address
1
1..*
Logical Device
Collections of logical nodes, implemented in one IED, i.e. not distributed
1
1..*
Logical Node
Functions in the real devices e.g. XCBR: circuit breaker
1
1..*
Data
Properties of logic nodes e.g. Position
• Dedicated data values
• Structured and well-defined semantic
• Exchanged according to well-defined rules and performance
• Configuration: assign selected values and exchange mechanisms
1
1..*
Data Attribute

Figure 2.5.: Data Model of the IEC 61850 [29]

Figure 2.5 shows the layers of the model starting from the physical device i.e. battery and down to a single attribute like a floating point number that represents the control value (ctrVal) for the device.

The logical node can be defined as a named grouping of data and associated services that are logically related to a power system function. For example, every device has a logical node called 'LLN0' which holds the basic information of the device and other logical nodes are are 'URCB' for reporting, 'DFAP' for the active power energy service and many others with different capabilities.[19]

2.4.2. Middleware

[E. Madaha]

The Middleware in distributed systems is defined as a software layer that is used to provide a link between separate software applications. In layman's terms it is called a 'software glue', since it works on connecting different software, in other words, glueing them together.

A semantic middleware is an architecture based on matching and mapping the data into an ontological schema defined by a common knowledge base for the distributed system. Simply said, this can be explained as a middleware that uses a defined syntax for the data, that obliges the different software to conform with the model to communicate with each other.[6]

This middleware architecture has the characteristic of inferring semantic knowledge from the data. In order to incorporate this characteristic in the semantic middleware, a service-oriented logic is required, where the services from the middleware are semantically annotated in a stack.[6]

The semantic element allows easy information exchanges between different types of computers, devices or components employing several kinds of operating systems and application languages. Therefore, interoperability of heterogeneous information sources between different middleware components can be enabled by means of this ontology. In addition to that, the ontology provides a knowledge domain where the semantic data reusability becomes plausible.

The protocol used to define the semantic feature is the IEC 61850 standard as it is already a common specification set for the power automation utilities. So, the OS4ES system runs an IEC 61850 stack at the heart of its middleware which converges the complete network of devices and entities to communicate in a single language. [20]

2.5. Behavioural-driven Development

[E. Madaha]

Behavioural Driven Development is a software development process where the requirements of the software are defined in a few tests, which defines the behaviour of the software in easily readable steps, which helps to bridge the gap between management and development teams. This process was originated from Test Driven Development(TDD) and fused with some concepts from the Acceptance Test Driven Development (ATDD)[1].

In this work, the Gherkin language is used to implement the BDD concept, where Gherkin is based off Cucumber, a Ruby-based framework of this concept. The way it works is based on determining the way the software should work, which is put into three layers; feature, scenario and step. The feature describes a feature of the software that is under test, which splits into different scenarios the given feature can operate in and finally the steps taken to create each scenario. The steps are also divided in the form of a precondition written as a "Given", the action as a "When" and the expected outcome as a "Then", this can also be the result that is tested[5].

```
Feature: Some terse yet descriptive text of what is desired

   Textual description of the business value of this feature
   Business rules that govern the scope of the feature
   Any additional information that will make the feature
   easier to understand

  Scenario: Some determinable business situation
      Given some precondition
        And some other precondition
       When some action by the actor
        And some other action
        And yet another action
       Then some testable outcome is achieved
        And something else we can check happens too

  Scenario: A different situation
            ...
```

Here is a sample of the Gherkin language and how it is used.[5]

2.6. OS4ES Lab Test

[E. Madaha]

The OS4ES project has been designed and implemented on separate functional blocks for the DER management, Registry, smart grid interfaces and communication protocols. The finalisation of the project is the integration of these functional blocks through the middleware and ensuring the smooth communication in between. The main work done on this thesis is based on testing the fully integrated OS4ES system on a live CHP and Battery.

The system pictures a trilateral interaction which is under the established Distributed Registry, the Aggregators and the Energy Producers (DERs).

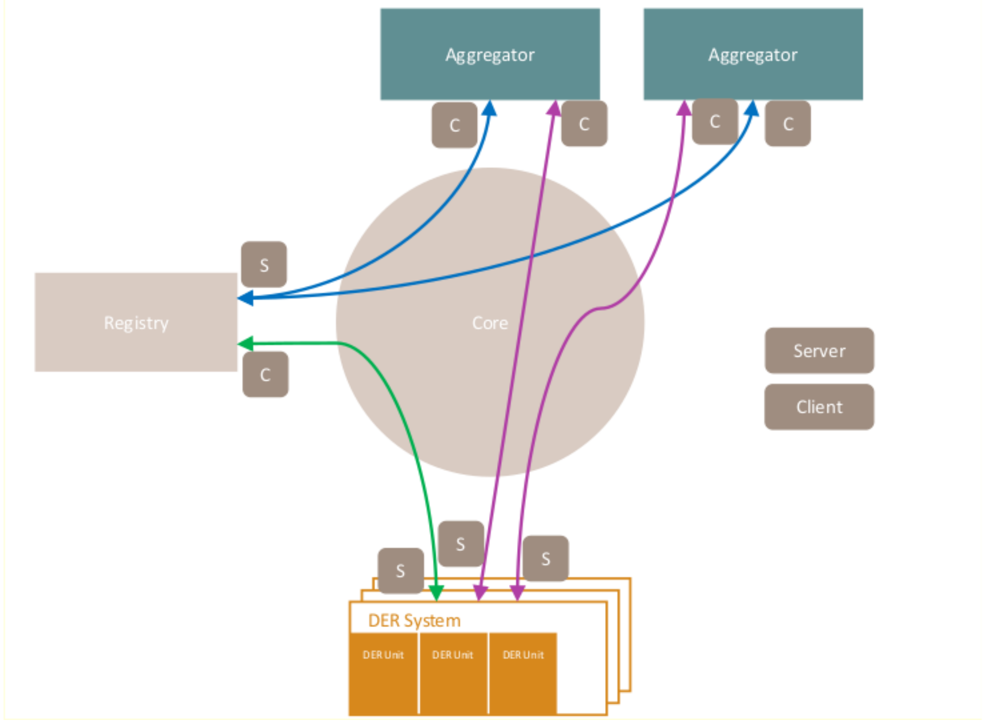


Figure 2.6.: DER-Aggregator-Registry interaction [21]

Figure 2.6 depicts the ideal communication interactions that occurs within the OS4ES system, where there is communication between each party; given that the system can have multiple DER-Systems and multiple aggregators too.

The end to end communication cycle involves the registration of a DER-System on the registry and updating it with an energy service offer. The aggregator then books an energy service from the registry and the aggregator orders the booked service from the relative DER-System.

The thesis serves to prove that the OS4ES project has implemented a system that can achieve the trilateral interaction shown in 2.6. Additionally, another objective is to make sure that the order from the aggregator directly works on a real device (in this case CHP and Battery), generating the exact amount of power ordered. Although in this case the devices used are from a lab, specifically the HUAS[2] Energie Campus in Bergedorf; hence, this test can also be called a lab test, since it was done in a controlled environment.

This Lab test focuses on DER-Systems that support registration of themselves and their energy services including energy service updates to the registry. Such DER systems make use of the extended IEC 61850 functionality that was defined in the semantic model of the project and mapped to new logical nodes. It aims to demonstrate that the DER-Systems can be intelligently controlled in a decentralized manner by the future smart grid actors[3] to form a dynamic Virtual Power Plant(VPP).[22]

The goal of this work is to confirm that the concept of the OS4ES system is feasible; the focus is not the to create a perfectly running system but rather to check if this system could work. Since this work is based on combining different modules developed by different teams (from different countries), the final product is exposed to some incoherences in the implementation as these teams chose their own ways to implement their functional modules.

This thesis is fundamentally information engineering[4] work, in the sense that not much theoretical work was done to accomplish this. Mainly it is based of the software implementation for integrating separate modules and hardware automation and controls of a battery and CHP System, this document aims to cover the technical aspects of the procedures taken to fulfil these tasks.

[2]HUAS: Hamburg University of Applied Sciences
[3]Aggregators are also smart grid actors
[4]this work can also be grouped as technical informatics

3. Requirements Analysis

The main objective of this thesis is to have an end-to-end communication test of the entire system. The end-to-end test will realise the complete process of registering a DER-System and its services to the registry, booking it from the aggregator and finally, delivering as promised from the registered DER-System. To accomplish this, several sub-requirements must be met in compliance with the OS4ES project and the testing platform.

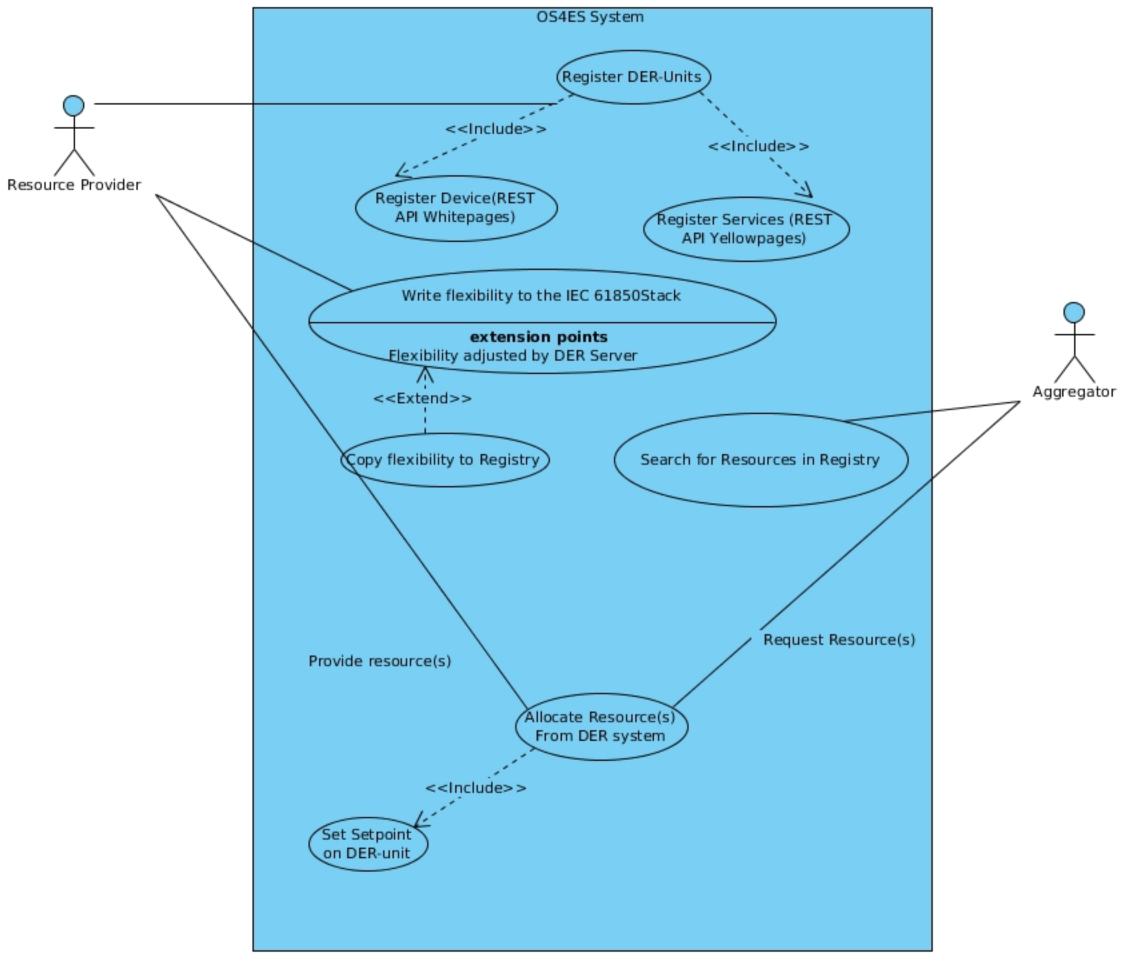

OS4ES System
Register DER-Units
Resource Provider
<<Include>>
<<Include>>
Register Device(REST API Whitepages)
Register Services (REST API Yellowpages)
Write flexibility to the IEC 61850Stack
extension points
Flexibility adjusted by DER Server
Aggregator
<<Extend>>
Copy flexibility to Registry
Search for Resources in Registry
Request Resource(s)
Provide resource(s)
Allocate Resource(s) From DER system
<<Include>>
Set Setpoint on DER-unit

Figure 3.1.: End-to-end communication use case

The use case diagram, Figure 3.1, shows the intended use of the system and the expected behaviour. The diagram gives a general description of how the *Resource Provider* and *Aggregator* actors will interact with the OS4ES. Each actor plays a vital role from a producer and consumer perspective with the OS4ES taking on the role of the "middle-man". In short the OS4ES pools together the resources attached to it (in the Registry) in the form of DER-Systems allowing a central point of aggregation to take place, the Aggregator.

The diagram gives a general description of the different steps involved in the complete process. To offer services from a DER-unit it must first be registered in the registry over the REST API which has been provided. The registration is a two step process: the whitepages and the yellowpages.

The whitepages is responsible for registering the resource provider. This is done by posting a json object containing the device name, the device type, location and capabilities among other information related to the device(DER-unit).

The yellowpages registers the devices service(s) and is used to determine the amount of resource to allocate from a particular device. Together, these two steps allow the device to be visible through the registry by other parts of the system such as the aggregator. The registration provides a mirrored image of the device and services offered by a DER-System.

The second step of the process is for the *Resource Provider* to offer its flexibility in the IE 61850 Stack. A change in the stack should trigger the registry client connection to copy this change to the registry REST API where is can be seeing by all the users of the OS4ES System. Once the service has been registered and its flexibility updated, the *Aggregator* is able to search for this resource and allocate it. The allocation of resource(s) is done by sending a setpoint to the *Resource Provider* (DER-System) which then processes this data and sends it to the Intended DER-unit(s).

The chapter describes in detail the requirements of the OS4ES project and the HUAS test environment. The requirements of the OS4ES serve as a guide to ensure that standards are maintained and the system will be integratable in the end. The requirements of the HUAS should serve as a starting point in constructing the test environment to ensure compatibility with the DER-units. From this point of view, it is possible to group both sets of requirements into functional and non-functional requirements. The sections are designed to give a clear understanding of the priority of each of the task with respect to the requirement group they belong. To illustrate this the requirements are tabularised and the MoSCoW method of prioritising applied.

The MoSCoW method of prioritising is ideal for this development process as the acronym brakes down to: must, should, could and would/wont. This method of prioritising highlights the importance of specific task to the goal of the project. A description of the acronym is given below:

- **M**ust have this requirement to make the project a success. The project is considered to be a failure if this condition is not met.
- **S**hould have this requirement/ functionality. An important part of the project but not essential to the main objective of the project.
- **C**ould have this requirement if time and resources allow.
- **W**ould like to have, **W**on't be this time

The "o"s in the acronym are not of any importance, hence written in lower case letters. The description given above is adjusted to this project. Other descriptions of MoSCoW exist and will vary from use case to use case but in the end describe a similar classification.

3.1. Functional Requirements

[G. Mckoy]

The section gives a short description of the functional requirements of the project in a tabular format and a further detailed description of the individual requirements as deemed necessary. The requirements outlined in this section are guidelines to ensure that the implementation will comply and integrate with the rest of the OS4ES System.

3.1.1. List of Functional Requirements

[G. Mckoy]

The table 3.1 gives a list of the functional requirements:

No.	Requirement	Description	Priority MoSCoW
1	End-to-End System Test	Offer a flexibility from a DER-System –> Book the service from the aggregator based on the flexibility offered -> DER-System responds to the aggregator's request.	M
2	Integration of the DER-System into the middleware	The provided middleware is to be used as the connecting gateway for the DER-System to registry and aggregator.	M

Table 3.1.: List of functional requirements

3.1.2. Detailed Description of the Functional Requirements

[G. Mckoy]

Integration into Middleware

[G. Mckoy]

The middleware consist of several components that work together to establish a system wide communication. These components are the XMPP server, the thrift server, the IEC 61850 Stack and the Java Python Bridge.

XMPP - All forms of communication between the registry, aggregator and the DER-System are done over the Jabber XMPP server in alignment with its protocol. This is accomplished by running an instance of the server in a central location and the different parts of the OS4ES-system connect to it as clients. The server works mainly as a relay collecting messages and forwarding them to their destination. Each client has its own unique username and password which is used to authenticate the client before messages are relayed to the intended recipient.

Thrift Server - The thrift server is used as a base platform to run the IEC 61850 server stack. The server is independent of the all other components and does not rely on any other part of the middleware to be functional.

IEC 61850 Stack - The IEC 61850 Stack runs on the thrift server and requires an instance of the server to be running before it can be instantiated. The stack works by loading in a configuration file that describes the devices to be represented and how to access these different devices.

Java Python Bridge - The Java Python Bridge works by converting data to and from string that is read by either Python or Java and casted to one of their respective primitive types. These strings are of a specific structure that allows the distinction between arrays, single objects, float, int and strings. To communicate data from Java to Python the bridge encodes the string in alignment with the IEC 61850 standard. The string is then parsed in Python and the command executed or data processed depending on the invoked method. Data received on the Java side of the bridge is decoded from the IEC 61850 standard into Java primitives and objects for future use. The bridge has it limitations as it is not able to fully represent both languages to communicate with each other. One such limitation is the ability to store mixed objects in Python which Java does not support being a strongly typed language.

Integration - The DER-System must make use of the Java Python Bridge to connect to the rest of the system. The implementation must comply with the specifications of the Java

Python Bridge in order transfer data accurately. Should additional functionalities be needed this may be added and deployed as version 4.

End-to-End Communication

[G. Mckoy]

For the end-to-end communication test, a dummy aggregator and registry client should be implemented to connect to the simulated DER-units via the DER-System over the middleware. The dummy registry must be able to read the changes made in the DER-Server and update it in the registry. As a starting point the registration of the DER-System and its services can be achieved over the REST API which is already implemented. Later this can be improved to allow registration to be done automatically from the registry by monitoring when a new device gets added to the IEC 61850 Stack on the DER-System. The aggregator client systems must be able to read the services being offered by a DER-System from the registry. Upon reading the intended data from the registry, the aggregator should use this data to make a request from the DER-System in reference. Once the request is received by the DER-System an acknowledgement should be sent to the aggregator or indicated in the IEC 61850 Stack.

3.2. Non-Functional Requirements

[G. Mckoy]

The requirements outlined in this section serve as the non-functional requirements of this thesis. A tabularised short description is given in subsection 3.2.1 and more detailed descriptions in subsection 3.2.2. The detailed description encompasses the reasons and constrains behind these requirements.

3.2.1. List of Non-Functional Requirements

[G. Mckoy]

The table 3.2 gives a list of the requirements:

No.	Requirement	Description	Priority MoSCoW
1	Java 7 and above	Keep inline with the rest of the implementation by the other partners.	M
2	Simulation Devices	The test environment should make use of simulation devices that will behave similar to the real system.	S
3	Testing of the Java Python Bridge version 2 and 3	The Java Python Bridge is the primary access point of the DER-System to the OS4ES project. All communication must adapt to using the bridge with the exception of the initial registration of a DER device and its services.	M
4	Setting up a test environment	A test environment is needed to assess the current status of the project and determine the necessary modifications to be done.	M
5	MQTT Compatibility	The implementation must be able to communicate over MQTT.	M
6	Record data to the influx database	Store the flexibility and forecast.	S
7	Smart Gateway	The Smart Gateway should act as the processing unit for the DER-System monitoring the use of resources attached, distributing the schedule and setpoints received among other functionalities.	S
8	Transparency	The simulated devices should not be distinguishable from the real system to the rest of the OS4ES system.	S
9	Scalability	The project is expected to support multiple DER-Systems on a large scale. However this implementation primary objective the feasibility of an end to end communication.	S
10	Security	Security measures should be taken to sucre the system as much as possible regardless of it operating within a secure network.	S
11	Robustness	The system should be developed and tested against multiple test case. The test cases should cover but not be limited to: Handling more than one device, disconnecting a device and passing incorrect data to the device.	S

Table 3.2.: List of non-functional requirements

3.2.2. Detailed Description the Non-Functional Requirements

[G. Mckoy]

Figure 3.2 describes the desired setup of the HUAS lab test. The figure focuses on the DER-System to be implemented whilst highlighting how the DER-System will communicate with rest of the project namely the registry and aggregator.

Smart Gateway

[G. Mckoy]

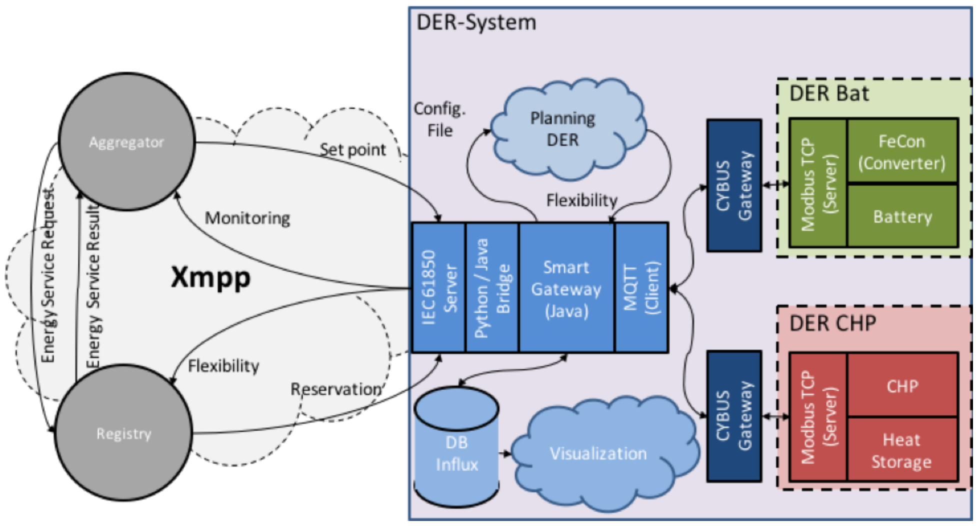


Figure 3.2.: DER-System and its communication over the middleware

A Smart Gateway should be developed that works as the main processing unit of the DER-System. The Smart Gateway should be capable of computing the flexibility of each DER device attached to it and offer this flexibility as an individual setpoint or as a schedule (ideally in the form of every 15 minutes for the next 24 hour).The Smart Gateway shall make use of the provided communication interface, the Java Python Bridge to interact with the rest of the system.
Below are the hardware and software specification for this requirement:

1. *Hardware Requirements:*
   - The Smart Gateways shall be installed on a desktop computer capable of running the thrift server and handling multiple DER devices simultaneously

2. *Software Requirements:*
    - Operating system should be Linux based (debian based preferable).
    - The provided middleware must be used to communicate with the rest of the system.
    - The code must be developed in Java 7 or above.

MQTT Compatibility

[G. Mckoy]

The real DER-units are connected to a server that is based on MQTT protocol. To communicate with these DER-units in the network, a MQTT client is needed. The application being used to manage this service is Cybus. The implementation must be able to communicate with the MQTT broker running via Cybus.

Simulation of DER Devices

[G. Mckoy]

The simulation models of a CHP, PV and a Battery should be implemented on two Raspberry Pis. The CHP should be on one of these devices and the Battery and PV on the second. These devices should then be connected to the Smart Gateway over the modbus communication interface to complete the DER-System.

The goal of a simulated devices is to be able to remove the Pis and attach a real DER-System without the rest of the OS4ES-system being able to distinguish the difference between the two. To accomplish this the simulated devices must behave the same as the real devices with respect to the way they receive and returns data. The simulation models can be a simple Python script that uses Modbus interface to communicate. Later the Matlab provided simulation models are to be converted into Python and integrated with the communication script created previously.

3.2.3. Lab Test

[G. Mckoy]

A pre-lab test must be done to verify the functionality of the the physical devices before connecting them to the implementation of the end-to-end communication. This test should include but not limited to;

- Powering on and off the DER-units:
  - This control must be done over the mqtt protocol.
  - A secure sub-network must be used that will interact with the cybus network.
- Setting a setpoint or schedule:
  - The systems do not have a storage device to receive a schedule and as such and intermediate application must be used to perform a storage behaviour which will distribute the schedule .
  - Observation of the reaction time to each setpoint should be recorded to aid in the implementation.

4. Test Environment and Design

The process of developing the Smart Gateway and setting up such a local test and development environment consists of two main parts: the hardware and the software configuration. These steps are described below in details under their respective sections. The sections describe the behaviour and the purpose of various devices and the communication protocol used. It offers an in-depth look at the specific parts of the source code and the technologies used. Definitions of unfamiliar concepts and technologies are given where deemed necessary and references to further reading for a more detailed description.

4.1. Hardware Communication and Networking

Figure 4.1 shows the desired setup of the local network created in order to communicate between the two computers and the Raspberry Pis. The figure highlights the different communication protocols between each device. For the purpose of clarification, the computers are described as PC1 and PC2 and the Pis as Pi1 and Pi2 in alignment with Figure 4.1.

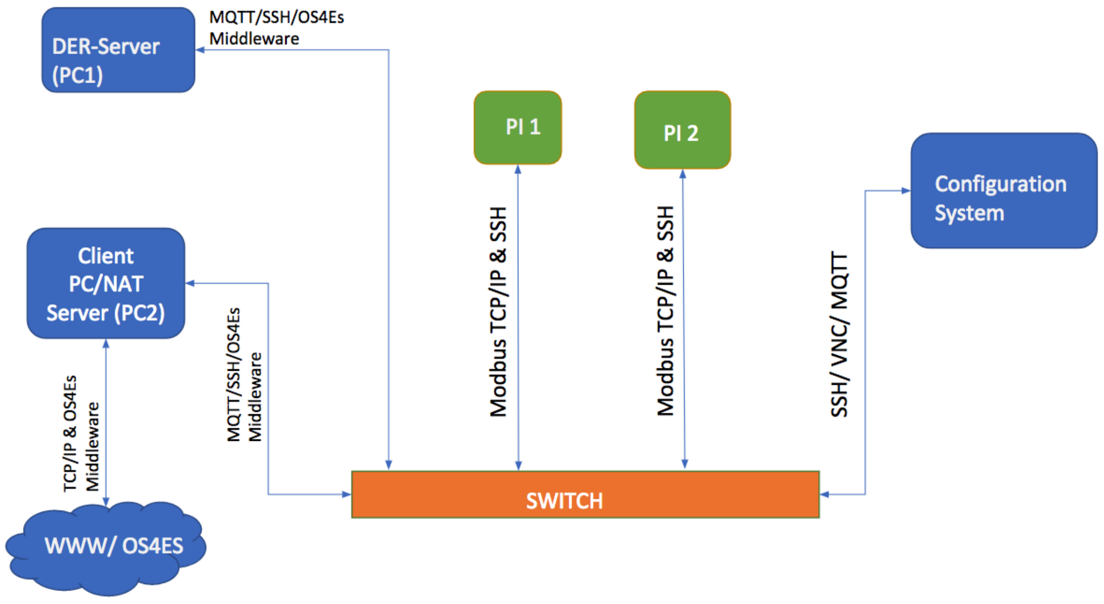


Figure 4.1.: DER-System network connections

Below is a list of the hardware devices and components required in order to realise the setup depicted.

- 2 Computers.
- 2 Raspberry Pis.
- 1 PCI Express Ethernet Card.
- 1 Switch (minimum 6 sockets).

4.2. Computer Hardware Configuration

4.2.1. PC1 Hardware Configuration

PC1 serves as the DER-System and is responsible for communication with the rest of the OS4Es System. PC1 runs the thrift server, the Smart Gateway and Node-RED. The computer is a slightly above the standard configuration of a regular home computer with a Xeon processor and 24GB of RAM to ensure adequate processing power.

4.2.2. PC2 Hardware Configuration

PC2 serves as a network address translation server (NAT server) and a client to connect to the DER-Server during testing. PC2 has a Xeon processor and 8GB of RAM with an additional Ethernet card that allows it to work as a NAT server. The purpose of the NAT server is to connect to the DER-Systems remotely. This is needed as the physical devices are in a different location from where the implementation takes place. Necessary security checks were put in place to secure the network from the outside world. This was achieved by creating the DER-System local network as a sub-network of the university's. By doing this, the university's security was adapted and accessing the sub-network was only possible via VPN.

4.3. Computer Software Configuration

4.3.1. PC1 Software Configuration

PC1 is configured with the Linux Mint 2017 operating system as its primary operating system. It is important to point out that just about any Linux Debian based operating system could be used for this purpose. Below is a list of software to be installed on PC1:

- Java Python Bridge and its dependencies
- Node-RED
- Hat-open distribution
- Thrift Server
- Paho-eclipse

Purpose of the Selected Software

Java Python Bridge - The Java Python Bridge is to be used to communicate between the DER-System and the rest of the system as part of the middleware. This piece of software is given as a part of the starting point of the thesis. For PC1 the server implementation is installed as this computer operates as the server for the DER-System. The Java Python Bridge has two versions, version two and version three. Both are to be installed and tested to ensure the functionalities are as described by the specifications. A simple Java program was created to communicate with the server from the client and another that writes data to the server via the Java Python Bridge.

Hat-open - The Hat-open distribution is used along with the Java Python Bridge in order to encode and decode the data being transferred between Java and Python via the IEC 61850 stack running on the thrift server.

Thrift Server - The Thrift Server works as the foundation on which the IEC 61850 server stack to run the DER-System.

Node-RED - Node-RED is a web-based application built on Node.js and uses JavaScript as it programming language. One of the key features of Node-RED is that it has excellent support for MQTT. More on MQTT can be found under Definition of Technologies Used subsection 4.4.5 of the thesis. The application takes advantage of the vast capabilities of the Node.js platform and the flexibility of JavaScript.

Paho-eclipse - The Paho-eclipse package is be used to communicate between the Smart Gateway and Node-RED. The implementation shall create one instance of the client that will serve as the only means of interacting with the DER-units. The package provides MQTT functionalities to subscribe, publish and monitor activities on an MQTT broker.

4.3.2. PC2 Software Configuration

PC2 is to be configured with the Ubuntu 14.1 operating system. This operating system was chosen to keep inline with the tutorials on how to create a NAT server[11]. In theory, just about any Debian based OS could have been applied to the tutorial used. However, to avoid time wasting the tutorial was followed step by step without modifications.

Operating as the NAT server PC2 provides the Dynamic Host Configuration Protocol (DHCP) within the local sub-network and assigns an address to all connected devices. Once this is done IP addresses are reserved for all devices that are used: PC1 and both Raspberry Pis. The addresses are then tested to ensure the correct addresses were assigned to each device.

The Java Python Bridge configurations are similar to that of PC1 with the main difference being that the client version is installed. The client version does not rely on the thrift server to be explicitly started in order to establish communication with the DER-Server.

4.3.3. Problems Faced During Software Configuration

Java Python Bridge

One of the most challenging problems faced was the installation of the Java Python Bridge mainly due to poor documentation. A great deal of time was spent trying to understand the code. The poor structure and lack of documentation resulted in several dead ends trying to find and understand the problem. The bridge has two versions, version two and version three as stated above. Version two had little complication while setting up and was running in a matter of a few minutes once all the necessary dependencies were satisfied. By close observation, it was evident that version two was lacking some essential features needed to communicate with the DER-System and as such version three was going to be the primary candidate.

Version three by design is far superior to version two with the ability to send and receive more complex data structures over the bridge. It provides a detailed description of a PV and battery as Java DER object. Additionally to this, the source code definition of a few of the logical nodes was provided. Though this was not aligned with the specification that the PV and battery are one system, it served as a useful starting point to build on. A major setback in configuring version three was one of the essential files being outdated. The outdated file led to the bridge not functioning as expected and a constant state of debugging to find the cause of the error. This error was narrowed down and resolved by contacting the respective partners to retrieve the updated file. Once this was done, it was a matter of minutes to have version three up and running.

4.4. Communication Between Smart Gateway and DER Devices

4.4.1. Node-RED

Node-RED is used for establishing communication with the physical devices and the test Raspberry Pis. Node-RED provides an easy and straightforward way to network different devices using the Internet of Things(IoT). It was chosen for the test system because of its similarity to Cybus which is used on the live system. With Node-RED installed the challenges of communication were drastically reduced and allowed for resources to be diverted elsewhere. To setup the communication between a device and Node-RED the following steps can be followed:

1. Create a flow.
2. Select the communication protocol you wish to use from the palette menu. If the palette is not available then search the Node-RED store.
3. Configure the communication settings from the GUI provided or if needed adjust the code manually.
4. Save and deploy.

4.4.2. Integration of Node-RED

The implementation relies on Node-RED to remove the complexity of setting up the various communication protocols between the Smart Gateway, database and the DER-units. The use of Node-RED in this setup is purely for testing purposes. The real system is configured with Cybus which is a more advanced and secure system with Node-RED running at its core. Node-RED is used primarily to communicate with the Pis over Modbus and to connect to the influx database. Other secondary uses are establishing a bridge between the Cybus remote connection and the simulated system. To interact with the Cybus system, a direct connection is possible through Java using the Paho-eclipse library. However, the Node-RED approach was chosen because of its flexibility.

To better structure the operations to be performed, similar activities are grouped together as flows. The flows are used in such a way that they isolate individual tasks and connect to other flows where needed. Below is a basic list of flows that would form a good Node-RED structure:

- Modbus communication
- Influx database
- Real DER-Systems

These fundamental flows will ensure that different parts of the system do not interfere with each other.

4.4.3. Modbus-tk

The Modbus-tk Python library is be used to create the Modbus server to which the Node-RED client connects. The library provides the functionality of creating a server or multiple clients based on the needs. For the purpose of this thesis, the server functionality is of interest as the simulated devices will not connect to each other directly over Modbus.

4.4.4. Paho-eclipse

The Eclipse Paho project is an open-source client implementation of MQTT and MQTT-SN messaging protocol designed to integrate and interact with the Internet of Things. The package is Java based and runs on Java 7 and above.

4.4.5. Definition of Technologies Used

Node-RED

Node-RED[1] is a programming tool used to establish communication between hardware devices, APIs and online services. It is a tool, which has a browser-based user interface that

[1] Node-RED was created in 2013 as a side-project by Nick O'Leary and Dave Conway-Jones of IBM's Emerging Technology Services group [17].

makes it easy to use on all browser supported platforms. It uses nodes to represent different functional behaviour which can be grouped together as flows . The platform has a wide range of palettes that can be used to configure various flows which can be deployed to its runtime in a single-click, more on Node-RED can be found at [17, p. 12].

MQTT

MQTT [2] is a Client Server publishing/subscription message transfer protocol. The simple, light-weight, open-source protocol is designed to promote a trivial implementation. These attributes make it ideal for deployment on devices with limited resources such as Machine to Machine (M2M) communication or in conjunction with Internet of Things contexts where network bandwidth and code footprint play a vital role. The protocol runs over TCP/IP, or over any other network protocols that support ordered, lossless, bidirectional connections [3, p. 81]. Its features include:

1. The ability to publish/subscribe message pattern providing a one-to-many message distribution in addition to decoupling of applications.
2. Transportation of messages that are agnostic to the content of the payload.
3. Three qualities of service for message delivery:
    a) "At most once" - messages are delivered based on the best efforts of the operating environment. Message loss may occur.
    b) "At least once" - messages are guaranteed to arrive. However, duplicates may occur.
    c) "Exactly once" - exactly one message is guaranteed to arrive.
4. To reduce network traffic a small transfer protocol is used to minimized overhead.
5. A notification system that alerts interested parties when an out of the ordinary disconnection occurs.

More can be read on the protocol and standards used from the MQTT-v3 manual [3].

[2] MQTT was invented by Dr Andy Stanford-Clark of IBM, and Arlen Nipper of Arcom (now Eurotech), in 1999 [3].

Modbus

Modbus is a free serial protocol for communicating with and between programmable logic controllers (PLCs). [3] It is one of the most widely used means of connecting various industrial devices and allows bidirectional communication among many different devices simultaneously on the same network. The protocol uses TCP/IP, RTU/IP or UDP to transfer data between devices. Modbus supports many different ways of transferring data such as coil, discrete input, input register and holding registers. Of interest for the implementation are the input and holding registers, these are readable and writeable 16-bit registers. They are be used to store the configuration and measurement values. More on Modbus can be found at [28] or just about anywhere on the internet since the protocol is so widely used.

4.5. Raspberry Pis

The Pis are used to represent the physical DER-Systems, in this case, a CHP, PV system and battery. One device simulates the behaviour of a CHP and the other a PV and battery. A simple program is to be written that establishes communication over Modbus using Python. Once this communication is completed the models created in Matlab are to be exported to the Pis to better represent the real DER-units.

The Pis are configured with the Raspbian Jessie Lite operating system. The lite version of the OS was chosen because a user interface is not needed since the communication protocol to be used is SSH [4].

4.5.1. Raspberry Pis

The Raspberry Pis are both of the type B models with 512MB RAM and a 4GB SD card to host the OS. In addition, a second SD card with a capacity of 8GB of storage is used to form a second implementation which simulates the Matlab versions of the DER-unit. Model B was chosen because it provides sufficient processing power to simulate multiple DER-units at the same time. Other mini computers meeting the minimum requirements of having an Ethernet port and the ability to run Python 2.7 and above can also be used.

[3] The protocol was developed in 1979 by Medicon and is now maintained and managed by Schneider Electric.

[4] The **S**ecure **SH**ell network protocol provides a secure means of communicating from one computer to the next. All data transfered over this protocal is encrypted.

4.5.2. Raspberry Pi Communication

Communication is done using Modbus over TCP/IP to open a port which is used to communicate with Node-RED. The Python library of choice is the Modbus-tk library. It provides the necessary methods for creating a Modbus server to send and receive data. Each Raspberry Pie device is configured with a static IP address so that they can be reached from Node-RED without the hassle of pinging each address to find a specific device.

4.5.3. Raspberry Pi Simulation Models

The simulated DER-units are written in Python 3.5 to keep in line with the version of Python being used in the rest of the project. The DER-Systems are written in the form of simple scripts that will run on start-up to ensure that they are always online after reboot. The second version of the simulation will use the exported Matlab versions which require the Raspberry Pis be configured according to the specification on Matlab's homepage. Once the simulation is deployed on the Raspberry Pi, the communication module should be integrated and tested.

4.6. Design

Based on the assessment of the testbed environment, it was determined that the provided implementation of version two of the Java Python Bridge was not sufficient to meet the demands of the Requirements Analysis. Version three of the implementation shall be extended to better describe a DER-System and extend its communication functionality.

4.6.1. PV Model Description

The Smart Gateway shall model the DER-Systems as objects while meeting the minimum specification outlines in the deliverable 6.6.2 of the OS4ES. Figure 4.2 shows a modified version of the class diagram in the deliverable 6.6.2. The class diagram shows the representation of a PV DER-unit with its logical nodes as properties. It is important to point out that a PV may have additional logical nodes or components based on its description. The representation below is a reflection of the implementation to be used in conducting the HUAS lab test. The modifications allow for the DER-units to be functional and object-oriented. From a programming perspective, it seemed that some encapsulation was missing and as such some of the attributes of the classes were changed from public to protected or private where deemed necessary. Additional functionalities were given to not just the PV but all future DER-Systems in the form of an interface DER-System Methods.

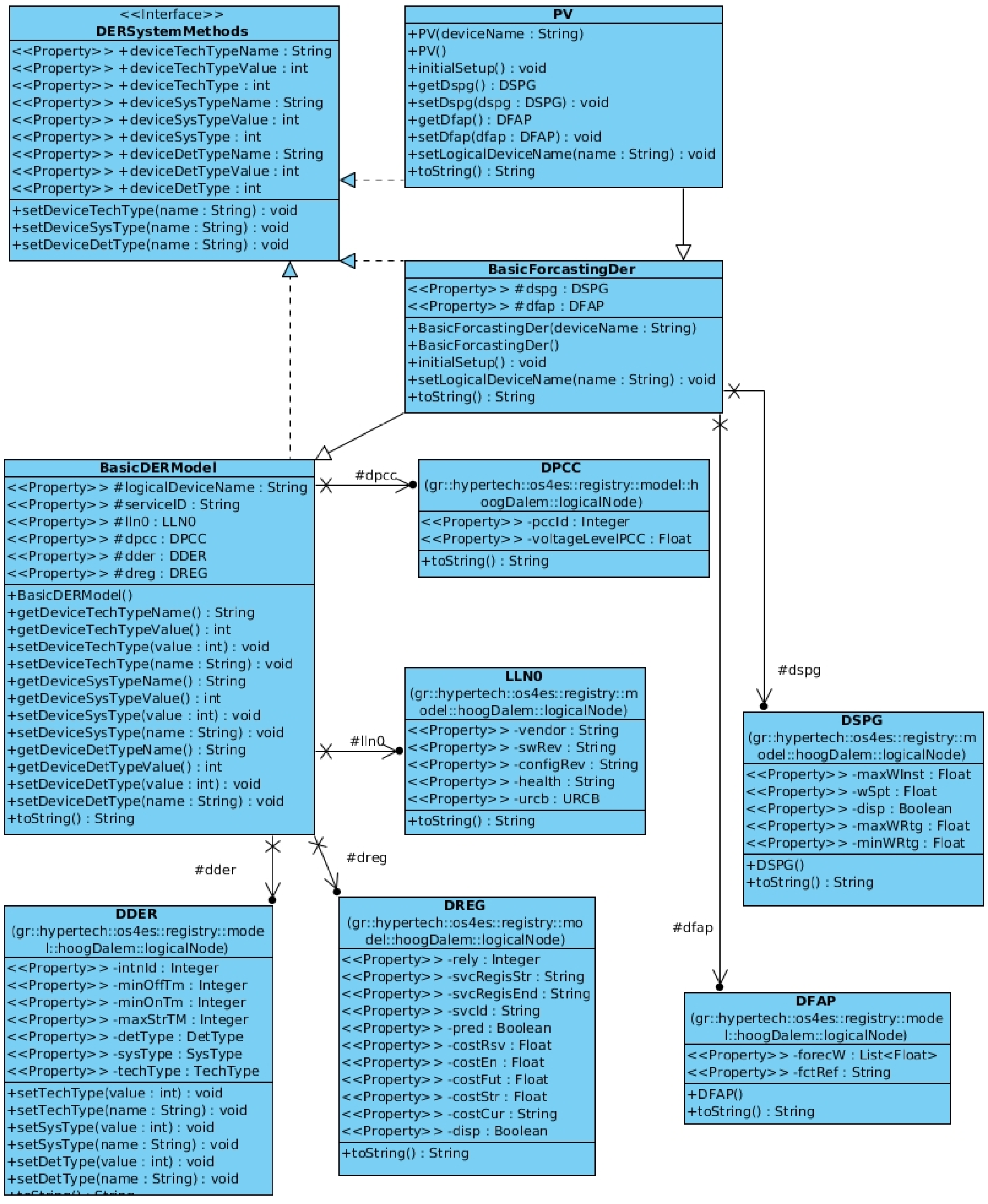

<<Interface>>
DERSystemMethods
<<Property>> +deviceTechTypeName : String
<<Property>> +deviceTechTypeValue : int
<<Property>> +deviceTechType : int
<<Property>> +deviceSysTypeName : String
<<Property>> +deviceSysTypeValue : int
<<Property>> +deviceSysType : int
<<Property>> +deviceDetTypeName : String
<<Property>> +deviceDetTypeValue : int
<<Property>> +deviceDetType : int
+setDeviceTechType(name : String) : void
+setDeviceSysType(name : String) : void
+setDeviceDetType(name : String) : void
PV
+PV(deviceName : String)
+PV()
+initialSetup() : void
+getDspg() : DSPG
+setDspg(dspg : DSPG) : void
+getDfap() : DFAP
+setDfap(dfap : DFAP) : void
+setLogicalDeviceName(name : String) : void
+toString() : String
BasicForcastingDer
<<Property>> #dspg : DSPG
<<Property>> #dfap : DFAP
+BasicForcastingDer(deviceName : String)
+BasicForcastingDer()
+initialSetup() : void
+setLogicalDeviceName(name : String) : void
+toString() : String
BasicDERModel
<<Property>> #logicalDeviceName : String
<<Property>> #serviceID : String
<<Property>> #lln0 : LLN0
<<Property>> #dpcc : DPCC
<<Property>> #dder : DDER
<<Property>> #dreg : DREG
+BasicDERModel()
+getDeviceTechTypeName() : String
+getDeviceTechTypeValue() : int
+setDeviceTechType(value : int) : void
+setDeviceTechType(name : String) : void
+getDeviceSysTypeName() : String
+getDeviceSysTypeValue() : int
+setDeviceSysType(value : int) : void
+setDeviceSysType(name : String) : void
+getDeviceDetTypeName() : String
+getDeviceDetTypeValue() : int
+setDeviceDetType(value : int) : void
+setDeviceDetType(name : String) : void
+toString() : String
#dpcc
DPCC
(gr::hypertech::os4es::registry::model::hoogDalem::logicalNode)
<<Property>> -pccId : Integer
<<Property>> -voltageLevelPCC : Float
+toString() : String
#lln0
LLN0
(gr::hypertech::os4es::registry::model::hoogDalem::logicalNode)
<<Property>> -vendor : String
<<Property>> -swRev : String
<<Property>> -configRev : String
<<Property>> -health : String
<<Property>> -urcb : URCB
+toString() : String
#dspg
DSPG
(gr::hypertech::os4es::registry::model::hoogDalem::logicalNode)
<<Property>> -maxWInst : Float
<<Property>> -wSpt : Float
<<Property>> -disp : Boolean
<<Property>> -maxWRtg : Float
<<Property>> -minWRtg : Float
+DSPG()
+toString() : String
#dder
#dreg
DDER
(gr::hypertech::os4es::registry::model::hoogDalem::logicalNode)
<<Property>> -intnId : Integer
<<Property>> -minOffTm : Integer
<<Property>> -minOnTm : Integer
<<Property>> -maxStrTM : Integer
<<Property>> -detType : DetType
<<Property>> -sysType : SysType
<<Property>> -techType : TechType
+setTechType(value : int) : void
+setTechType(name : String) : void
+setSysType(value : int) : void
+setSysType(name : String) : void
+setDetType(value : int) : void
+setDetType(name : String) : void
DREG
(gr::hypertech::os4es::registry::model::hoogDalem::logicalNode)
<<Property>> -rely : Integer
<<Property>> -svcRegisStr : String
<<Property>> -svcRegisEnd : String
<<Property>> -svcId : String
<<Property>> -pred : Boolean
<<Property>> -costRsv : Float
<<Property>> -costEn : Float
<<Property>> -costFut : Float
<<Property>> -costStr : Float
<<Property>> -costCur : String
<<Property>> -disp : Boolean
+toString() : String
#dfap
DFAP
(gr::hypertech::os4es::registry::model::hoogDalem::logicalNode)
<<Property>> -forecW : List<Float>
<<Property>> -fctRef : String
+DFAP()
+toString() : String

Figure 4.2.: Class diagram battery modification

The DER-System methods interface forces the users to define the behaviour of a DER-System while keeping the same method names. This adjustment allows us to retrieve or just update information within a system regardless of the inner mechanism. The abstract class BasicNodeInfo will extend the capabilities of a logical node. Capabilities such as a logical node's functional constraints or path as it may vary from node to node.

4.6.2. Test Description

As with any system testing is a necessary process in the implementation. For this, it was decided to use a direct approach and the Gherkin test description. The test cases are designed for individual components of the system and finally a full system test to ensure the components work together as expected. A more detailed description of this can be found in the subsection 4.6.2 of this chapter.

End-to-End Communication Test

The sequence diagram Figure 4.3 describes the general behaviour that is expected of the end-to-end communication.

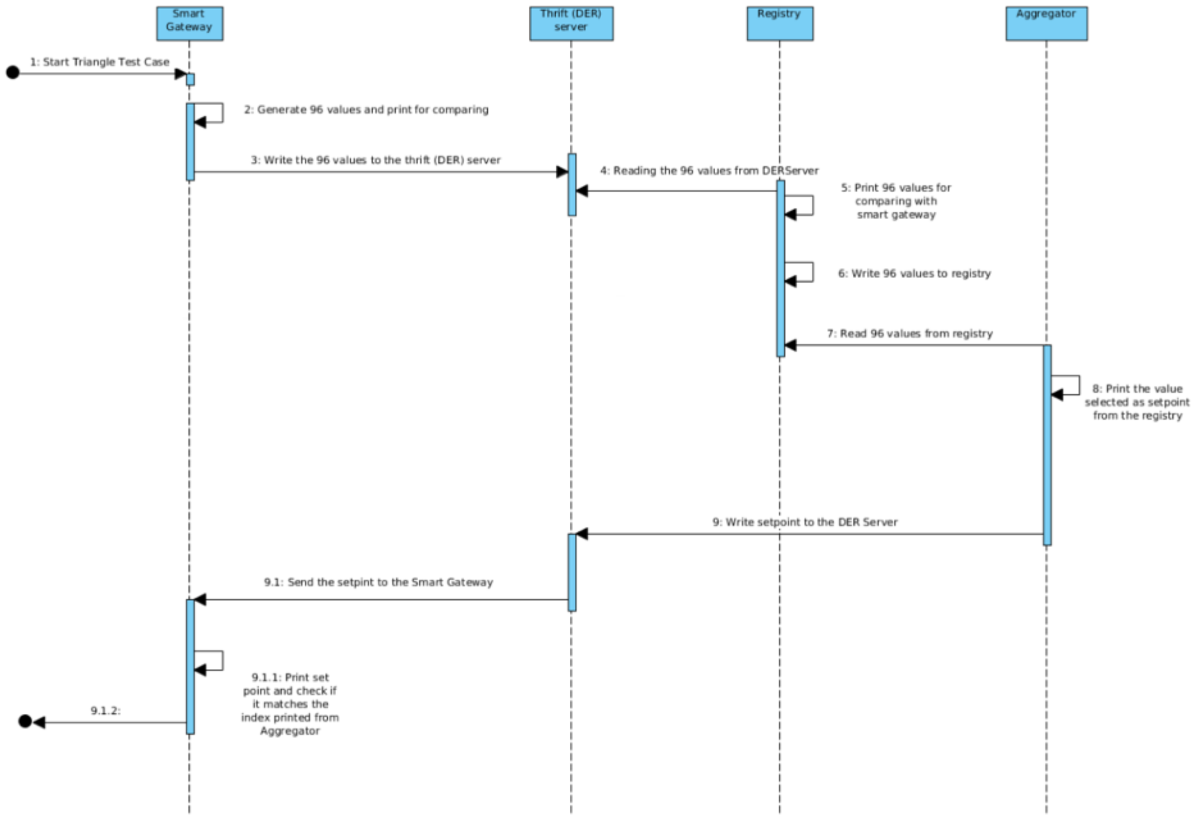

Smart Gateway
Thrift (DER) server
Registry
Aggregator
1: Start Triangle Test Case
2: Generate 96 values and print for comparing
3: Write the 96 values to the thrift (DER) server
4: Reading the 96 values from DERServer
5: Print 96 values for comparing with smart gateway
6: Write 96 values to registry
7: Read 96 values from registry
8: Print the value selected as setpoint from the registry
9: Write setpoint to the DER Server
9.1: Send the setpint to the Smart Gateway
9.1.1: Print set point and check if it matches the index printed from Aggregator
9.1.2:

Figure 4.3.: Sequence diagram of end-to-end communication test.

The sequence diagram shows the setup of the end-to-end communication between the DER-System, Registry and the Aggregator. Points to note: In the diagram above the Smart Gateway and the DER-Server are on one system but are separated here for a more detailed description of the internal processes of the test case. The flow starts by generating the availability represented by the 96 values. The values are then printed out to be used later for verification. The 96 values are then written to the DER-Server (via JPB–>IEC 61850) where the registry reads them (via IEC 61850—>JPB) and makes a printout for comparison and writes them to the registry's database. Once written to the database the aggregator reads all 96 values (via the REST API) and selects what it would like to use as setpoints. After selecting the desired values, the values are then written directly to the DER-Server (IEC 61850—>JPB) as a setpoint to represent the logical device's availability. Finally, the Smart Gateway is notified of the change in the DER-Server and a check is made if the value(s) set was sent by the aggregator.

Gherkin Test

Part A:
Registry schedule update from the DER-Server The test case description from the Behavioural description document already satisfies the current setup with the new Java Python Bridge test made. Only the addition of a start time is left for this case.

Schedule Update
Purpose: Process a schedule update from the DER IEC 61850 server. Corresponding energy service: ActivePower.

Scenario

1. Given the registry contains an energy service of type ActivePower
2. There is an IEC 61850 DER-Server running.
3. The registry is subscribed to the DER-Server with a Logical Device name as the service ID of (1)
4. When the registry requests for a schedule update
5. Then it is expected that the setpoints of (1) are updated to be the same as the forecasted Watt values of (2).

Exception a : No Schedule available

4a. When the registry requests for a schedule update which is empty

5a. Then an empty schedule is received

Part B:
OAA update schedule from Registry Purpose: Read a schedule from the registry on the Specified Device/Energy Service Corresponding energy service: ActivePower.
Scenario

1. Given the registry contains an energy service of type ActivePower and the updated schedule from the relative DER device.
2. The Aggregator is subscribed to the registry with a Logical Device name as the service ID of (1)
3. When the Aggregator requests for the schedule from (1)

4. Then I expect that the schedule of (2) is successfully updated to be the same as the registry values of (1).

Exception a : No Schedule available

4a. Then an empty list is received and the update should fail.

Part C:
OAA set setpoint to DER
Behavioral Description:
Purpose: Send over the setpoints to the DER-System on the registered energy service Corresponding energy service: ActivePower.
Scenario

1. Given the OAA knows the logical device, Energy Service for the relative DER-System
2. The DER-Server is running with the same Logical Device name and energy service
3. The OAA is subscribed to the DER-Server
4. When the OAA publishes a setpoint
5. Then I expect that the setpoint from (3) to be updated in the (2) DER-Server on the respective logical device and address
6. Finally the Smart Gateway is updated with a new setpoint and runs the generator (CHP or PV) according to the setpoint.

Exception a : Wrong Numeric Type

4a. When the OAA publishes a setpoint that is not the right type

5a. Then the publishing fails and server does not do anything

6a. Smart Gateway receives no new command

Exception b : setpoint out of operable range

4b. When the OAA publishes a setpoint that is not in the operating range of the device

5b. Then the setpoint is updated in the DER-Server

6b. Smart Gateway refuses to set the value to the device.

5. Implementation

The implementation work of this thesis is based on accomplishing a full test environment for the OS4ES system. To realise this, an internal DER-System is needed along with two external components namely the Aggregator and the Registry. The external components do not realise the full potential of their respective requirements. Instead, these components are constructed to meet the minimum requirements for conducting the end-to-end communication test. In addition, a few adjustments to the middleware are needed to aid in improving communication with the components that are to be attached to the DER-System.

During the implementation process alternative work was done which could not quite fit into the final implementation. This work is mentioned briefly in this chapter and the bulk of it discussed in the Appendix.

5.1. Simulated DER Devices

[G. Mckoy]

The implementation of a DER device can be divided into two primary parts: The communication protocol and the system behaviour. The communication module describes how the communication protocol between the DER-System and the DER-units is established. The system behaviour covers the detailed implementation of each of the devices simulated.

5.1.1. Communication Module

[G. Mckoy]

The communication between the DER-System and the simulated DER-units (Battery, PV and CHP) is achieved over Modbus as to the specification of the Requirements Analysis. The Modbus-tk library is used on the Pis to connect to Node-RED. The library works by creating a Modbus server for Node-RED to connect to in order to send and retrieve data. The hold register functionality of the protocol is used for this purpose, see the Modbus section of Chapter 4 of this thesis for a description of the registers used by the protocol.

Data is transferred in the form of three bytes or more depending on the functionality. The first byte is the control byte which tells the system in which mode to operate in and what to type of data to expect. The second and third bytes transfers a value such that; the second byte represents the upper 8 bits of a short int and the third byte represents the lower 8 bits. Additional information can be sent if both the sender and the receiver know the number of bytes being transmitted. Once received, the values are joined together by means of shifting and bitwise operations on both sides of the communication channel. The JavaScript example of joining two values together is shown in listing 5.1.

```
 1  const context = this.context.global;
 2
 3  const batt = ((msg.payload[1] << 8) | msg.payload[2]);
 4  const conv = ((msg.payload[3] << 8) | msg.payload[4]);
 5
 6  msg.payload = batt + conv;
 7  msg.bat = batt;
 8  msg.conv = conv;
 9
10  return msg;
```

Listing 5.1: Code snippet of the function that receives the output of the converter and battery Modbus output

Listing 5.1 shows data received in the form of an array, payload,from the Modbus communication in Node-RED. The conversion process is for two numbers. The first number is stored in indexes 1 and 2 of the array while the second number is stored in indexes 3 and 4. This is seen in the listing at line numbers 3 and 4 respectively. The upper eight bits of the number are shifted to the left and "OR" with the second number to create the 16 bit value. The reverse of this process is performed from the Python side when sending data to the Modbus holding register.

5.1.2. System Behaviour

[G. Mckoy & E. Madaha]

CHP

[E. Madaha]

OperationThe way the CHP works is more as a part of a larger system rather than just a simple generator; since the CHP is also involved with the heating of the building. This makes the CHP directly dependent on the heat storage, mainly when the water in the heat storage goes beyond critical temperatures. When the heat storage is at a critically low temperature, it triggers the CHP to turn on. Once on the CPH Will run until the heat storage has reached a critically high temperature where it then triggers the CHP to turn off.

The heat storage avoids staying in a critical low temperature to ensure that the building can always be heated. Hence, it automates the running of the CHP to prevent a situation where the heat demand is high and the heating system needs to wait for the CHP to turn on. Adversely the heat storage also avoids overheating to keep some parts of the heating infrastructure from being damaged or malfunctioning as that would compromise the heating system as a whole.

To simply simulate the normal operation of this device, it is required to study the heat demand and create a similar type of characteristic from this data, this is the focus of the work done in the heat demand aggregation 5.3.2. As concluded in this section, the module built for running the forecast was not ready at that time, so instead a simple heat simulation with a linear behaviour was used, following the average change in the heat storage's temperature throughout the day.

The code for the simulation at the end is just a linearly growing heat demand where once the set critical low temperature is reached the simulated CHP is changed to an 'ON' state. Once the CHP is on the temperature starts to rise linearly by the estimated average based on the historical behaviour, eventually it is turned to the 'OFF' state once the threshold for the high temperature is reached. The code was written as a simple JavaScript functional node in separate Node-RED flow.

PV & Converter

[G. Mckoy]

The PV Converter simulation has two modes. The first is a script that generates random values depending on the time of day. The program generates values ranging from 30 to 80 percent of the maximum kW value produced by the PV between the hours of 9:00 and 17:00 to simulate daylight. For the remaining hours of the day the values are between 0 and 5 percent. The second version of the PV simulation makes use of the communication module 5.1.1 to receive actual readings from a real PV system. These values are sent to the Raspberry Pi over Modbus where they are processed and written to a file, PV_output_file.txt. The following is a description of the PV_output_file.txt file:

```
MODE: REAL_PV
CHARGE_RATE: 11
DISCHARGE_RATE: 0
CHARGING_BAT: True
```

```
        SUPPLYING_GRID: True
```

The *MODE* is used to tell if the system providing the values is a real or simulated one. *CHARGE_RATE* is the kW value being produced by the real or simulated system. *DISCHARGE_RATE* is the amount of energy in kW to be supplied to the grid. The *CHARGING_BAT* indicates if the battery is charging or discharging while *SUPPLYING_GRID* indicates if the converter is outputting to the grid.

Battery

[G. Mckoy]

To simulate the charging and discharging of the battery, a Python script was created that makes use of the communication module described in section 5.1.1. The script works by reading the contents of the converter output file and the Modbus register assigned to it in an infinite loop to know its current mode of operation.
The battery has three modes of operation: *CHARGING*,*DISCHARGING* and *None*. The state of the battery is stored in a text file so that the data is not lost on reboot of the Raspberry Pi. Below is a description of the construct of this file:

```
        CHARGING: None
        CHARGED: 1800000
        CAPACITY: 1800000
```

Upon running the script the values of the battery file are stored in memory to be used to help determine its mode of operation. To charge the battery a charge signal can be sent to the Pi over Modbus from Node-RED. This signal is in the form of a byte with each bit representing a different behaviour. Once the mode has been determined, the program checks the current state of the battery to see if it is fully charged or not. If the battery is fully charged, the charge command is ignored. If the battery is not fully charged, the program reads the value of the file to which the converter stores its output, PV_output_file.txt. After successfully reading the *CHARGE_RATE* value from the PV_output_file it is added to the state-of-charge of the battery. Should this value be larger than that of the remaining storage then the state-of-charge is set to that of the capacity.

To discharge the battery, a discharge signal similar to that of the charge signal can be sent from Node-RED. The program then checks, if the battery has a state-of-charge to meet the value stored in the discharge file. If the state-of-charge is more than the discharge value, then it is subtracted and the new state-of-charge stored. If the value is less, then it is set to zero. Once the state-of-charge value is set to zero the discharge command is ignored.

For the lab test, the controls of the battery are rooted into the state of the converter. The converter will charge the battery as long as the PV is producing more energy than the grid demands and the battery is not fully charged. If the energy being produced is less than the demand then the battery will automatically fill this gap. A slight increase can be seen in the output of the battery from time to time resulting from rounding during the conversion process.

5.2. Middleware Integration

[G. Mckoy & E. Madaha]

The middleware used, was based on the ongoing work from one of the OS4ES partners, and had to be integrated into the system to fulfil a complete OS4ES test environment. The following is a brief input on what was implemented to complete the integration of the middleware.

5.2.1. Retrieving the complete DER-Server Model

[E. Madaha]

On the Java Python Bridge some parts had to be worked on to aid in having a smooth communication over the OS4ES system. The server is initialized with a group of devices and configurations set by a yaml file. It is hard to debug during runtime and have an overall image of the state of the system. The implementation is based on a simple single parameter querying a specific devices or node. This hinders the development process by preventing the retrieval of a full representation of the system. For analysis and debugging it is essential to have the complete system view to see and understand the system behaviour in response to data changing on the server.

The process starts with analysing the Python code for the IEC 61850 server, and creating a new function within this class to allow the retrieval of the server model. Since it follows the same idea as the parameter querying function, with just a different output, this function was written under the same class, 'Element', which is a child of the server class. The function 'get_attributes()' shown in listing 5.2, is a recursive function that builds a dictionary of the complete element tree from a given element of the server, hence it returns all the objects under the element which it is called from.

```python
    def _iec_to_dict(self, srv, logical_devices):
"""
This function returns a dictionary with all the elements of the
DER server, providing a full system status for all given logical devices

@srv : The IEC 61850 server from the hat.drivers
@logical_devices : a dictionary of all logical devices

return : a dictionary of logical devices
   containing all the elements of the IEC 61850 server
"""
        dictDER={}
        for ld in logical_devices:
            dictLD={}
            for element in ld['elements']:
                dictLD.update(srv.get_attributes(ld['name'], [element['name']]))
                #print (srv.get_attributes(ld['name'], [element['name']]))
            dictDER[ld['name']]=dictLD
        return dictDER
```

Listing 5.2: Code snippet of the function recursively building the dictionary

If a logical device is used as the element for this function (5.2), then it is possible to get the complete object of the device expressed as a dictionary. If all the logical devices are put into a dictionary the entire server can be observed in the form of a massive dictionary tree. From the server side a function needs to be called for joining together all the logical devices' to form a dictionary that fully represents the entire IEC 61850 server. This is implemented in the function 'iec_to_dict()' shown in listing (5.3), which receives a dictionary of logical devices currently in the server and the server object itself, then returns the complete server dictionary.

```python
    def _iec_to_dict(self, srv, logical_devices):
"""
This function returns a dictionary with all the elements of the
DER server, providing a full system status for all given logical devices

@srv : The IEC 61850 server from the hat.drivers
@logical_devices : a dictionary of all logical devices

return : a dictionary of logical devices
   containing all the elements of the IEC 61850 server
"""
        dictDER={}
        for ld in logical_devices:
            dictLD={}
            for element in ld['elements']:
                dictLD.update(srv.get_attributes(ld['name'], [element['name']]))
                #print (srv.get_attributes(ld['name'], [element['name']]))
            dictDER[ld['name']]=dictLD
        return dictDER
```

Listing 5.3: Code snippet for writing the IEC 61850 server as a dictionary

As expressed earlier this function was fundamental for debugging and keeping track of multiple server changes during the modification of the Java Python Bridge. However, it was never used as part of the main code due to its Java counterpart not being able to represent the dictionary without changing some of the core behaviour and functionality of the middleware. Thus another purpose for this code would be to fully integrate it into the Java side whether with defined classes for all IEC 61850 elements or just a string based data structure such as the Python dictionary used from the Python side.

The modification would lead to full access of the server from the bridge rather than limitations on single parameters. As a proposition for further work, there could be a Java class built to read such a Python dictionary. With such improvements, the DER would be able to register its IEC 61850 data model defining all its energy services to the registry and avoid using a different service as the current solution which leaves the risk of data inconsistency.

5.2.2. Java Python Bridge

[G. Mckoy]

After carefully testing the Java Python Bridge it was determined that several functionalities to interact with the Smart Gateway were missing. The implementation was done inline with the design from the Test Environment and Design chapter 4 of this document. Listed below are some of the modifications made to the Java Python Bridge implementation:

- A better code description of the relevant logical nodes were implemented in order to allow reusability. All logical nodes were given an abstract base class to allow further extension of variables and methods should they be needed.
- The methods of the interface for the DER objects focus on extracting and parsing data to and from the respective logical nodes they contain. The methods now allows the code to be better separated from the communication section of the bridge.
- The communication section of the bridge that is responsible for instantiating the server was separated from the DER-objects. To replace this, a class test DER-System was created that implements the methods associated with sending and receiving information over the Java Python Bridge.

5.3. DER-System

[G. Mckoy & E. Madaha]

The section describes the implementation process associated with creating the DER-System from a programming point of view. A detailed description is given of how the individual components that make up the DER-System were created and the reason for the approach used. The section offers a further detailed look into the inner mechanism of the overall system discussed in Chapter 4 and how the overall implementation could be improved.

5.3.1. Smart Gateway

[G. Mckoy]

The Smart Gateway is not fully implemented according to the specification of the requirements analysis. The reason for which is strongly related to finding and fixing the problems associated with the JPB. The current implementation of the Smart Gateway is split between Java and Node-RED. The Java side works mainly as wire and merely passes information from the JPB with very little processing to Node-RED where it is processed and sent to the DER-units. This approach of the implementation was adapted due to time constraints and as such all abstract processing should later be removed from Node-RED and placed in the Java side of the implementation.

Node-RED

[G. Mckoy]

Figure 5.1 shows the communication flow of the Smart Gateway and the processing associated with each step. The incoming data is received by the use of the MQTT input node labelled, ***SM-Gateway In***, in the diagram. The information is then processed by the ***Process input*** node which determines if a real device or a simulation model is to be used and assigns it the correct topic. Once the correct topic is assigned, the switch ***Simulation / Real*** sends the message to its respective flow where further processing takes place.

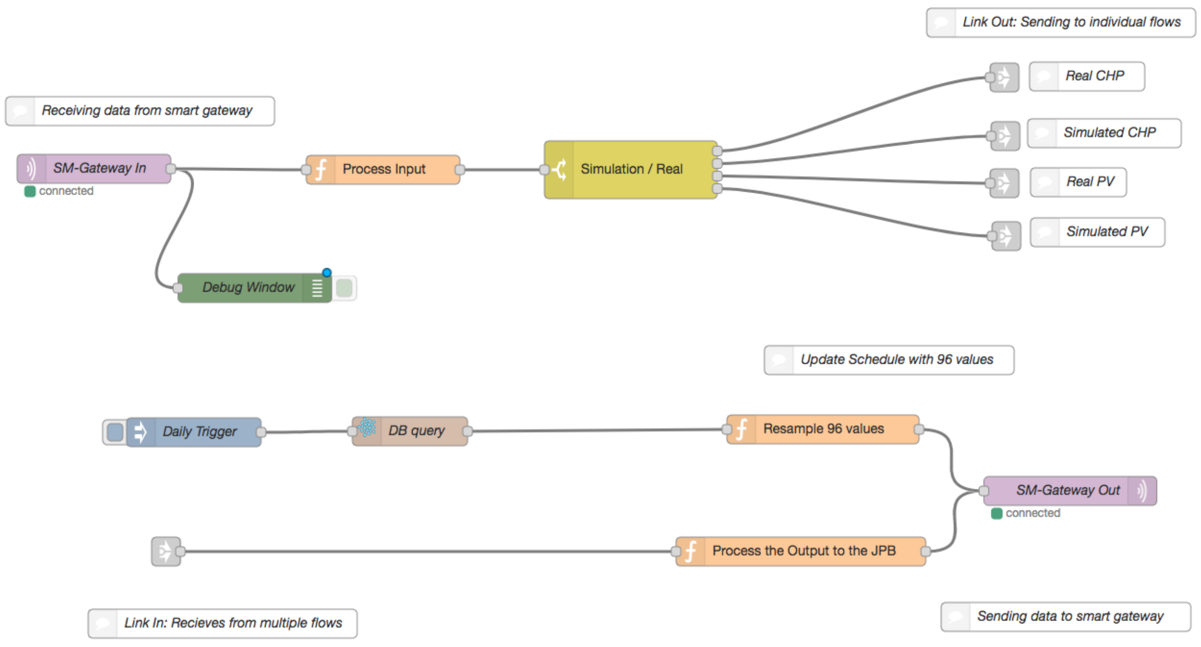

Link Out: Sending to individual flows
Real CHP
Simulated CHP
Real PV
Simulated PV
Receiving data from smart gateway
SM-Gateway In
connected
Process Input
Simulation / Real
Debug Window
Update Schedule with 96 values
Daily Trigger
DB query
Resample 96 values
SM-Gateway Out
connected
Process the Output to the JPB
Link In: Recieves from multiple flows
Sending data to smart gateway

Figure 5.1.: Node-RED communication flow

All information to be communicated to the Node-RED side of the Smart Gateway is received through the ***Link In*** node. This data is processed and formatted if necessary and sent back to the Java side of the Smart Gateway via the ***SM-Gateway Out*** node. The debug node is used to see the input and output of the connected node. This node is essential to the development process as it provides real-time values in the form of JSON objects which are easy to read and access.

Figure 5.2 shows the controls of the real CHP DER-unit used in the lab test. The flow is of greater complexity than that of the communication flow described in Figure 5.1. The flow has two sources of input, the Communication Flow and the ***CHP Energie-Campus*** node from the physical DER-unit itself. The information received from the physical device is used to determine the current status of the DER-unit whether it is on or off. This data is also sent back to the Communication Flow where it is published to the ***SM-Gateway Out*** MQTT node to have a central input and output point of reference. The information received from the communication flow is used in combination with the status of the CHP to power it on or off as well as set the rate at which the device is to be operated. The mode and rate of operation are then passed to the ***Turning on*** switch where it is filtered and sent to its destination.

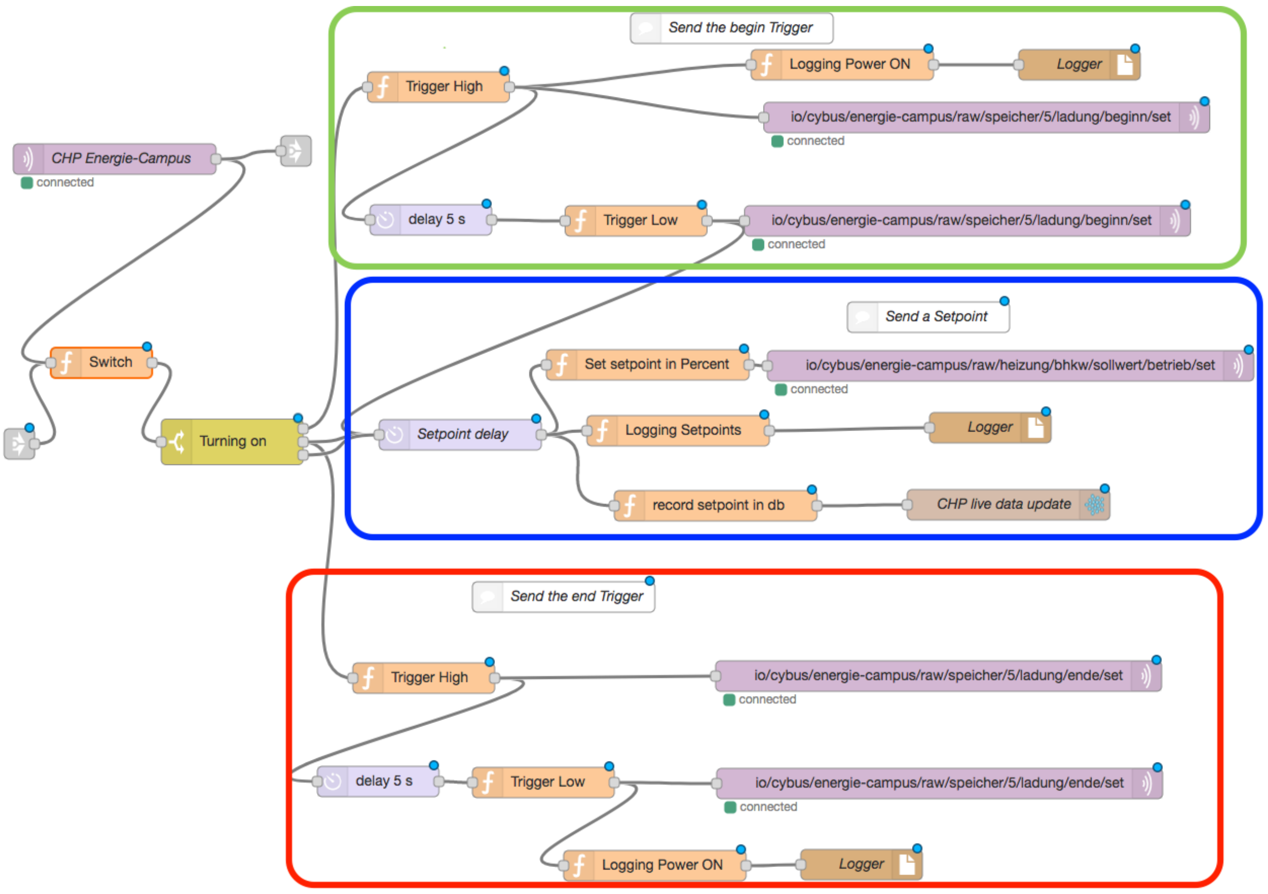


Figure 5.2.: Real CHP DER-unit controls flow

To simplify the rest of the flow, colour coded boxes are drawn to understand what each section of the flow is responsible for. Green represents turning on the DER-unit, blue represents setting a setpoint and red powering off the device. One additional functionality shared between all the controls is the logging of the data to a file. The logger is used to compare and analyse if the operations are done according to the setup described.

The blue box, responsible for setting the setpoints, has another important node inside; ***CHP live data update***. The node is responsible for logging the setpoints of the CHP to the influx time series database. This database logging along with the graphing tool Graphana allows for better analysis and comparison of the test results.

Powering on and off the CHP is of a relatively simple process which involves setting a trigger high and then low again via an MQTT publishing node. The MQTT publishing node is connected to the heat storage units that hold the hot water produced by the CHP. In total there are three storage devices where each can be communicated with separately. However, all three devices control the same CHP. The storage devices have an automatic power off functionality to ensure that the CHP production does not exceed storage capacity. The fail-safe should be taken into account when performing testing as the override might cause inconsistency in the expected results. More on the controls and behaviour of the CHP can be found in Appendix A.2.

Setting a setpoint is done by sending a JavaScript object, listing 5.4, responsible for adjusting the rate at which the CHP operates. The *type* indicates if the device being used is a real system and the *value* signifies the percentage at which to operate. Note the maximum and minimum operating values commented above the code.

```
\** CHP Operating Mode
* Min: 20
* Max: 80
* Values are in percentage,
* 1) if the max value is exceeded it will be set to 80
* 2) if the value is below 20 it will be set to 20
*\


msg.payload ={
  type: "real",
  value: msg.setpoint
};
```

Listing 5.4: Code used to set an operation rate from Node-RED

The setup of the real battery DER-unit is of a similar construct to that of the CHP and can be seen in Appendix B of this thesis. However, due to the battery not functioning as expected the simulation model described in the section 5.1 was used. To interact with the simulation model, the flow shown in Figure 5.3 was developed.

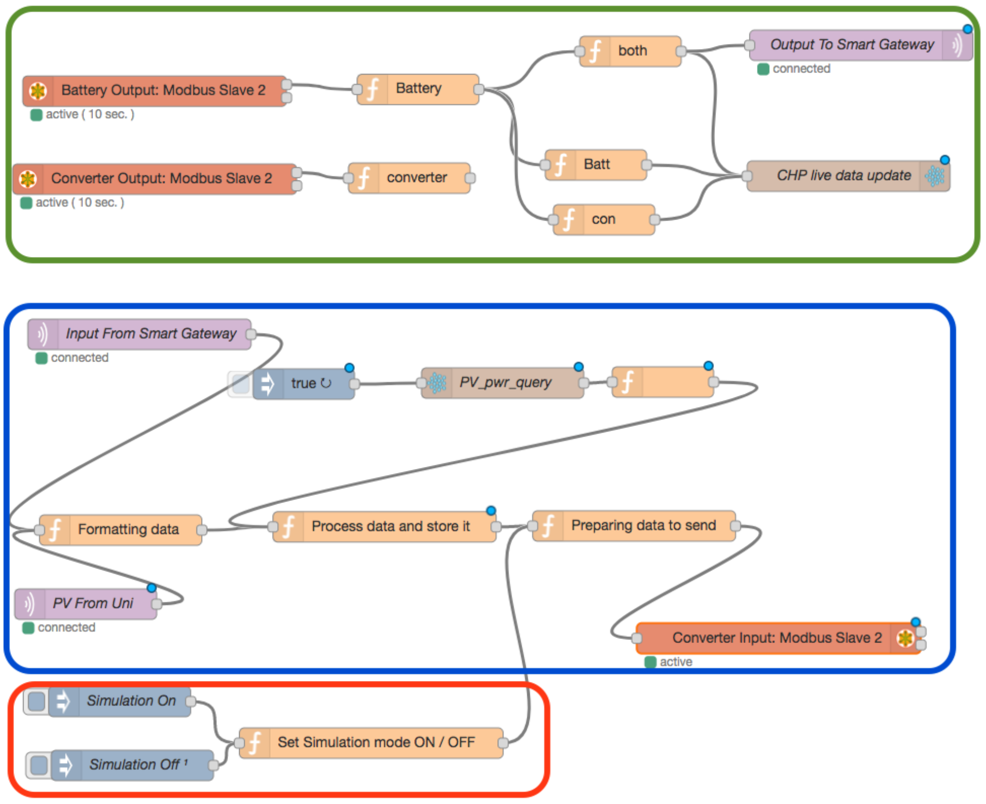


Figure 5.3.: Node-RED connection to the battery

Figure 5.3 is divided in three sections. The first section, in green, is the Modbus communication output of the battery and the converter. The values received are stored in the influx database for future analysis.

The second, in blue, is where the values to be sent to the converter are received and processed. The MQTT node ***input from smart gateway*** receives data from the setup for the real device test. The ***PV From Uni*** MQTT node receives values in kW from the real PV DER-unit. These values go through a conversion process and are later passed to the convert Modbus input node. The *textbfpv_pwr_query* node is used to read data from the past produced by the PV. This allows the simulation to take place regardless of the time of day or the weather conditions. The final section, in red, controls if the PV values to be used will come directly from the PV or if it will use the history from the influx database.

Java

[G. Mckoy]

The Java implementation of the DER-System, the registry and the aggregator extends the Java Python Bridge. The implementation of the three each use a static instance of the Java Python Bridge server and hash maps to pointing to different DER-unit instancies used to store data. The class diagram Figure 5.4 shows the construct of the test classes for the DER and the aggregator. The registry implementation is not shown in the diagram as it differs very little from that of the aggregator.

The TestDerSystem and TestOaaClient classes both implement the *DERSystem-ToIEC61850Methods* interface. The interface holds the common functionalities that exist between the client and the server. These functionalties are left to the user to describe. Descriptions of the *PV*, the *CHP* and the *BasicforecastingDer* can be found in the PV Model Description section 4.6.1.

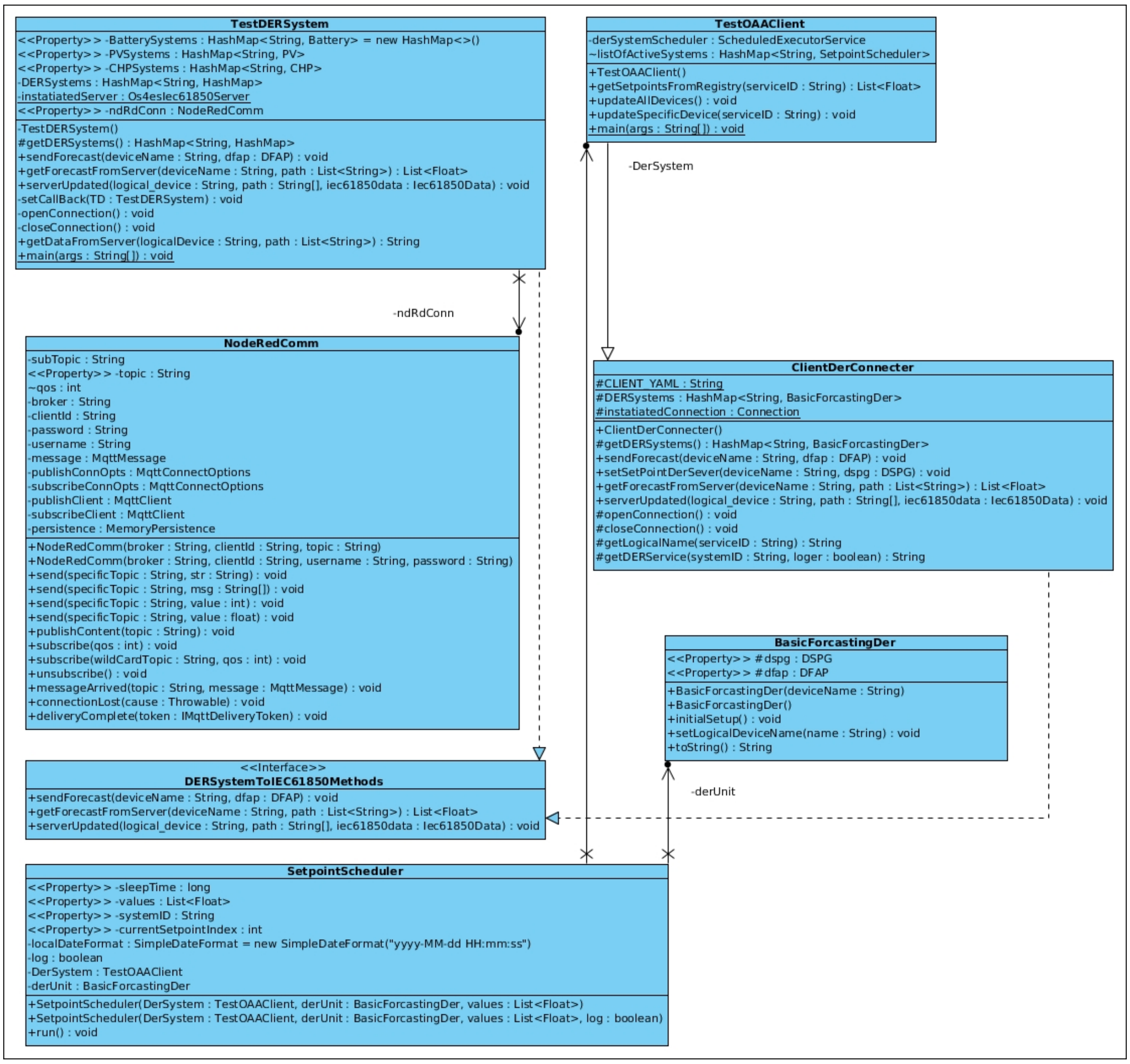

TestDERSystem
<<Property>> -BatterySystems : HashMap<String, Battery> = new HashMap<>()
<<Property>> -PVSystems : HashMap<String, PV>
<<Property>> -CHPSystems : HashMap<String, CHP>
-DERSystems : HashMap<String, HashMap>
-instatiatedServer : Os4eslec61850Server
<<Property>> -ndRdConn : NodeRedComm
-TestDERSystem()
#getDERSystems() : HashMap<String, HashMap>
+sendForecast(deviceName : String, dfap : DFAP) : void
+getForecastFromServer(deviceName : String, path : List<String>) : List<Float>
+serverUpdated(logical_device : String, path : String[], iec61850data : Iec61850Data) : void
-setCallBack(TD : TestDERSystem) : void
-openConnection() : void
-closeConnection() : void
+getDataFromServer(logicalDevice : String, path : List<String>) : String
+main(args : String[]) : void
TestOAAClient
-derSystemScheduler : ScheduledExecutorService
~listOfActiveSystems : HashMap<String, SetpointScheduler>
+TestOAAClient()
+getSetpointsFromRegistry(serviceID : String) : List<Float>
+updateAllDevices() : void
+updateSpecificDevice(serviceID : String) : void
+main(args : String[]) : void
-DerSystem
-ndRdConn
NodeRedComm
-subTopic : String
<<Property>> -topic : String
~qos : int
-broker : String
-clientId : String
-password : String
-username : String
-message : MqttMessage
-publishConnOpts : MqttConnectOptions
-subscribeConnOpts : MqttConnectOptions
-publishClient : MqttClient
-subscribeClient : MqttClient
-persistence : MemoryPersistence
+NodeRedComm(broker : String, clientId : String, topic : String)
+NodeRedComm(broker : String, clientId : String, username : String, password : String)
+send(specificTopic : String, str : String) : void
+send(specificTopic : String, msg : String[]) : void
+send(specificTopic : String, value : int) : void
+send(specificTopic : String, value : float) : void
+publishContent(topic : String) : void
+subscribe(qos : int) : void
+subscribe(wildCardTopic : String, qos : int) : void
+unsubscribe() : void
+messageArrived(topic : String, message : MqttMessage) : void
+connectionLost(cause : Throwable) : void
+deliveryComplete(token : IMqttDeliveryToken) : void
ClientDerConnecter
#CLIENT_YAML : String
#DERSystems : HashMap<String, BasicForcastingDer>
#instatiatedConnection : Connection
+ClientDerConnecter()
#getDERSystems() : HashMap<String, BasicForcastingDer>
+sendForecast(deviceName : String, dfap : DFAP) : void
+setSetPointDerSever(deviceName : String, dspg : DSPG) : void
+getForecastFromServer(deviceName : String, path : List<String>) : List<Float>
+serverUpdated(logical_device : String, path : String[], iec61850data : Iec61850Data) : void
#openConnection() : void
#closeConnection() : void
#getLogicalName(serviceID : String) : String
#getDERService(systemID : String, loger : boolean) : String
BasicForcastingDer
<<Property>> #dspg : DSPG
<<Property>> #dfap : DFAP
+BasicForcastingDer(deviceName : String)
+BasicForcastingDer()
+initialSetup() : void
+setLogicalDeviceName(name : String) : void
+toString() : String
-derUnit
<<Interface>>
DERSystemToIEC61850Methods
+sendForecast(deviceName : String, dfap : DFAP) : void
+getForecastFromServer(deviceName : String, path : List<String>) : List<Float>
+serverUpdated(logical_device : String, path : String[], iec61850data : Iec61850Data) : void
SetpointScheduler
<<Property>> -sleepTime : long
<<Property>> -values : List<Float>
<<Property>> -systemID : String
<<Property>> -currentSetpointIndex : int
-localDateFormat : SimpleDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")
-log : boolean
-DerSystem : TestOAAClient
-derUnit : BasicForcastingDer
+SetpointScheduler(DerSystem : TestOAAClient, derUnit : BasicForcastingDer, values : List<Float>)
+SetpointScheduler(DerSystem : TestOAAClient, derUnit : BasicForcastingDer, values : List<Float>, log : boolean)
+run() : void

Figure 5.4.: Class diagram of the DER and the aggregator test classes used for the lab test

The *TestDerSystem* class holds several key functions in combination with the methods of the *DERSystemToIEC61850Methods* interface used. The notable functions are:

1. *setCallBack*
2. *serverUpdated*
3. *sendForecast*

The *setCallBack* method ties into the Java Python Bridge by passing an instance of itself to the bridge so that its *serverUpdated* method can be called. In short, the method behaves like a callback function in other languages such as JavaScript and is triggered whenever data changes on the bridge. This allows the test class instance created to execute any of its other methods without explicitly monitoring when data changes on the DER-Server.

The *sendForecast* function is used to write the forecast of each DER-unit with a forecast (DFAP) to the server. The method takes in the name of the DER-unit as it is represented in the DER-Server and an instance of DFAP. The method then loops over each element of the array of Floats and converts them to the IEC 61850 syntax so that it can be sent over the Java Python Bridge.

The static instance of the *NodeRedComm* class is used to send and receive data from the DER-units connected to the Node-RED MQTT broker. The instance monitors the Node-RED side of the Smart Gateway and can track all changes published by a DER-unit.

The *TestOAAClient* class extends the *ClientDerConnector* class which holds the common functionalities shared between all client connections to the DER-Server. The *TestOAAClient* class has an instance of *ScheduledExecutorService* which is used to execute runnables created for each DER-service. The runnables are instances of the class *SetpointScheduler* which implements the Java Runnable class. These runnables are executed at specific time intervals depending on the specification of the device. This process takes place in the *updateAllDevices* and the *updateSpecificDevice* methods. The *getSetpointsFromRegistry* method is used to retrieve a DER-unit services from the registry, extract the forecast array parse it to a list of Floats and return it. These methods with along with the *setSetPointDerServer* method, from the super class *ClientDerConnector*, allow the system to read data from the registry and modify values in DER-Server.

5.3.2. Forecast and Flexibility

[E. Madaha]

One of the key features for the OS4ES system is that the DER devices should be able to give a forecast of the amount of power they can offer and relatively a flexibility range between the limits of the generation. Each DER has different forecasts depending on the constraints that determine its operation. A robust algorithm is needed to compute the factors associated with the forecast and the specific behaviour of the corresponding device.

For the PV the constraint is directly depending on the weather conditions, ideally so not much additional computation is required to give a forecast. While the CHP generator works together with the heat storage, therefore making the heat demand of the facility another constraint for the forecast, which means it needs the heat demand of the facility needs to be characterised by analysing the data from the heat storage.

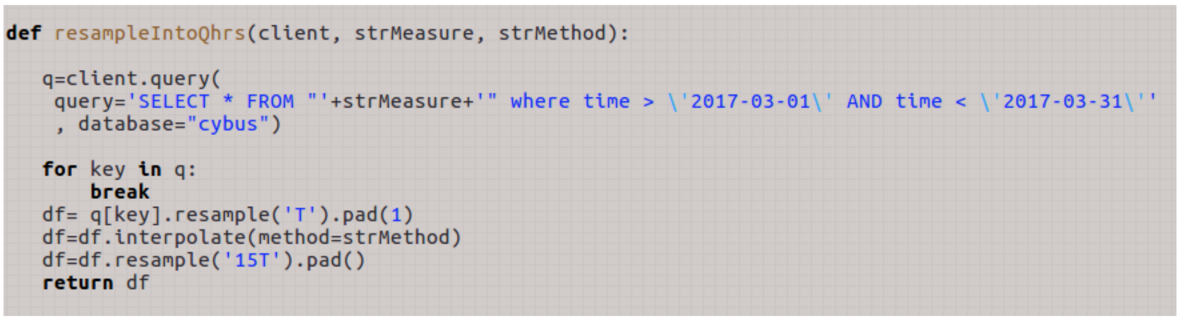
Re-sampling of the Day-ahead values

[E. Madaha]

The DER device is supposed to send a forecast of the availability for the day ahead to the Registry. This forecast is sent in 15-minute intervals over a 24 hours period of the next day; this results in a total of 96 values in the form of an array.

The data saved in the databases about the DER-System and multiple other measurements taken in from the DER system is unfortunately asynchronous and sparsely sampled. Ending up with 4-5 hour gaps between two sampled data points, this makes it difficult to pull a 24 hours query from the time-series database with 96 values, usually the outcome is 20-35 samples a day. Since the databases only record new data if there is a significant change, the frequent of data is not enough for a 96 point re-sampling.

This meant the system would need to use some data analysis and signal processing tools to extract a 96 value array from a much lower sampled series. For this implementation, the Python packages numpy and pandas were used for the resampling while also another package matplotlib for the display of the time series information in graphical plots.



```
def resampleIntoQhrs(client, strMeasure, strMethod):

   q=client.query(
    query='SELECT * FROM "'+strMeasure+'" where time > \'2017-03-01\' AND time < \'2017-03-31\''
    , database="cybus")

   for key in q:
       break
   df= q[key].resample('T').pad(1)
   df=df.interpolate(method=strMethod)
   df=df.resample('15T').pad()
   return df
```

Figure 5.5.: Code Snippet for the re-sampling of the values per 15-minute intervals

From the code in Figure 5.5 the function starts with a database query for the values within a given time frame, in this case, the test takes the whole month of March instead of just one day. The series is padded in one-minute intervals and then interpolated so that the series now has one minute period with real values. Then finally it is re-sampled to get it with 15-minute intervals. Thus, the re-sampling of the database values is completed.

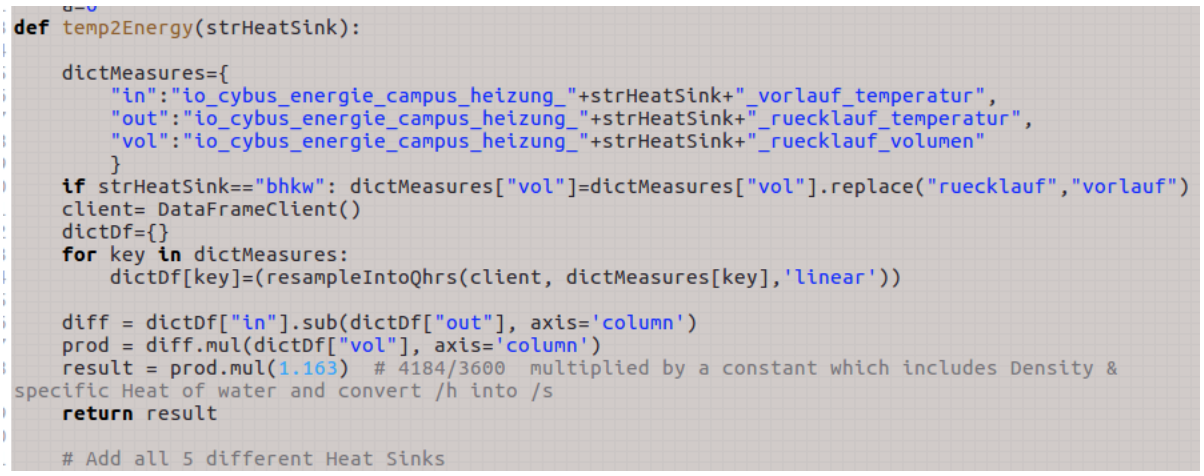
The goal of the implementation is to be able to reproduce 96 values from the previous day and send them over to a forecasting algorithm where other parameters such as the weather, a behavioural pattern of the system and a definition of the desired output; and the program would give an array of values for the forecasted availability of the device. Unfortunately, this forecasting algorithm was not ready by the time of the implementation so instead the historical values served as the published availability for testing purposes.

Heat Demand Aggregation

[E. Madaha]

For this implementation on the CHP system the Heat Demand was the main entity that needed forecasting. Since the CHP system is coupled together with the heat storage; hence the high heat demand of the heat storage triggers the CHP to run and the excess of heat triggers it to stop. Thus to fairly characterize the availability of the CHP the heat demand needs to be forecasted since the CHP does not have any constraints as it could be switched on and off at any time given that the Heat Demand allows it.

The building has 5 separate heating sections, all of which have to be re-sampled into 15-minute intervals and then added together to get the total heat demand figures. It should be noted that the values of the heat consumption is in temperature values, so additionally some with the help of other readings such as the volume of the water the heat value could be computed from the temperature, and this was done before adding the five sections.



```
def temp2Energy(strHeatSink):

    dictMeasures={
        "in":"io_cybus_energie_campus_heizung_"+strHeatSink+"_vorlauf_temperatur",
        "out":"io_cybus_energie_campus_heizung_"+strHeatSink+"_ruecklauf_temperatur",
        "vol":"io_cybus_energie_campus_heizung_"+strHeatSink+"_ruecklauf_volumen"
        }
    if strHeatSink=="bhkw": dictMeasures["vol"]=dictMeasures["vol"].replace("ruecklauf","vorlauf")
    client= DataFrameClient()
    dictDf={}
    for key in dictMeasures:
        dictDf[key]=(resampleIntoQhrs(client, dictMeasures[key],'linear'))

    diff = dictDf["in"].sub(dictDf["out"], axis='column')
    prod = diff.mul(dictDf["vol"], axis='column')
    result = prod.mul(1.163)  # 4184/3600  multiplied by a constant which includes Density &
specific Heat of water and convert /h into /s
    return result

    # Add all 5 different Heat Sinks
```

Figure 5.6.: Code Snippet for the computing of heat from Temperature

From the given piece of code in 5.6, this shows the calculation of heat from the water flow in m^3/h and temperature[°C] through the heat section. The formula used is composed from the Heat Transfer Formula [$Q = mc\Delta T$][1] optimised to give power in kilowatts to match the heat supply measured from the CHP. Whereby the mass is computed by multiplying the water volume by density (1000kg/m^3), the temperature difference is the same unit and since the water flow is already per unit time, so this already achieves energy per unit time which is power.

After the heat is derived from the database values of different heating sections of the facility, the five time series are added to get to one total heat load, with this value the heating can now be analysed. So the sum of all heat lost can be compared to the heat produced by the CHP now that the samples are within a reasonable frequency and the units match that from the CHP.

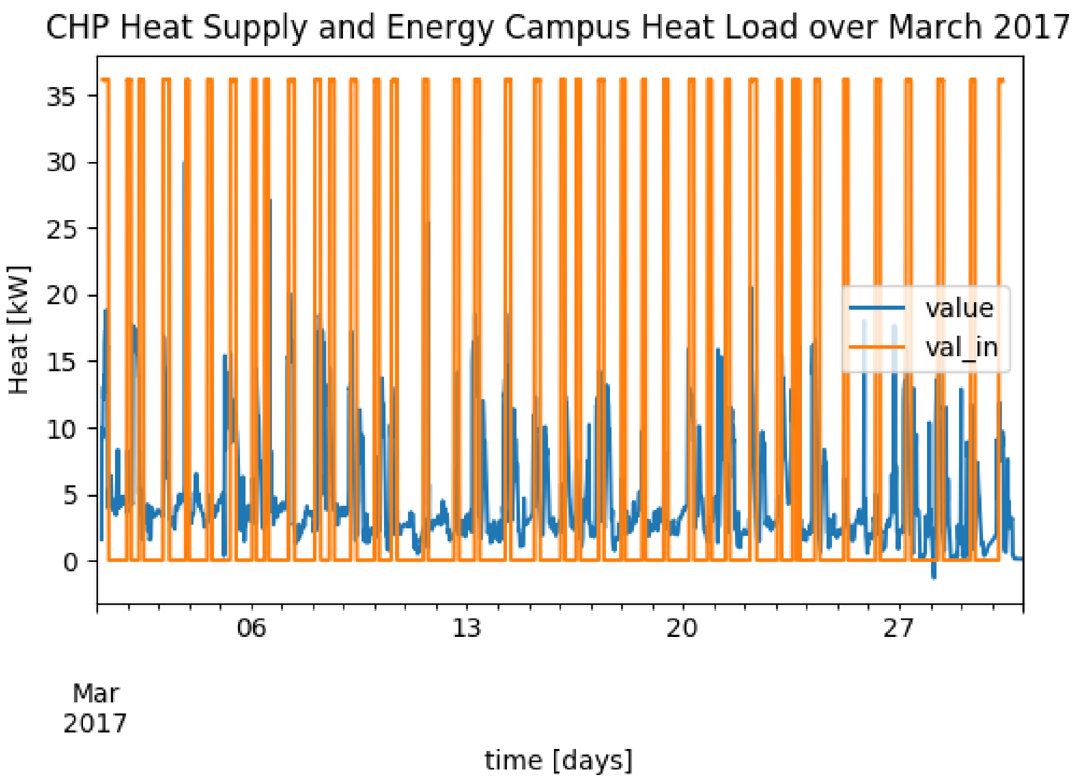


Figure 5.7.: Heat Load and Supply in kW over March

This plot 5.7, from matplotlib shows the heat usage of the building in this blue graph labelled 'value', which can clearly be observed to have a lot of noise, and also the other overlapping

[1] $Q = mc\Delta T$ where Q : is heat energy expressed as Joules or Watts*seconds, m is mass [kg], c specific heat constant (for water) expressed in kJ/(kg°C) and T is the Temperature difference works with any unit

graph of the plot, 'val_in' is the heat generated by the CHP, which appears more uniform as the generator produces consistent power.

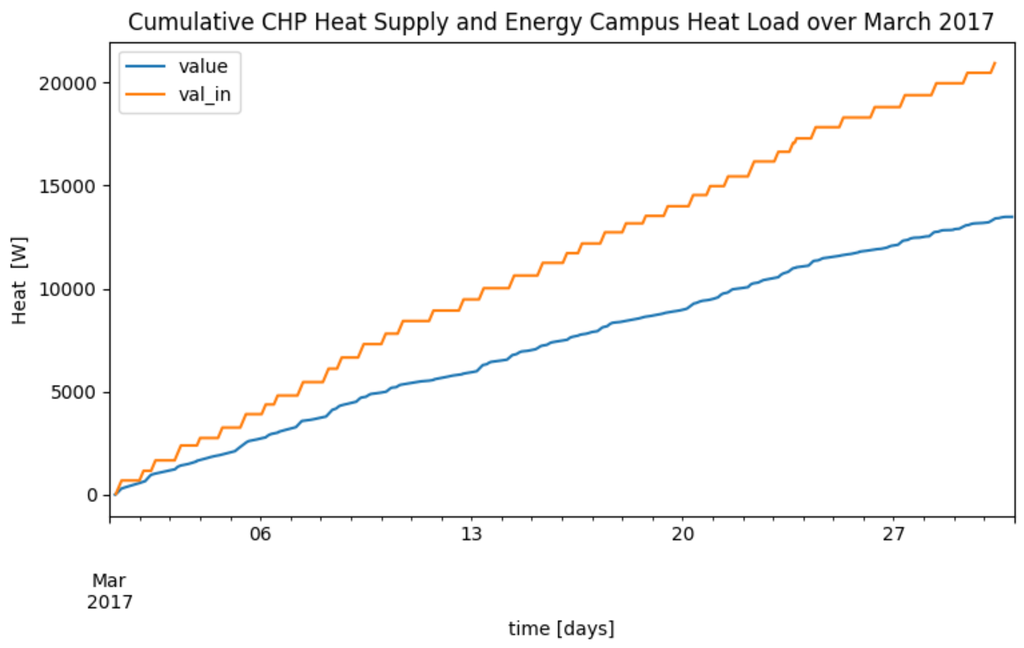


Figure 5.8.: Cumulative Heat Load and Supply in kW over March

And in this Figure 5.8 the two series are cumulatively compared to see how the sum of the heat consumed fares by the heat generated. The labels from this graph follows the same names as the previous one 5.7, 'val_in' for CHP and 'value' for the heat consumed. And a growing gap is seen to go on between the two values, meaning that over time there is more heat produced than heat consumed by the facility.

This work on this data ended here, mainly just resampling the data and aggregation of the heat load from the facility, and from this point it could span into two paths. One is that the data can be modified to suit a forecasting algorithm used to predict the future behaviour of the heat demand, which considers more factors, such as the human behaviour, the weather and other factors which contribute to the heating of the facility. Secondly, it can also be used in researching on that gap in between the heat consumed and produced, as this loss in heat could be resolved once discovered and help to make a more sustainable heating; since from the Figure 5.8 it shows an over 30% amount of loss over the month.

5.4. Full System Test

[G. Mckoy]

To perform a full system test, additional components are needed, specifically a registry and an aggregator. These components will store and request services to and from the DER-System respectively. The specification of these components is described below.

5.4.1. Additional Components Needed

[G. Mckoy]

Aggregator

[G. Mckoy]

The implementation of the aggregator uses a modified version of the Java Python Bridge client to connect to the DER-Server. The aggregator is implemented with the following functionalities:

- Read data from the registry server over the REST API related to a specific DER-unit.
- Write setpoints to the DER-Server.

Reading data from the registry is achieved by requesting a specific DER-unit from the registry as a JSON object and converting it to a Java object. Once converted to a Java object, an array of setpoints is extracted and set one by one on the DER-Server on a fifteen-minute interval via the JBP client.

Registry

[G. Mckoy]

The implementation of the registry uses a modified version of the Java Python Bridge client to connect to the DER-Server. The registry is implemented with the following functionalities:

- Read data from the registry server over the REST API related to a specific DER-unit.
- Write data to the registry server over the REST API to a specific DER-unit.
- Read data from the DER-Server.
- Write data to the DER-Server.

The registry works by first reading the DER-unit in question from the DER-Server and store the information about it in its respective object. The registry client then request the specific DER-unit from the registry as a JSON object and converting it to a Java object. Once converted, the availability is updated with the data read from the DER-Server. The DER-unit JSON object is then converted to a string and sent as an update back to the registry over the REST API.

5.4.2. Implementation Flaws

[G. Mckoy]

The registry lacks a subscription process that tracks when data has changed on the DER-Server. This implementation was given as part of the registry client provided by the other partner of the project. However, it was decided not to use this implementation due to its complexity and lack of documentation that would have resulted in not meeting the deadline of the end to end communication test.

Based on the construct of the JPB, data in the DER-Server can be changed by any of the three parties: the DER-System, th aggregator or the registry. This can lead to data corruption in the server or one party over writing before the information is processed by the DER-System. The aggregator and registry clients should be restricted in their write access to only what is necessary for them to pass on a request or command. Should they need to adjust another data point on the server this request should be made through the DER-Server where the server can decide if the request should or should not be processed.

5.5. Future Work

[G. Mckoy]

5.5.1. Rewriting the Implementation

[G. Mckoy]

Based on the Requirements Analysis the DER-System was developed in Java and was proven to be successful. However, the system would have been a lot easier to implement and build on if it were done in Python 3.5 in alignment with the IEC 61850 implementation by the other partners. The Java Python Bridge caused unnecessary complications, confusion and overhead to implement.

It is understandable that the JPB was needed for the registry and the aggregator as they were being developed parallel to the Python implementation of the IEC 61850, to run on each system as clients. On the other hand, the DER-System was developed later and could have used Python as its language of choice. Some of these reasons are outlined below:

1. Ability to better monitor activities on the IEC 61850 stack.
2. Python allows different objects to be stored in the same array without having to cast them back explicitly. i.e One could store a battery, PV and a CHP in the same array since they are all DER-units.
3. Reduce the complexity of converting to a specific data type by avoiding numerous function calls.
4. Easier adaptation of the source code to accommodate changes.
5. Faster and easier debugging of the source code.
6. A more direct path between the data received and being processed by the Smart Gateway. This also reduces the loss of data as the bridge does not support all data types.

The algorithms used in the Java implementation are very much applicable if it should be decided to rewrite the implementation in Python. The implementation uses standard Java functionalities which are common to Python as well with the exception of two external packages, the Paho-eclipse MQTT library and JSON library. To solve these problems there also exits a Paho MQTT package for Python. As for JSON, it is natively supported through the standard imports of Python; *import json*.

5.5.2. Extending on the Implementation

[G. Mckoy]

The current implementation is lacking the fully working functionality of the Smart Gateway to calculate the flexibility and proper logging of the data received and send out to and from the DER-System. Additionally, the sections of the implementation done on Node-RED that contain computational logic and conversion of data should be moved to Java or Python should the rewrite take place.

To accommodate these changes, the source code was modularised in such a way that code transferred from Node-RED can be integrated with very little modification to other parts of the code.

6. Results and Validation

This chapter presents mostly the work done at the lab facility in Bergedorf, also called the Energy Campus, where different energy resources can be found such as wind turbines, an electric car and specifically for this test the solar panels, battery and CHP. The tests for the OS4ES system are documented here in depth elaborating on the final outcome and behaviour of the framework when working on a real device and generating live electricity. Some physical aspects can be observed since the lab environment places real conditions on the system and the compatibility can be tested for these devices with OS4ES system.

6.1. Status of the Devices

[E. Madaha]

The conditions of the devices for the test were not exactly as expected though in an ideal situation they are supposed to be readily programmable and linked with a communication interface to allow remote access and certain controls when necessary. This section gives a brief overview on the status of these devices which are to be used in the testing of the the OS4ES system and also explain the chosen alternatives to overcome some of the difficulties faced.

6.1.1. Solar Panels (PV)

[E. Madaha]

The energy campus has a rooftop lined with solar panels with the capacity of upto 10kW, though given the grey skies dominating the weather, the average generation of 8000kWh over the year. The solar panels have sensors feeding into the system the continuous solar power generated and this data is logged with the rest of the measurements in a central database. The solar panels have already been incorporated into the facility and the use of them in this test does not require any controls rather just the electrical power being generated and the consequent allocation of this energy whether in battery storage, grid supply or being consumed by the facility itself.

6.1.2. Battery

[E. Madaha]

The status of the work on the battery is that it cannot be used for the lab tests of this project, due to some issues faced during the integration and setup. The main issue is an operational defect; where the battery keeps raising an error status every two hours, without any actual problem occurring. And the error kept the battery from running, until it was cleared then control of the battery could be regained.

Secondly, the battery always shutdown once the charge was below 10% and this occurred by default while the system never throws any warning about the low status of charge or a forthcoming shutdown. Once the battery dies from low charge, it cannot be restarted without the manufacturer's assistance hence this takes a while until a technician from the manufacturing company comes to fix it. And all this of course made it very difficult to operate and run any tests for the battery, especially with the continuous errors, hence the lab team which is responsible for integrating the device controls into software commands did not manage to fully accomplish their task.

Therefore the actual device for the battery was not used in the lab test but a simulation instead that was implemented on raspberry pi. This simulation is implemented as a clone of the battery which operates and follows the same specifications for the device. It charges with live PV values from the solar panels at the energy campus and discharges on the controls given to it from the Smart Gateway for responding to the setpoints.

6.1.3. CHP

[E. Madaha]

The CHP generator that was setup at the energy campus is a natural gas generator with an electrical power range of 5kW-16kW, giving a 31.5% electrical efficiency but achieves a 101% total efficiency due to the good utilization of the heat within its production cycle. This device is working together with a heat system and they are interdependent on their operation thus when using the generator the heat storage and demand should also be considered.

Some issues happened prior to the testing; where one of the temperature sensors involved with the heat storage was malfunctioning continuously reporting that the system was on critical temperatures hence keeping the CHP in a default mode; which limited the CHP controls. This problem was resolved by one of the engineers on site and the CHP was fully functional by the time the testing had to start.

6.2. Operation of the Devices

[E. Madaha]

On the lab perspective there are a few things to know as to understand the way these devices operate and before going into the testing of the system, here is a brief explanation to get it into context.

6.2.1. Controlling the CHP for the Lab Test

[E. Madaha]

The default operation mode for the CHP is completely automated and just based on satisfying the heat demands of the building/facility its connected to, then the electricity is more of a by-product to supplement to the buildings supply. So when the Heat Storage is not within the critical temperature range (low/high), that is when the CHP can be freely operated to be turned on/off either for testing or feeding the market demand to the smart grid.

Considering that the CHP and the heat storage act altogether as one system, the lab tests uses the heat storage parameters to switch on the CHP generator. Otherwise there would be a need to create a new interface for the direct control of the CHP via it's own parameters.

There is a "begin" and "end" trigger that need to be set for controlling the CHP operation and as for during the running time there is an operational setpoint for selecting the amount of electricity produced ranging from 5kW to 16kW.

Firstly the begin value is set to 1, and after some time (usually 5 seconds) it's set back to 0; since it works as a trigger so it does not need to stay at 1. This puts the heat storage's load state into Requested; which henceforth initiates the system to turn on the CHP.

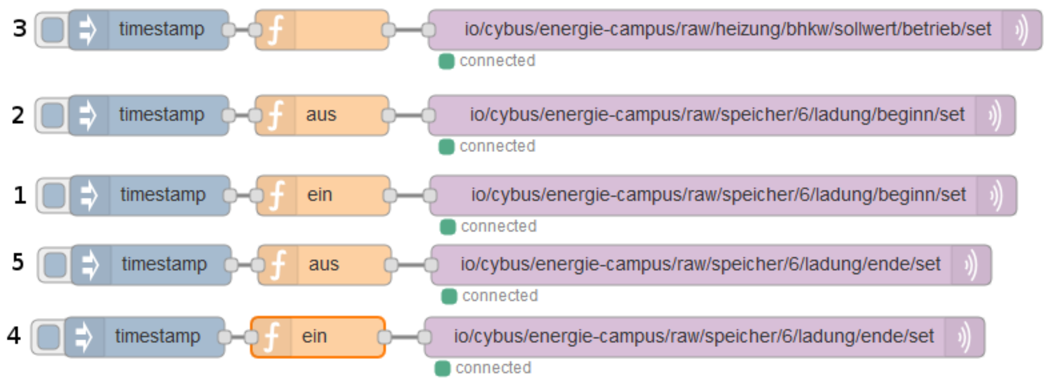

3
timestamp
io/cybus/energie-campus/raw/heizung/bhkw/sollwert/betrieb/set
connected
2
timestamp
aus
io/cybus/energie-campus/raw/speicher/6/ladung/beginn/set
connected
1
timestamp
ein
io/cybus/energie-campus/raw/speicher/6/ladung/beginn/set
connected
5
timestamp
aus
io/cybus/energie-campus/raw/speicher/6/ladung/ende/set
connected
4
timestamp
ein
io/cybus/energie-campus/raw/speicher/6/ladung/ende/set
connected

Figure 6.1.: Node-RED Flow showing the running of a CHP

The flow above represents the whole procedure, though the nodes are not in sequential order hence the numerical guideline to define the sequence.

Once the CHP is running, the magnitude of electrical power its producing can be changed by sending a setpoint which will consequently adjust the power to the desired measure.

Finally to turn the generator off the "end" parameter is triggered to initiate the shutdown procedure by the setting the Load status of the heat storage back to Not Requested.

6.2.2. Key Constraint for the CHP

[E. Madaha]

The Operational Setpoint (sollwert/betrieb) functions as the value that will directly change the amount of power being generated. The value is set in terms of percentage where the operable range is 20%-80% which reflects the CHP's operable range 5 kW-16kW. Anything set that exceeds the operable range is considered the same as the limiting value; meaning the maximum or minimum depending on the lower or upper limit.

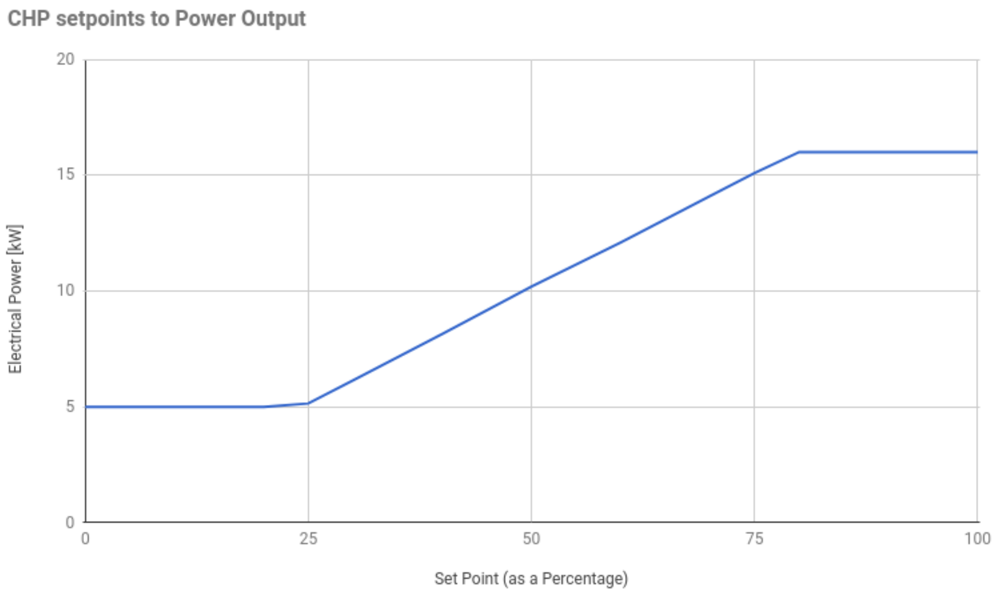


Figure 6.2.: Electrical Power[kW] to Setpoint (Percentage) relationship

6.3. Results of the First Preliminary Test

[E. Madaha]

On the PV system there was no lab test due to the technical issues going on with the battery, which also influenced in the incomplete communication interface for controlling the battery.

Mainly the issues were that the battery kept on sending an error status, even when there was no problem with it. And clearing the error would release it for a while but then it would eventually return into this state, rendering it only operable for an hour, two at most.

With the CHP system the outcome was better as it was able to successfully run and adapt to new setpoints during runtime, so it responded in the expected behaviour. So far no technical difficulties were found and the communication interface confirmed to be fully capable of handling the procedure. This test also stood to prove that the DER-Systems can be handled remotely (in this case only the CHP), since the all operational commands were set via the Smart Gateway which runs on a different network.

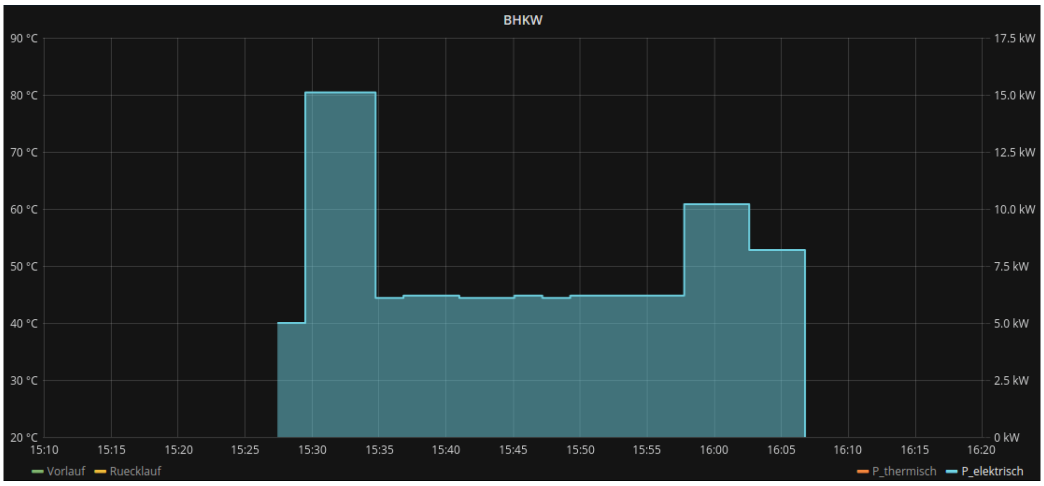


Figure 6.3.: The CHP output for the 1st Test run

The screenshot above displays the test results from Grafana, where the CHP parameters are queried for the specific time window of this test. The blue graph is representing the Electrical power generated by the CHP in terms of kW, la belled from the left axis.

Seen here is a demonstration of how the power generation can be successfully manipulated remotely, even adjusting its magnitude during runtime. At a closer look the CHP starts at the minimum level 5kW then changes to 15kW following with 6kW for a while, then 10kW and finally 8kW. All the changes here are sent via the Smart Gateway specifically by writing a new value and triggering the node labelled 3 from the figure x above.

Along with the fully functioning capabilities of the CHP a subtle but significant issue was uncovered with the timing of all the communication. It was not fully measured during the test but the delay was clearly observable between sending the setpoint and having the generator respond.

6.3.1. Assessment

[E. Madaha]

The battery could not be tested due to some malfunctioning, while the CHP tests worked and led to the discovery of a significant delay. Thus, for the CHP some more investigation must be put in, so that the delay can be well handled within the operations.

The first and most clear assumption for the delay is simply the fact that the CHP generator takes some setup time (probably running some requirement checks) then it gets ready. Even once it's running there is again time needed until stable Electrical power is actually generated, since the conversion isn't immediate.

Also, another proposal was that the CHP takes different times to turn on depending on how warm it is, or how long has it been off for. Meaning that there would be a Cold startup time which would be longer and a warm startup time, whereby the difference is from the idea that the CHP needs to warm up before it starts.

Lastly a certain thing to acknowledge is that there is a prerequisite of sending a trigger through the heat storage, as explained above, and this alone accounts for a small delay.

Conclusively a second test for the CHP is necessary for modelling the Simulated CHP with a defined time delay behaviour from precise measurements, between the receiving of a set-point from the Aggregator and the CHP generating the requested amount of Power.

6.4. Second Preliminary Test

[E. Madaha]

6.4.1. Objectives

[E. Madaha]

For the second lab test that was also conducted at the energy campus in Bergedorf. The main objectives were the measurement of the time delay for the CHP and to try out the battery test again. An added change for the tests was some automation in the startup procedure. To help in finding a precise time measurement between the arrival of a control command and the response of the CHP.

The test here works to interrogate the theories deduced in the assessment of the first preliminary test based on the time delay. Including whether the delay differs between start-up when the CHP is in a cold state, "Cold Delay" and when it’s in a warm state, "Warm Delay".

Whereby Cold State is when the CHP has a relatively cold temperature and it has been turned off for a long time, considerably for a day or more. While the warm state is the state of the CHP being relatively warmer than usual and it has just been running recently, i.e. within the past hour.

6.4.2. Operational adjustments

[E. Madaha]

The flow for the test is modified to limit all the manual clicking involved previously, refer to figure x since this involved human interaction which could lead to inaccurate timing. Thus, after the first click the rest of the flow is propagated through delay nodes which account for the required wait time between each of the steps.

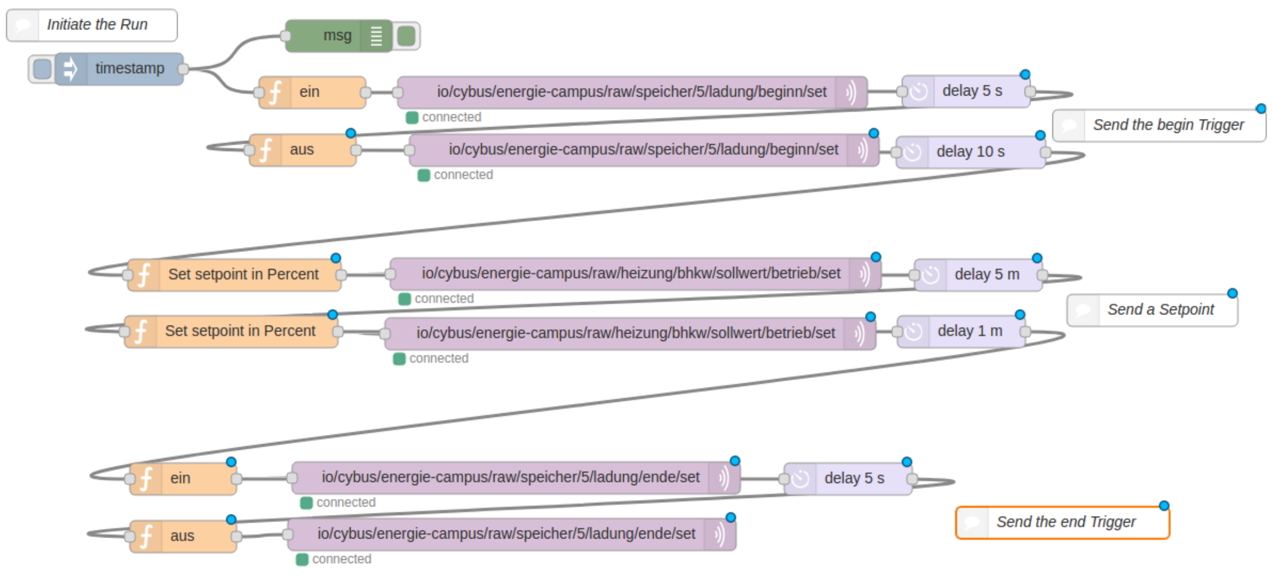

Initiate the Run
timestamp
msg
ein
io/cybus/energie-campus/raw/speicher/5/ladung/beginn/set
connected
delay 5 s
Send the begin Trigger
aus
io/cybus/energie-campus/raw/speicher/5/ladung/beginn/set
connected
delay 10 s
Set setpoint in Percent
io/cybus/energie-campus/raw/heizung/bhkw/sollwert/betrieb/set
connected
delay 5 m
Send a Setpoint
Set setpoint in Percent
io/cybus/energie-campus/raw/heizung/bhkw/sollwert/betrieb/set
connected
delay 1 m
ein
io/cybus/energie-campus/raw/speicher/5/ladung/ende/set
connected
delay 5 s
Send the end Trigger
aus
io/cybus/energie-campus/raw/speicher/5/ladung/ende/set
connected

Figure 6.4.: Flow showing the automated running with delay nodes

The flow in figure x shows the updated flow of the test, where the previous functionalities are now automated and there is an additional setpoint added in between the flow.

6.4.3. Cold Delay

[E. Madaha]

The first run of the CHP for this test had to be one where it is in a cold state, whereas it has not been running for a long time. And for this test, it had been off for at least 48 hours. Thus, it is enough to say that it is in a cold state. So the response time expected here is much longer than when the CHP generator has been already warm.

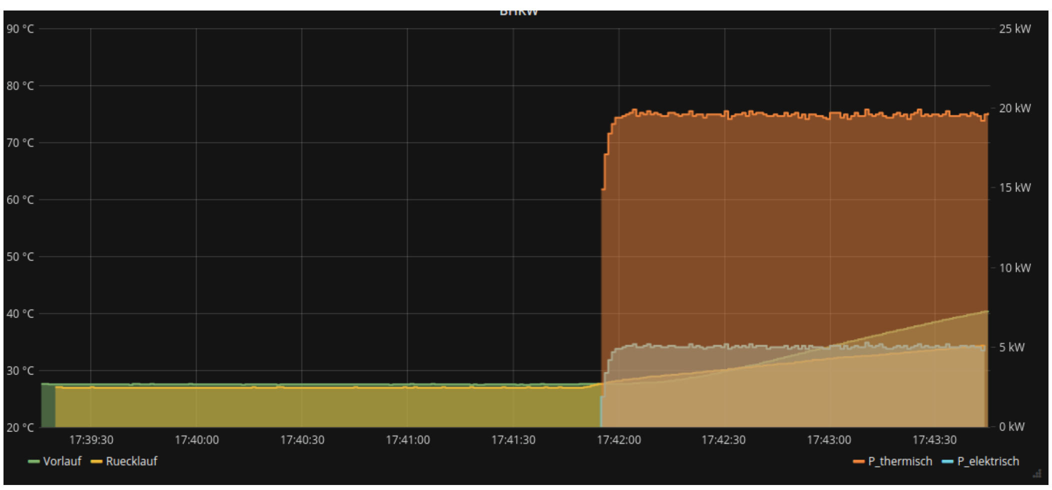


Figure 6.5.: Grafana depicting the CHP startup from a cold state

Temperatures of the water flowing through the CHP is represented by the yellow line for inflow and green line for outflow. This is well below 30°C, and suffices as the cold state for the system. Later on it gradually begins to rise once the CHP is on.

Power generated by the CHP is labelled on the left side, where the graph in orange is the thermal power produced, and blue is electrical. The thermal power being always higher given that heat is the primary output of the CHP.

The command sent at 17:37:22 initiated the heat storage to trigger the start of the CHP generator. The generator started running at 17:41:54 seconds, roundabout four and a half minutes from the time the command was set. Therefore, the measure called delay for the CHP generator was found to be four and a half minutes but unfortunately, it was not easy to get an average since it would take too long for to get cold again so this is the value that will be used for now.

6.4.4. Warm Delay

[E. Madaha]

For the warm delay, the same procedure was repeated only this time a few minutes right after the first test was conducted. Since in this case, it is already known that the CHP generator is quite warm due to the fact it was left running for at least half an hour at full power (

16 kilowatts). The major difference now is that multiple tests can be made since the CHP generator will always be warm after running.

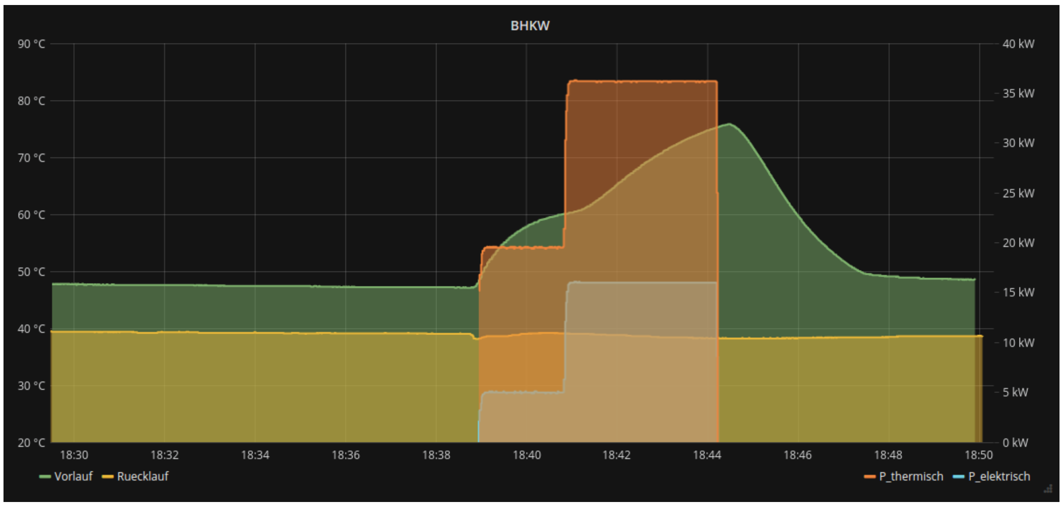


Figure 6.6.: CHP stats from the warm state startup

The first test for the warm delay started at the time 18:34:29, with a command from the Smart Gateway. The CHP generator started running at 18:38:56, which would result in a similar time frame as that of the cold delay four and a half minutes. Thus proving the CHP delay is not related to the temperature of the CHP generator at the moment it is requested to run, rather it is a common delay caused by the amount of time takes to start up the whole system.

As for this case the temperature of the water flowing through the CHP is clearly higher, approaching 50°C and the difference between the water going in and out is between 8°C-10°C. This indicates that the system is at a sufficiently warm state, the previous run was a couple of minutes earlier.

Further on two more tests were conducted to accumulate a good average for the time it takes the CHP to startup. This is fundamental for the Aggregator to know when should the order take effect, so that it can be accounted for beforehand.

The third and fourth CHP runs for this test, followed the same steps for starting up the system as done in the two first cases of the test. After the startup, another step was added to analyse further the response time of the system during runtime. Because the main three functions that should be tested are turning on the CHP, switching it off and adjusting its output. Since a delay was found in turning it on, so the other two functions also need to be measured to fully account for the CHP's performance.

The additional step is found at the "Send a Setpoint" Tag from the flow in the figure x, specifically the set of nodes with the "delay 1 m" at the end. So 5 minutes after the first setpoint, the output of the generator should change its magnitude to the new setpoint.

Even for the earlier cases in figures [] the CHP always starts at the minimal output of 5kW after some minutes changes to the given setpoints. While clearly from the flow (figure x)there is only a 10 second wait in between the begin trigger and the first setpoint. So there is time taken for the CHP to start up and furthermore additional time for the CHP to build up the output to the set magnitude. And altogether these different times need to be fully characterised to grasp the full behaviour of the CHP and heat storage system.

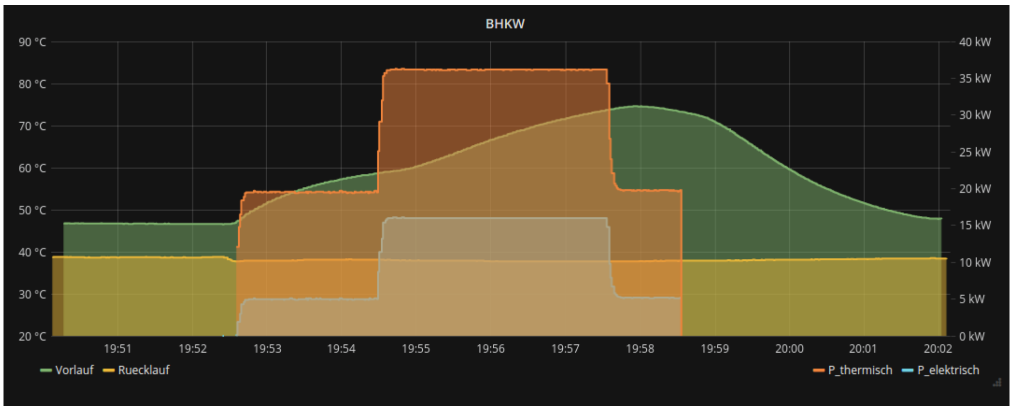


Figure 6.7.: CHP stats after sending the 1st and 2nd setpoints

The results from the last run in the figure x; show a similar behaviour to the earlier warm delay results but only the additional tail in the end where the value is changed back to minimum output.

Battery

[E. Madaha]

The battery was performing a lot better than in the first test run it was operational under the manufacturer's software. As it could discharge and charge to any value within its range. But the problem was found in trying to accomplish the operations remotely over the mqtt Modbus communication. Even though the programming was properly set, it was not yet fully tested so the precise problem in the communication was hard to figure out.

6.4.5. Results and Analysis of Preliminary Tests

[E. Madaha]

The first analysis is to simply compare the different start up times from all the four runs of the system. Resolve the assumption about whether the cold state or the warm state of the system in any way affect the startup time. When the exact times are tabulated and the differences computed as seen in the table, the comparison is clearly seen.

Test	Time of sending setpoint	Time of CHP turning on	Time difference (delay)
Cold Delay	17:37:25	17:41:55	00:04:30
Warm Delay 1	18:34:29	18:38:56	00:04:27
Warm Delay 2	19:17:10	19:21:37	00:04:27
Warm Delay 3	19:48:09	19:52:36	00:04:27

Table 6.1.: The Timing results for the CHP startup time measured

With only a 3 seconds difference between the cold run and the warm run, this theory can be discarded given that the difference only accounts for 1.1% percent of the total time delay. So the temperature state of the CHP does not affect the delay rather the delay is approximately the same for both cases.

Still, the reaction time of the system needs to be also analysed to fully describe the performance of the CHP as a DER-System. As mentioned earlier after the CHP turns on it takes a while to achieve the output from the 1st setpoint sent, also the same occurs when a 2nd setpoint is sent during runtime.

And here both of the times are analysed; while as for the 2nd setpoint the sending time can be computed from the delays in the flow. This time is 00:05:15 from initial start time, also it was corroborated during the test to be the same, since the tasks in between are executed just in milliseconds.

From the retrieved information two main points can be deduced.

- The CHP takes approximately 4:30 minutes to respond (+/- 15 seconds from the 2nd setpoint)
- The CHP needs a 2 minute minimal power run at startup (from b-a)

Test	Time of sending setpoint	Time of CHP turning on	Time of 1st setpoint change	Time of 2nd setpoint change	Startup delay	1st setpoint Delay	2nd setpoint delay
Calculation	I	II	III	IV	a = I-II	b=I-III	c=I-IV-05:15
Cold Delay	17:37:25	17:41:55	17:43:49	-	00:04:30	00:06:24	-
Warm Delay 1	18:34:29	18:38:56	18:40:53	-	00:04:27	00:06:24	-
Warm Delay 2	19:17:10	19:21:37	19:23:38	19:26:46	00:04:27	00:06:28	00:04:21
Warm Delay 3	19:48:09	19:52:36	19:54:36	19:57:42	00:04:27	00:06:27	00:04:16

Table 6.2.: The CHP Timing Analysing the 1st and 2nd Setpoints Response

Preliminary Test Conclusion

[E. Madaha]

Finally the test can affirm that the CHP is ready for the final lab test and the time behaviour can be characterised for any simulation purposes. The main purpose for the prep tests is to make sure that when a setpoint comes from the Smart Gateway the DER device is able to respond accordingly. And this case was successfully proven for one of the DER devices, CHP namely. The battery unfortunately was not remotely operable, though once the communication interface is properly functioning, the device should be fully integratable.

6.5. Final Lab Test

[E. Madaha]

The final test here is to prove that the main requirements 3.1 of this thesis have been accomplished; that is the End-to-end communication and the successful DER integration. This test can fundamentally be put into three sections as in each leg of the trilateral interaction between the DER-System, registry and the aggregator.

The first step of the test is the DER-System registering itself on the registry and following with an offer of a flexibility schedule from its energy service. Then the aggregator queries the Registry and selects an offer,then pulls the offer's profile which includes the information about the DER and the flexibility schedule. Finally the aggregator should send setpoints to the DER-System according to the offer sent and the DER-System should generate power to the level set by the aggregator, this should go on until the aggregator completes the sequence of setpoints.

The procedures described in sections 6.5.1, 6.5.2 and 6.5.3 can be applied to most if not all DER-units, energy services or offers with a flexibility. The CHP was chosen for the full description because it was the only working real device that could be used for the lab test and as such the steps are of a more hands on process.

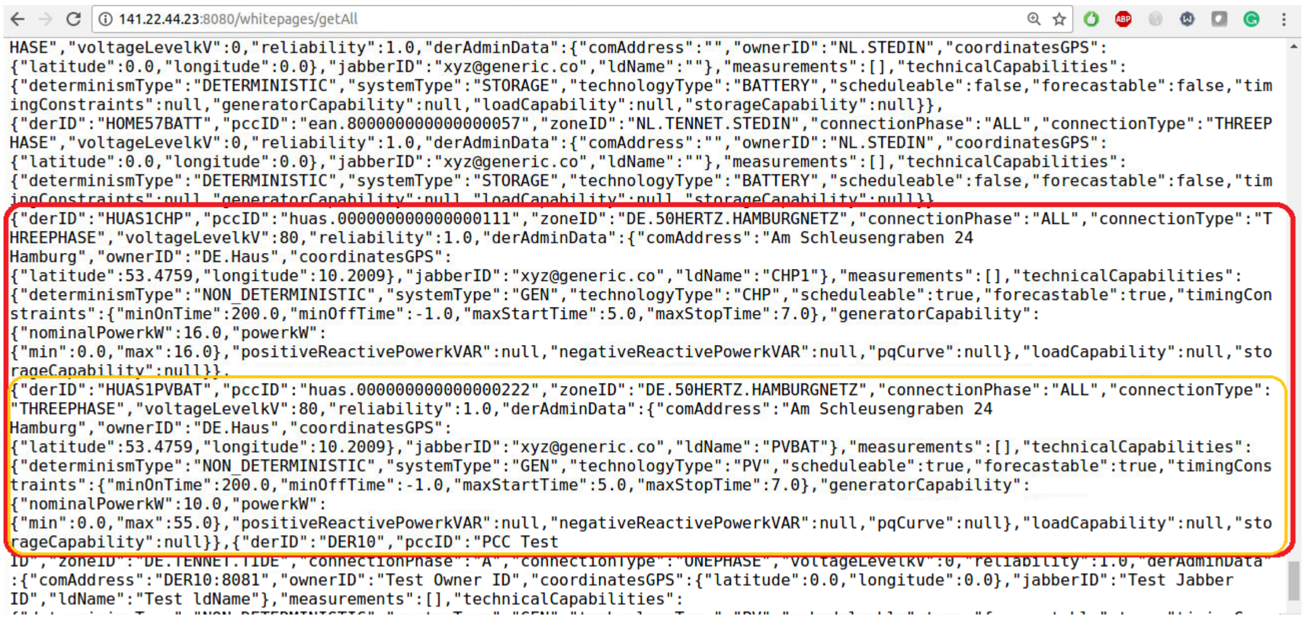
6.5.1. DER-System Registration

[E. Madaha]

For a DER-System to get registered on the registry there are two main things to be considered about the registry. The registry has a white page; which is used for registering a DER-System and keeping all the basic information about it here; also there is a yellow page; which holds the list for the available energy services of registered DER-Systems. So this process has three steps :

- Registration of a DER-System on the white pages of the registry.
- Registration of the energy service on the yellow pages of the registry.
- Update the energy service with an offer of 96 values.

The Registry is running on a different server and the DER-System needs to run a client to establish a connection on the server. The server is implemented with a REST API so the DER-System used a REST client to post the registration info following the defined JSON structure given by the Registry. This communication was intended to run on the IEC 61850 standard under the implemented middleware but unfortunately some features were not fully completed on the middleware, thus allowing the REST API to be a simpler alternative.

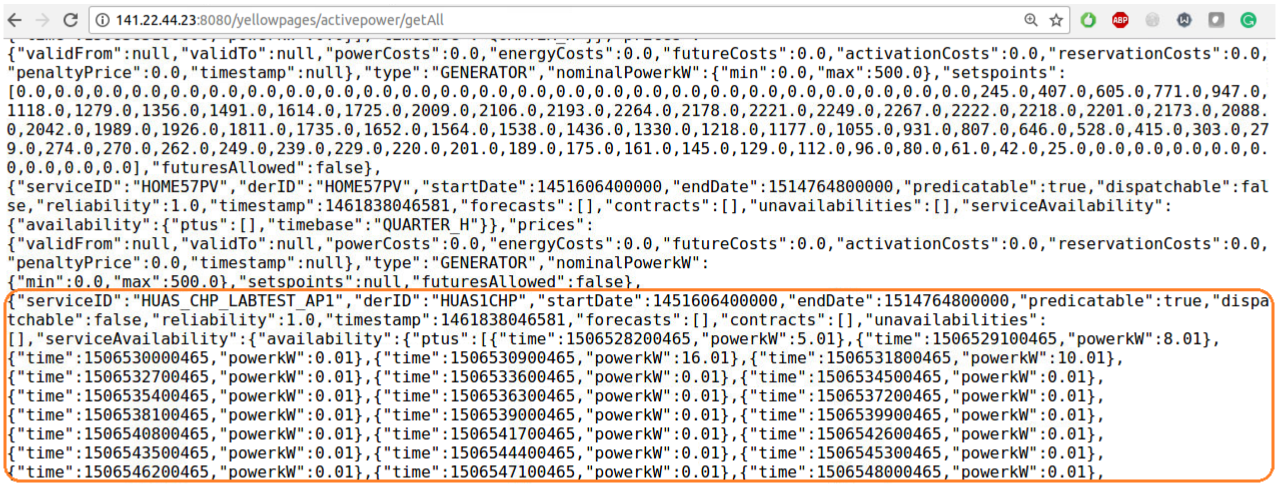


```
HASE","voltageLevelkV":0,"reliability":1.0,"derAdminData":{"comAddress":"","ownerID":"NL.STEDIN","coordinatesGPS":
{"latitude":0.0,"longitude":0.0},"jabberID":"xyz@generic.co","ldName":""},"measurements":[],"technicalCapabilities":
{"determinismType":"DETERMINISTIC","systemType":"STORAGE","technologyType":"BATTERY","scheduleable":false,"forecastable":false,"tim
ingConstraints":null,"generatorCapability":null,"loadCapability":null,"storageCapability":null}},
{"derID":"HOME57BATT","pccID":"ean.800000000000000057","zoneID":"NL.TENNET.STEDIN","connectionPhase":"ALL","connectionType":"THREEP
HASE","voltageLevelkV":0,"reliability":1.0,"derAdminData":{"comAddress":"","ownerID":"NL.STEDIN","coordinatesGPS":
{"latitude":0.0,"longitude":0.0},"jabberID":"xyz@generic.co","ldName":""},"measurements":[],"technicalCapabilities":
{"determinismType":"DETERMINISTIC","systemType":"STORAGE","technologyType":"BATTERY","scheduleable":false,"forecastable":false,"tim
ingConstraints":null,"generatorCapability":null,"loadCapability":null,"storageCapability":null}},
{"derID":"HUAS1CHP","pccID":"huas.000000000000000111","zoneID":"DE.50HERTZ.HAMBURGNETZ","connectionPhase":"ALL","connectionType":"T
HREEPHASE","voltageLevelkV":80,"reliability":1.0,"derAdminData":{"comAddress":"Am Schleusengraben 24
Hamburg","ownerID":"DE.Haus","coordinatesGPS":
{"latitude":53.4759,"longitude":10.2009},"jabberID":"xyz@generic.co","ldName":"CHP1"},"measurements":[],"technicalCapabilities":
{"determinismType":"NON_DETERMINISTIC","systemType":"GEN","technologyType":"CHP","scheduleable":true,"forecastable":true,"timingCon
straints":{"minOnTime":200.0,"minOffTime":-1.0,"maxStartTime":5.0,"maxStopTime":7.0},"generatorCapability":
{"nominalPowerkW":16.0,"powerkW":
{"min":0.0,"max":16.0},"positiveReactivePowerkVAR":null,"negativeReactivePowerkVAR":null,"pqCurve":null},"loadCapability":null,"sto
rageCapability":null}},
{"derID":"HUAS1PVBAT","pccID":"huas.000000000000000222","zoneID":"DE.50HERTZ.HAMBURGNETZ","connectionPhase":"ALL","connectionType":
"THREEPHASE","voltageLevelkV":80,"reliability":1.0,"derAdminData":{"comAddress":"Am Schleusengraben 24
Hamburg","ownerID":"DE.Haus","coordinatesGPS":
{"latitude":53.4759,"longitude":10.2009},"jabberID":"xyz@generic.co","ldName":"PVBAT"},"measurements":[],"technicalCapabilities":
{"determinismType":"NON_DETERMINISTIC","systemType":"GEN","technologyType":"PV","scheduleable":true,"forecastable":true,"timingCons
traints":{"minOnTime":200.0,"minOffTime":-1.0,"maxStartTime":5.0,"maxStopTime":7.0},"generatorCapability":
{"nominalPowerkW":10.0,"powerkW":
{"min":0.0,"max":55.0},"positiveReactivePowerkVAR":null,"negativeReactivePowerkVAR":null,"pqCurve":null},"loadCapability":null,"sto
rageCapability":null}},{"derID":"DER10","pccID":"PCC Test
ID","zoneID":"DE.TENNET.TIDE","connectionPhase":"A","connectionType":"ONEPHASE","voltageLevelkV":0,"reliability":1.0,"derAdminData"
:{"comAddress":"DER10:8081","ownerID":"Test Owner ID","coordinatesGPS":{"latitude":0.0,"longitude":0.0},"jabberID":"Test Jabber
ID","ldName":"Test ldName"},"measurements":[],"technicalCapabilities":
```

Figure 6.8.: Registration of HUAS DERs on the Registry

On the Figure 6.8 is the registered DER-Systems queried directly from the live registry, where the first entry within the red outline, with the derID: "HUAS1CHP" is the successfully registered DER-System for the CHP test. Below the CHP entry, there is a yellow outline surrounding the entry representing the simulated PV-Battery DER-system, registered with the derID: "HUAS1PVBAT", this simulation is elaborated on the subsection 6.5.4. The outlined text covers all the information registered, including a unique ID, other details about location, device specifics and capabilities.

Now that the device is registered to the server, an energy service can be posted under the derID of the device. Consequently, the flexibility offer can be updated usually with a schedule of 96 points (24 hour schedule in 15 minute intervals), here we only plan to run the device for two hours so only the first 6 values are relative for this test. Here these values are read from the IEC 61850 server running on the middleware since this is where the DER-Server stores the schedule for the availability however the communication is still done via REST.



141.22.44.23:8080/yellowpages/activepower/getAll

```
{"validFrom":null,"validTo":null,"powerCosts":0.0,"energyCosts":0.0,"futureCosts":0.0,"activationCosts":0.0,"reservationCosts":0.0,
"penaltyPrice":0.0,"timestamp":null},"type":"GENERATOR","nominalPowerkW":{"min":0.0,"max":500.0},"setspoints":
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,245.0,407.0,605.0,771.0,947.0,
1118.0,1279.0,1356.0,1491.0,1614.0,1725.0,2009.0,2106.0,2193.0,2264.0,2178.0,2221.0,2249.0,2267.0,2222.0,2218.0,2201.0,2173.0,2088.
0,2042.0,1989.0,1926.0,1811.0,1735.0,1652.0,1564.0,1538.0,1436.0,1330.0,1218.0,1177.0,1055.0,931.0,807.0,646.0,528.0,415.0,303.0,27
9.0,274.0,270.0,262.0,249.0,239.0,229.0,220.0,201.0,189.0,175.0,161.0,145.0,129.0,112.0,96.0,80.0,61.0,42.0,25.0,0.0,0.0,0.0,0.0,0.
0,0.0,0.0,0.0],"futuresAllowed":false},
{"serviceID":"HOME57PV","derID":"HOME57PV","startDate":1451606400000,"endDate":1514764800000,"predicatable":true,"dispatchable":fal
se,"reliability":1.0,"timestamp":1461838046581,"forecasts":[],"contracts":[],"unavailabilities":[],"serviceAvailability":
{"availability":{"ptus":[],"timebase":"QUARTER_H"}},"prices":
{"validFrom":null,"validTo":null,"powerCosts":0.0,"energyCosts":0.0,"futureCosts":0.0,"activationCosts":0.0,"reservationCosts":0.0,
"penaltyPrice":0.0,"timestamp":null},"type":"GENERATOR","nominalPowerkW":
{"min":0.0,"max":500.0},"setspoints":null,"futuresAllowed":false},
{"serviceID":"HUAS_CHP_LABTEST_AP1","derID":"HUAS1CHP","startDate":1451606400000,"endDate":1514764800000,"predicatable":true,"dispa
tchable":false,"reliability":1.0,"timestamp":1461838046581,"forecasts":[],"contracts":[],"unavailabilities":
[],"serviceAvailability":{"availability":{"ptus":[{"time":1506528200465,"powerkW":5.01},{"time":1506529100465,"powerkW":8.01},
{"time":1506530000465,"powerkW":0.01},{"time":1506530900465,"powerkW":16.01},{"time":1506531800465,"powerkW":10.01},
{"time":1506532700465,"powerkW":0.01},{"time":1506533600465,"powerkW":0.01},{"time":1506534500465,"powerkW":0.01},
{"time":1506535400465,"powerkW":0.01},{"time":1506536300465,"powerkW":0.01},{"time":1506537200465,"powerkW":0.01},
{"time":1506538100465,"powerkW":0.01},{"time":1506539000465,"powerkW":0.01},{"time":1506539900465,"powerkW":0.01},
{"time":1506540800465,"powerkW":0.01},{"time":1506541700465,"powerkW":0.01},{"time":1506542600465,"powerkW":0.01},
{"time":1506543500465,"powerkW":0.01},{"time":1506544400465,"powerkW":0.01},{"time":1506545300465,"powerkW":0.01},
{"time":1506546200465,"powerkW":0.01},{"time":1506547100465,"powerkW":0.01},{"time":1506548000465,"powerkW":0.01},
```

Figure 6.9.: Update of the energy service on the Registry

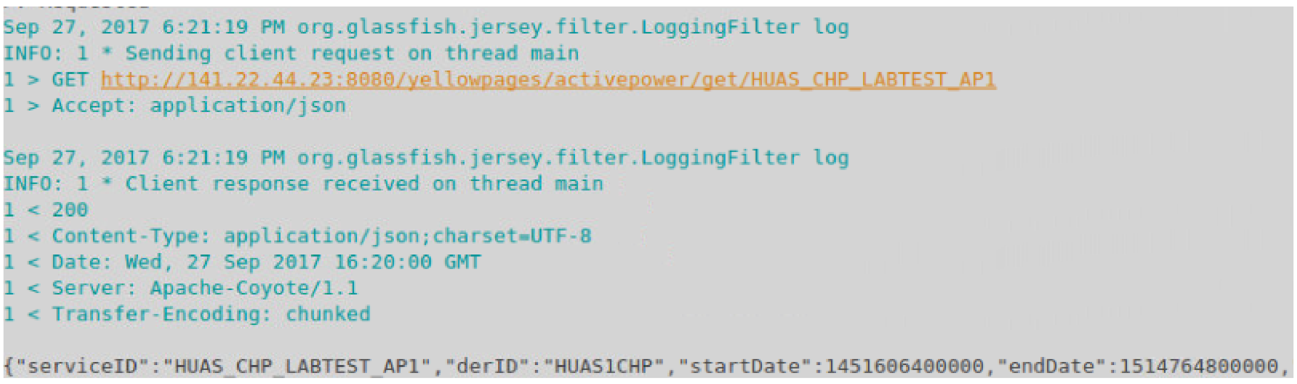
The yellow pages status shows the energy service with serviceID = "HUAS_CHP_LABTEST_API" is posted onto the list of available services, outlined in an orange box. And along with it is the schedule starting from the third line of the outlined text, under the service availability parameter, with the first power value being 5kW and goes on until 10kW then followed with zeros after the testing is done.

This shows the DER-System is successfully registered and the energy services is also properly updates the registry with the flexibility schedule. And this is the first leg of the trilateral communication tested to be working using elements of the IEC 61850 and mainly the REST API to facilitate the exchange.

6.5.2. Aggregator requests for a schedule

[E. Madaha]

This is the second leg of the final test, where an aggregator makes a bid for an energy service offer that is on the registry. This process usually involves agreeing on a contract and the registry then updates the offer to indicate that it has been booked, so as to avoid conflicting orders on the energy services. The DER-System is thus also updated with the status of the offer once it's booked, hence to be ready and prepare for the orders. But in this test only the schedule is read by the aggregator skipping through the contracts and pricing, since the goal here is to check the successful exchange of the energy service once this is done the rest can be tested and improved.



```
Sep 27, 2017 6:21:19 PM org.glassfish.jersey.filter.LoggingFilter log
INFO: 1 * Sending client request on thread main
1 > GET http://141.22.44.23:8080/yellowpages/activepower/get/HUAS_CHP_LABTEST_API
1 > Accept: application/json

Sep 27, 2017 6:21:19 PM org.glassfish.jersey.filter.LoggingFilter log
INFO: 1 * Client response received on thread main
1 < 200
1 < Content-Type: application/json;charset=UTF-8
1 < Date: Wed, 27 Sep 2017 16:20:00 GMT
1 < Server: Apache-Coyote/1.1
1 < Transfer-Encoding: chunked

{"serviceID":"HUAS_CHP_LABTEST_API","derID":"HUAS1CHP","startDate":1451606400000,"endDate":1514764800000,
```

Figure 6.10.: Console output from schedule request

The Figure 6.10 displays the java console for the Aggregator, where most of the red text is the REST Client logging the connection and the pulling of the data from the registry's server. On the last line is the is the service profile of the CHP under the ID: 'HUAS_CHP_LABTEST_API', same as the one registered on the previous step found in Figure 6.9, it's displayed all along the same line so only the first few values can be seen.

Once again this leg of the communication uses the REST API, since the registry is implemented as such. The bridge for using the IEC 61850 server (Java Python Bridge) did not suffice for this use, hence using the REST alternative same reasons as for the previous step. Even though the complete middleware communication was intended to run with the IEC 61850 as explained in the system description; some compromises were made as the implementation failed to meet the set expectation.

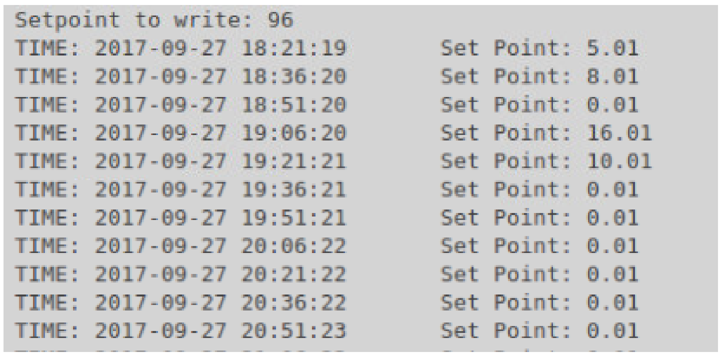
6.5.3. Aggregator sends the setpoints

[E. Madaha]

Finally, the test is completed when the aggregator can call on the DER-System to generate the electricity that has been offered, and the DER to provide as promised. This is the last leg of the trilateral communication; when the circle is completed since it starts with the DER giving an energy service offer and is completed when the DER delivers on the services offer it registered.

Results from the CHP

[E. Madaha]



```
Setpoint to write: 96
TIME: 2017-09-27 18:21:19       Set Point: 5.01
TIME: 2017-09-27 18:36:20       Set Point: 8.01
TIME: 2017-09-27 18:51:20       Set Point: 0.01
TIME: 2017-09-27 19:06:20       Set Point: 16.01
TIME: 2017-09-27 19:21:21       Set Point: 10.01
TIME: 2017-09-27 19:36:21       Set Point: 0.01
TIME: 2017-09-27 19:51:21       Set Point: 0.01
TIME: 2017-09-27 20:06:22       Set Point: 0.01
TIME: 2017-09-27 20:21:22       Set Point: 0.01
TIME: 2017-09-27 20:36:22       Set Point: 0.01
TIME: 2017-09-27 20:51:23       Set Point: 0.01
```

Figure 6.11.: Console output from sending setpoints

This Figure 6.11 shows the setpoints being sent over in 15 minute intervals, and by looking at the actual values they match the 'powerkW' values from the first step on Figure 6.9. The DER-System receives the order by writing to the respective parameter on the IEC 61850 server which holds the responsibility for being a setpoint of the power generation. This part maintains the usage of the IEC 61850 standard and uses the middleware in the way intended; since it was one of the first working parts of the Java Python Bridge.

The next step is the forwarding the setpoint over to DER-System to command the system to generate the demanded electrical power. And this is facilitated by the Smart Gateway which runs on Node-RED, the final implemented flows for this operation is displayed on Figure 5.1 the setpoint is read into the system here from the IEC 61850 server, and on the flow in Figure 5.2 the CHP is set to run on the given setpoint.

Eventually the CHP starts and gradually achieves the value that the aggregator has set for this run, here onwards a physical result can be observed from the test, since this is the part where a real CHP generator is running (not just a software simulation).

To make this test while following all the OS4ES procedures for providing energy services, such as running setpoints in 15 minute intervals. Some preparation had to be done so as to fit the conditions for the devices, CHP specifically in this case, where the heating system had to be prepared before having such a long run on the CHP generator. So the heat storages had to be left with very low temperatures to allow the long term running of the CHP without any bad side effects such as overheating.

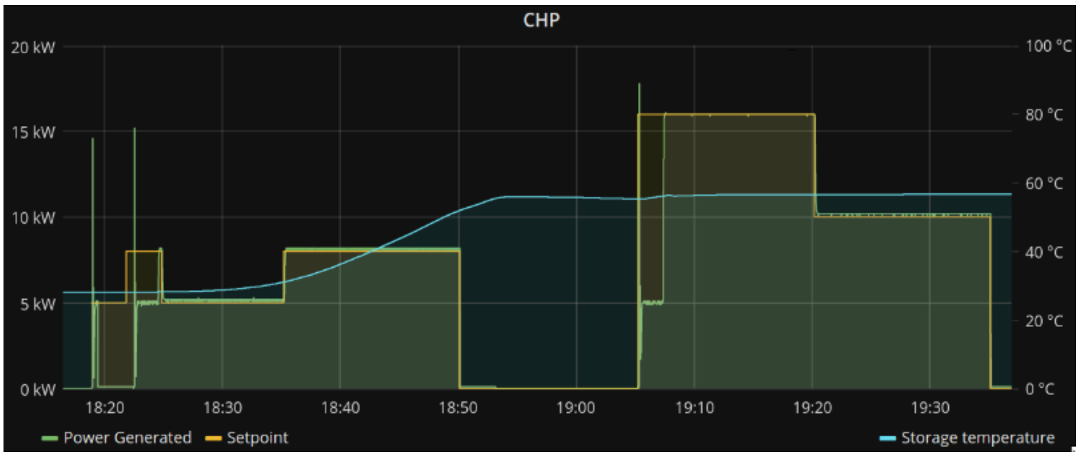


Figure 6.12.: CHP output after receiving setpoints

The resulting run of the CHP is shown on the Figure 6.12 which also matches the sequence provided by the aggregator as also seen in the previous Figure 6.11, with the setpoints : 5 , 8 , 0 , 16 , 10 , 0 , 0 ,... the zeros just representing the following off time after the test is over.

The yellow line on the graph represents the setpoints being sent from the aggregator, and as observed from the figure they seem to be almost fully synchronized. It appears that the delay for the CHP, that was recorded and measured in the preliminary tests, has been significantly reduced or perhaps the different systems that records each of the timestamps might have had an offset in the system clock. But further testing can be conducted on this matter, to get a vivid picture about the CHP's response time and also the communication path of the setpoints can be checked to see if there is any source of procedural delay.

The beginning of the graph shows the CHP starting and immediately turning off again, this happened because the test was started with a shorter interval than 15 minutes and had to be reset again to fit the test description. Also an additional comment on the first setpoint's time slot (on the 5kW); there was an interference in between and a different value of 8kW was sent and briefly then returned back to 5kW.And This was due to a node that held the value of the previous run, which was released late into this run of the CHP, but this could be avoided by resetting the flows to clear all states and stored values before a new run. The node being referred to here is the one called "Setpoint Delay" which can be seen in this Figure 5.2 under the red outline.

The temperature behaviour from the storage can be evaluated from the Figure 6.12, given a steady rise once the CHP has started running. For this test the heat storages used had to be prepared by leaving them without heating over a period of time, so as to avoid exceeding any high temperature thresholds that could turn off the CHP. Since every setpoint runs on 15 minute intervals storage was heating up quite fast given that most of the setpoints kept the

CHP running and at the given pace the storage wouldn’t be able to take another 2 hour test. So from this graph the effect on the heat storage from the grid demand can be seen, and if there is no or low heat demand on the facility it keeps the operational time frame for the CHP under 4 hours (double the testing time).

And with this run can finally confirm that the end to end communication of the OS4ES system has been accomplished, also the running of a DER-System has been executed properly without any significant problems.

6.5.4. PV & Battery Simulation

[G. Mckoy]

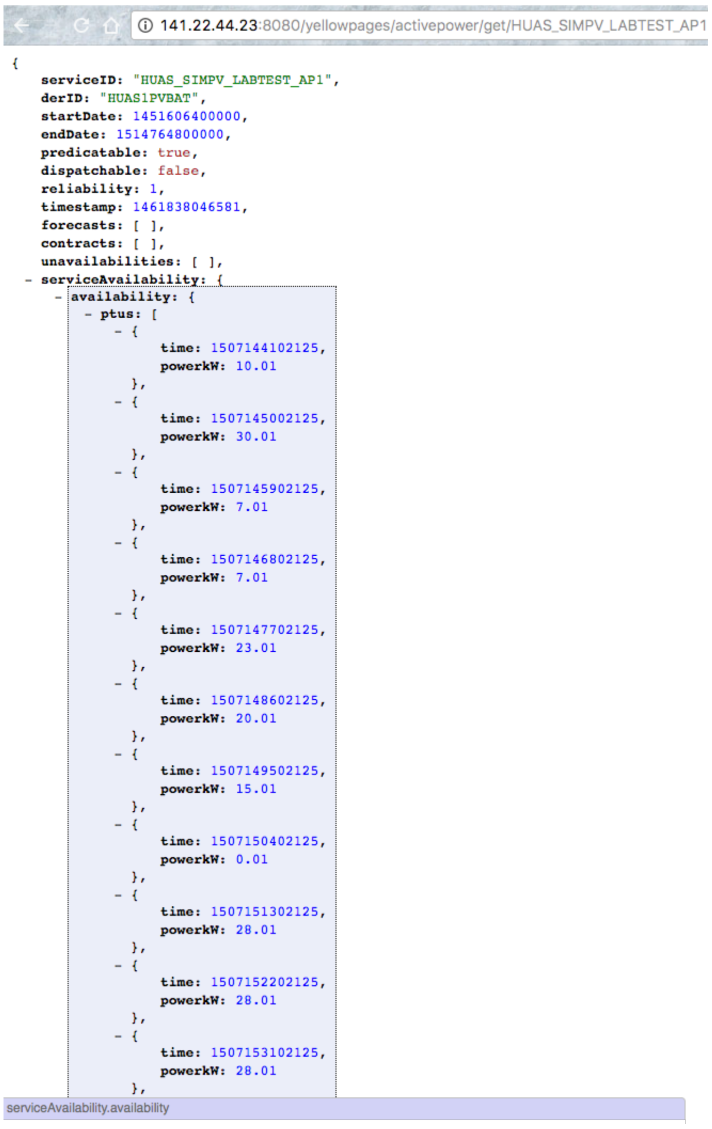
The subsection describes and discusses the simulation process of the PV-Battery model. It gives a description of the step by step process involved in running the simulation and the reason behind the various approaches used.

PV-Battery Registration

[G. Mckoy]

The whitepages registration that registers a DER device via the REST API was done in advance and can be seen in Figure 6.8 along with the CHP. The section of interest for the PV-Battery simulation is surrounded by the orange box.

Figure 6.13 shows a screenshot of the yellowpages registration of the PV-Battery simulation model. The *serviceID* and *derID* can be seen at the top of the figure and the service availability towards the end. Prior to executing the REST command seen in the search field, the service availability was updated by the registry. The update was done by reading the availability offered by the DER-Server and writing it to the ptus array of the availability object.



```
141.22.44.23:8080/yellowpages/activepower/get/HUAS_SIMPV_LABTEST_AP1
```

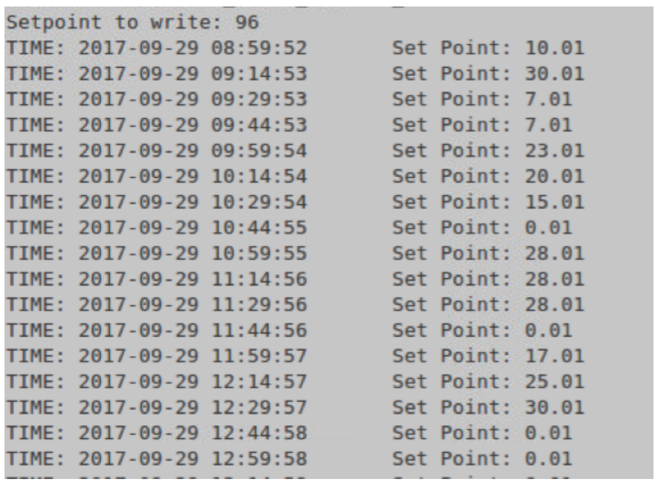
```
{
    serviceID: "HUAS_SIMPV_LABTEST_AP1",
    derID: "HUAS1PVBAT",
    startDate: 1451606400000,
    endDate: 1514764800000,
    predicatable: true,
    dispatchable: false,
    reliability: 1,
    timestamp: 1461838046581,
    forecasts: [ ],
    contracts: [ ],
    unavailabilities: [ ],
  - serviceAvailability: {
      - availability: {
          - ptus: [
              - {
                    time: 1507144102125,
                    powerkW: 10.01
                },
              - {
                    time: 1507145002125,
                    powerkW: 30.01
                },
              - {
                    time: 1507145902125,
                    powerkW: 7.01
                },
              - {
                    time: 1507146802125,
                    powerkW: 7.01
                },
              - {
                    time: 1507147702125,
                    powerkW: 23.01
                },
              - {
                    time: 1507148602125,
                    powerkW: 20.01
                },
              - {
                    time: 1507149502125,
                    powerkW: 15.01
                },
              - {
                    time: 1507150402125,
                    powerkW: 0.01
                },
              - {
                    time: 1507151302125,
                    powerkW: 28.01
                },
              - {
                    time: 1507152202125,
                    powerkW: 28.01
                },
              - {
                    time: 1507153102125,
                    powerkW: 28.01
                },
```

Figure 6.13.: PV yellowpages snippet

Aggregation of PV-Battery

[G. Mckoy]

To control the DER-unit, the Aggregator requests the flexibility of the PV-Battery simulation model from the registry using the *serviceID*. The Aggregator then extracts the availability schedule stored in the *ptus* array and loops over the values. The values are then sent directly from the Aggregator to the DER-Server where the request is processed. Figure 6.14 shows the setpoints and the corresponding timestamps when each setpoint was transferred to the DER-Server.



```
Setpoint to write: 96
TIME: 2017-09-29 08:59:52       Set Point: 10.01
TIME: 2017-09-29 09:14:53       Set Point: 30.01
TIME: 2017-09-29 09:29:53       Set Point: 7.01
TIME: 2017-09-29 09:44:53       Set Point: 7.01
TIME: 2017-09-29 09:59:54       Set Point: 23.01
TIME: 2017-09-29 10:14:54       Set Point: 20.01
TIME: 2017-09-29 10:29:54       Set Point: 15.01
TIME: 2017-09-29 10:44:55       Set Point: 0.01
TIME: 2017-09-29 10:59:55       Set Point: 28.01
TIME: 2017-09-29 11:14:56       Set Point: 28.01
TIME: 2017-09-29 11:29:56       Set Point: 28.01
TIME: 2017-09-29 11:44:56       Set Point: 0.01
TIME: 2017-09-29 11:59:57       Set Point: 17.01
TIME: 2017-09-29 12:14:57       Set Point: 25.01
TIME: 2017-09-29 12:29:57       Set Point: 30.01
TIME: 2017-09-29 12:44:58       Set Point: 0.01
TIME: 2017-09-29 12:59:58       Set Point: 0.01
```

Figure 6.14.: PV-Battery setpoints sent from the Aggregator.

The simulation was done using the kilowatt values produced by the PV on the 29th of September 2017, between the hours of 9:00 and 13:00. This time period was chosen for the test as a result of the weather conditions being fairly sunny, which implied the PV was generating reasonably high kilowatt values.

Booking of PV-Battry Service from DER-Server

[G. Mckoy]

The Figure 6.15 illustrates the setpoints received by the DER-System from the aggregator, the output of the converter, the output of the battery and the total output of the battery and converter together. The converter values shown in the figure, are the values received directly from the PV. This is to simulate the behaviour of the converter receiving a kilowatt value from the PV and charging the battery with it or outputting it directly to the grid. The converter simulation model tries to supply the demand with values directly from the PV first and uses the battery to supply the difference.

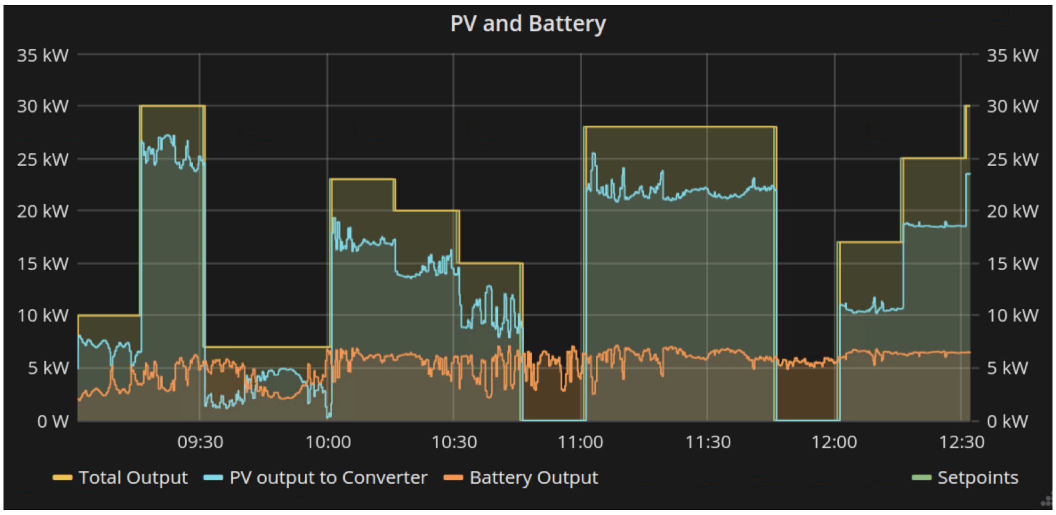


Figure 6.15.: PV simulation lab test

The overall implementation and execution of the end-to-end test was of the most straight forward and pragmatic approach. The aggregator in particular, ignores the time values and simply loops over the availability and sets the setpoints in the DER-Server. This approach was chosen as it was easy to implement with the limited time available. This approach does not affect the results because the test environment was controlled and the battery was set to its maximum capacity. As a result, neither time nor the sequence in which the setpoints are sent would lead to the battery not being able to supply its initial offer. Of more interest is that the setpoints set by the aggregator are equal to the output of the simulated DER-unit. By analysing Figure 6.15, it can be observed that the simulated model behaves as expected. The yellow and green lines on the graph indicates the total output and the setpoints respectively.

While the graph shows the intended behaviour of the simulated battery it is important to keep in mind that the aim of this test is not to prove that the battery can produce the correct values but instead that a services can be offered, registered aggregated and booked. This process has been proven with the results of this simulation.

6.6. Requirements Outlook

[E. Madaha]

As the results of the lab test and simulation are laid out with a successful outcome, for the rest of the work it would be best to revisit the earlier defined requirements to give fully scaled outlook of the work done in this thesis.

6.6.1. Functional Requirements

[E. Madaha]

End to end test, this part is well defined in the above section 6.5 elaborating each and every step from the DER-System registration to the registry to the final end which is the DER-System providing the requested energy from the Aggregator.

Integration of the DER-System into middleware, for this some software to define a DER-System had to be designed and developed, in compliance with the rest of the OS4ES software modules, on 5.3 DER-System under the implementation chapter there is an in-depth elaboration. Further on the DER-System is then integrated onto the Java Python Bridge as described in the 5.2 Middleware Integration section of the Implementaion chapter. These parts were implemented not to the best possible extent as some features were not fully operational due to some prior implementations that were incoherent to the project's definition. But the main functional requirement was fulfilled and also the implementation done is flexible for future adaptations of more features.

6.6.2. Non-Functional Requirements

[E. Madaha]

Testing, this was done amidst working on the middleware integration as it was inherently needed to test both versions of the Java Python Bridge and most of this part was under the context of the same work in the Middleware Integration subsection(5.2).

Data recording, as a smart grid concept, the key part of having a data layer added to the grid network is getting all the necessary data from each datapoint. Thus at most of the flow transitions: Smart Gateway » CHP generator in figure 5.2 or from DER-System » Aggregator like in figure 5.1 have connections to the database logging all the datapoints.

Smart Gateway, this was just a suggestion that the DER-System should use a smart interface to manage all the DER units operating within its network. Through Node-RED it has manifested quite well into a real smart DER management system where it is easier to add or remove DER units just by adding/removing a node with their network address and also to replace a simulated device with a real device has been simplified by redirecting flows to their respective nodes. This is well elaborated under the 5.3.1 Smart Gateway in the implementation chapter.

Simulation, DER devices were both simulated but the battery served well as to replace the defunct real device that wasn't operational for the final lab test, and the results from this simulation are vividly described in the above section 6.5 prior to this one. While description of it's implementation are well defined in the 5.1 subsection Simulated DER Devices.

Compatibility, this was not explicitly defined throughout the thesis but the code used was compatible to the rest of the project's standards as usage of Java 7 and Python 3. And also most of the communication uses the MQTT protocol which is the common standard for most of the DER related interfaces. And the others such as REST and IEC 61850 was used in the respective domains.

6.6.3. Overview of achieved requirements

[E. Madaha]

All the defined requirements of this work were fulfilled to their extent and the definitions were applied as specified. Even though the final result does not run cohesively with all integrated modules of the project, this is the natural outcome of inhomogeneous code implementation. A final comment is that the feasibility of this system has been checked and proven to be feasible, but the perfection and cohesion of this system now relies upon the system's deployment, as this project only stands for research into the possibilities of future energy markets.

7. Conclusion

[G. Mckoy]

The process of integrating different energy resource into the OS4ES system and having an end-to-end communication test of the System was carried out at the Hamburg University of Applied Sciences Energy Campus, Bergedorf. The successful integration of the DER-System into the OS4ES verifies the concept of achieving a sustainable source of energy by grouping smaller suppliers together. The results also contribute to the practicality of developing such a system, through multiple organisations, as well as highlights the challenges associated with such a development process.

The individual software components from the previous work packages were tested separately and in small blocks. These components include the Middleware (Java Python Bridge), the client connection used by the registry and aggregator and the controls of the individual devices. This was done to ensure that each block operates as expected within the final system. With a few modifications, these individual components were brought together to work as one unit. This unit served as the final platform for conducting the lab test and the end-to-end communication tests.

The heat storage and CHP components were integrated into the OS4ES system and serve as the foundation for concluding that the integration process was a success. The CHP was connected to the OS4ES through the implemented DER-System using the Middleware. This connection allowed for the registry and the aggregator to see and interact with the physical devices connected to the DER-System. The CHP integration test process involved turning on and off the device and setting the rate at which it operated. Performing these tasks from the Aggregator based on information supplied to the registry from the DER-System fulfilled the end-to-end communication test requirements.

On the other hand, the battery was not successfully integrated into the OS4ES due to technical and manufacture faults. These faults prevented the devices from being controlled by the DER-system and by extension the OS4ES system. Based on the construct of the system, this block was removed and replaced with a simulation model of the battery which works with the real PV system. The model allowed for a full system test to be carried out for the battery and verifies that the remainder of the system works as expected. It is expected that once the problems with the battery have been resolved, the simulation model can be replaced.

Though the implementation is currently designed for specific devices, the integration process for a new physical device will require very little effort; this is a result of the system being designed in functional blocks. With this in mind the possibilities of connecting an energy source to the OS4ES poses very little complications restricted to the block associated with the specific device.

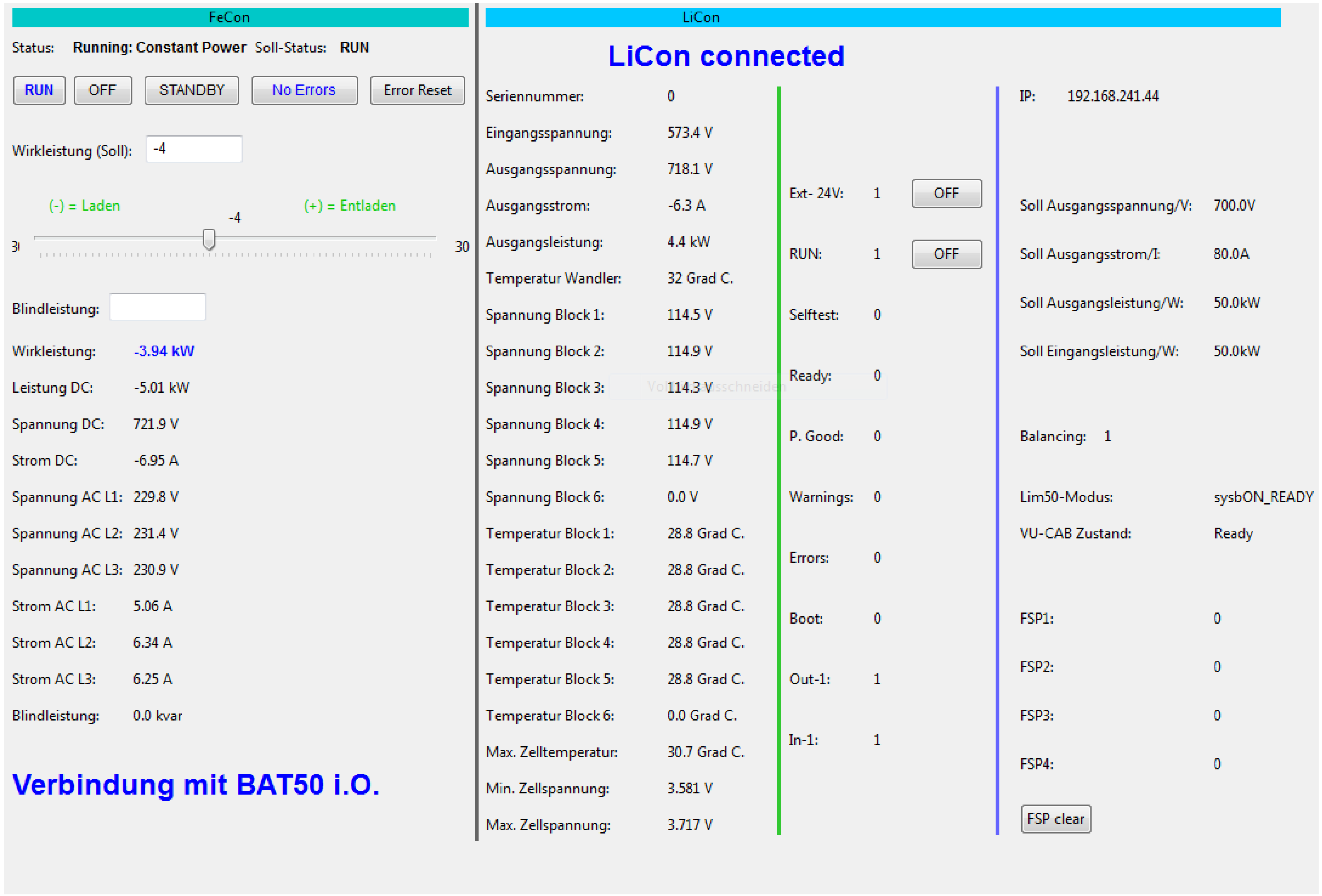
A. Physical Devices

A.1. Battery and Converter

A.1.1. User Interface of the Battery and Converter

The controls shown in A.1 on the following page, are used to charge and discharge the battery. The slider bar discharges the battery if the value is below zero and charges it if the value is above zero. If the needle is positioned at zero the converter is in a idle state; neither charging nor discharging the battery.

The battery is currently able to communicate with only one device at a time over its modbus communication channel. This implies that either the converter or a computer can be connected. The battery gets into an error state from time to time and needs to be rest or restarted fom a direct connection to a computer. This process requires one to physical unplug the RJ45 cable (ethernet) from the convert and then to the computer to solve the problem.



Status: **Running: Constant Power** Soll-Status: **RUN**

No Errors

Wirkleistung (Soll): -4

(-) = Laden -4 (+) = Entladen

3 30

Blindleistung:

Wirkleistung:	-3.94 kW
Leistung DC:	-5.01 kW
Spannung DC:	721.9 V
Strom DC:	-6.95 A
Spannung AC L1:	229.8 V
Spannung AC L2:	231.4 V
Spannung AC L3:	230.9 V
Strom AC L1:	5.06 A
Strom AC L2:	6.34 A
Strom AC L3:	6.25 A
Blindleistung:	0.0 kvar

Verbindung mit BAT50 i.O.

LiCon connected

Seriennummer:	0
Eingangsspannung:	573.4 V
Ausgangsspannung:	718.1 V
Ausgangsstrom:	-6.3 A
Ausgangsleistung:	4.4 kW
Temperatur Wandler:	32 Grad C.
Spannung Block 1:	114.5 V
Spannung Block 2:	114.9 V
Spannung Block 3:	114.3 V
Spannung Block 4:	114.9 V
Spannung Block 5:	114.7 V
Spannung Block 6:	0.0 V
Temperatur Block 1:	28.8 Grad C.
Temperatur Block 2:	28.8 Grad C.
Temperatur Block 3:	28.8 Grad C.
Temperatur Block 4:	28.8 Grad C.
Temperatur Block 5:	28.8 Grad C.
Temperatur Block 6:	0.0 Grad C.
Max. Zelltemperatur:	30.7 Grad C.
Min. Zellspannung:	3.581 V
Max. Zellspannung:	3.717 V

Ext- 24V:	1
RUN:	1
Selftest:	0
Ready:	0
P. Good:	0
Warnings:	0
Errors:	0
Boot:	0
Out-1:	1
In-1:	1

IP:	192.168.241.44
Soll Ausgangsspannung/V:	700.0V
Soll Ausgangsstrom/I:	80.0A
Soll Ausgangsleistung/W:	50.0kW
Soll Eingangsleistung/W:	50.0kW
Balancing:	1
Lim50-Modus:	sysbON_READY
VU-CAB Zustand:	Ready
FSP1:	0
FSP2:	0
FSP3:	0
FSP4:	0

Figure A.1.: PV simulation lab test

A.1.2. Battery configuration yml file

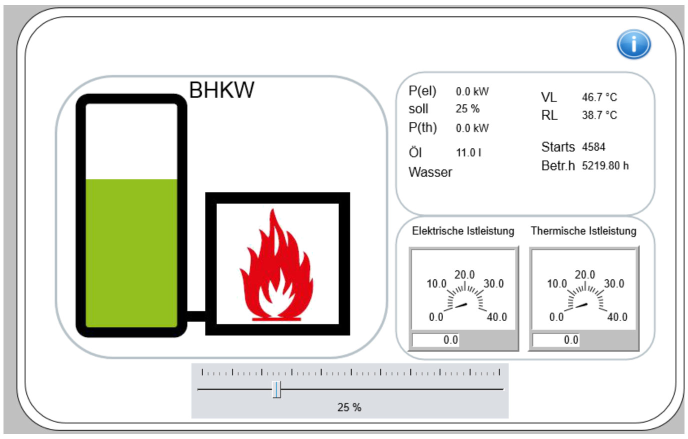
The listing A.1.2 shows the modbus configuration file for the battery. The file contains the name of the Modbus communication data points as well as the the addresses, ports and functionality of the protocol used.

```
# ----------------------------------------------------------------------------#
# Commissioning File
# ----------------------------------------------------------------------------#
# Manufacturer: PowerInovation
# Device: VU-CAB52
# Copyright: Energie-Campus
# Contact:
# ----------------------------------------------------------------------------#
# Source Interface Definition - Modbus TCP
# ----------------------------------------------------------------------------#
source:
  driver: modbus
  connection:
    protocol: modbus.tcp
    host: 192.168.241.44
    port: 502
    unitId: 1
  defaults:
    operation: subscribe
    interval: 5000 # ms
    length: 1
    fc: 4
# ----------------------------------------------------------------------------#
# Target Interface Definition - MQTT (Cybus Connectware Broker)
# ----------------------------------------------------------------------------#
target:
  driver: mqtt
  defaults:
    operation: write
    topicPrefix: io/cybus/energie-campus/raw/batterie
# ----------------------------------------------------------------------------#
# Device Datapoint Mappings
# 1. Measurements
# ----------------------------------------------------------------------------#
mappings:
# ----------------------------------------------------------------------------#
# 1. Measurements
# see corresponding documents
#   * manual
#   * FeCon GmbH Interface Description Modbus TCP and EtherCAT
# ----------------------------------------------------------------------------#
- source:   # Grid Voltage L1(16bit UINT fc4); 2315=231,5 V
    address: 0
  target:
    topic: spannung/l1
- source:   # Grid Voltage L2(16bit UINT fc4); 2315=231,5 V
    address: 1
  target:
    topic: spannung/l2
- source:   # Grid Voltage L3 (16bit UINT fc4); 2315=231,5 V
    address: 2
  target:
    topic: spannung/l3
- source:   # Grid Current L1(16bit UINT fc4); 1234=12,34 A
    address: 100
  target:
    topic: strom/l1
- source:   # Grid Current L2(16bit UINT fc4); 1234=12,34 A
    address: 101
  target:
    topic: strom/l2
- source:   # Grid Current L3 (16bit UINT fc4); 1234=12,34 A
    address: 102
  target:
    topic: strom/l3
- source:   # Active Power(16bit INT fc4); −895=−8,95 kW
    address: 2000
  target:
    topic: wirkleistung
- source:   # Reactive Power(16bit INT fc4); −895=−8,95 kvar
    address: 2002
  target:
    topic: blindleistung
- source:   # DC Power(16bit INT fc4); −895=−8,95 kW
    address: 2003
  target:
    topic: leistung/DC
- source:   # Voltage Vdc(16bit INT fc4); −1234=123,4 V
    address: 200
  target:
    topic: strom/DC
- source:   # Current Idc(16bit INT fc4); 1234=12,34 A
    address: 201
  target:
    topic: spannung/DC
- source:   # Active Power Set Point(16bit INT fc6); −400=−4,00 kW (BAT50)
    operation: write
    fc: 16
    address: 5801
  target:
    operation: subscribe
    topic: wirkleistung/set
- source:   # Reactive Power Set Point(16bit INT fc6); −500=−5,00 kVAr (BAT50)
    operation: write
    fc: 16
    address: 5801
  target:
    operation: subscribe
    topic: blindleistung/setcaption
```

A.2. CHP and Heat storage

A.2.1. user interface of the CHP system

Figure A.2 shows the UI of the CHP. The CHP is kept at a minimum state of 25% of its operating capacity and operates at a maximum of 80% to prolong the life time of the device. The Sub-Figure *a* show the CHP in an off state neither producing heat nor electricity. The Sub-Figure *b* shows the device operating at its maximum allowed capacity of 80%. The gold ring around the UI indicates that the device is currently active while the current state can be read from the legend on the right. The slider at the bottom can be used to adjust the operating output in percentage of the device.


BHKW

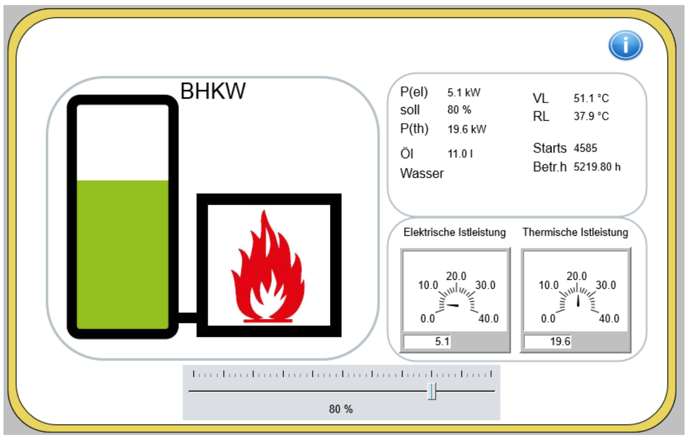
P(el) 0.0 kW
soll 25 %
P(th) 0.0 kW
Öl 11.0 l
Wasser

VL 46.7 °C
RL 38.7 °C
Starts 4584
Betr.h 5219.80 h

Elektrische Istleistung 0.0
Thermische Istleistung 0.0

25 %

(a) Inactive state.


BHKW

P(el) 5.1 kW
soll 80 %
P(th) 19.6 kW
Öl 11.0 l
Wasser

VL 51.1 °C
RL 37.9 °C
Starts 4585
Betr.h 5219.80 h

Elektrische Istleistung 5.1
Thermische Istleistung 19.6

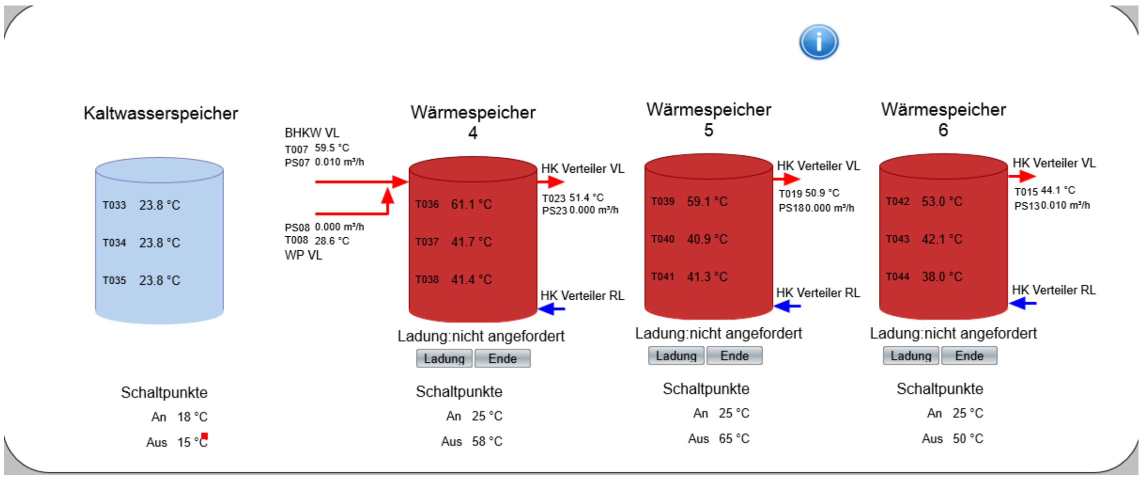
80 %

(b) Active state

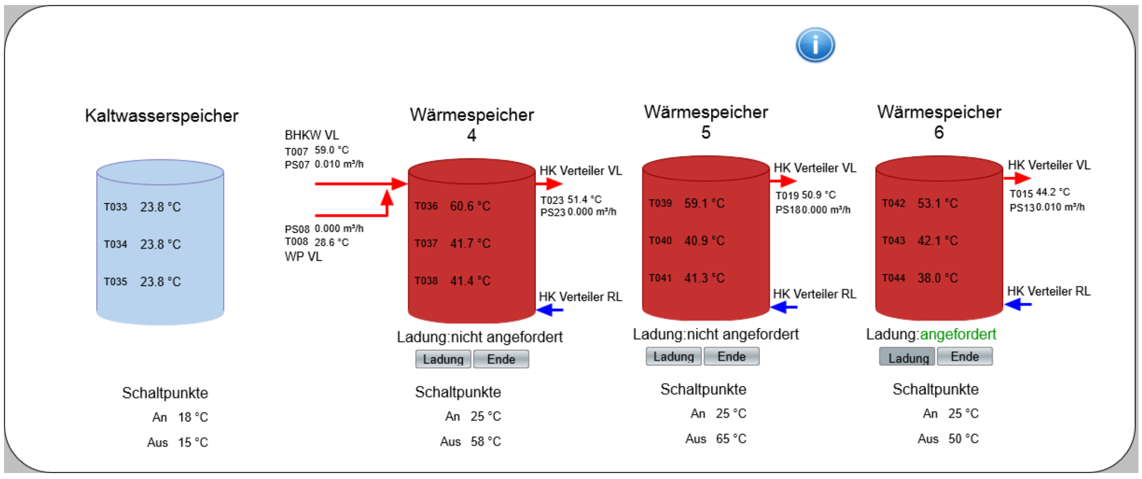
Figure A.2.: CHP in active and inactive state.

A.2.2. User Interface of the Heat storage

Figure A.3 shows the UI of the heat storage devices four, five and six. Each device shows the current temperature of the water at three levels: bottom, middle and top. The devices can can be tuned on or off using the buttons ladung and ende below the image of each unit. The Sub-Figure *a* show all three devices in an inactive state while the Sub-Figure *b* shows storage six active indicated by angefordert text in green.



(a) Inactive state.



(b) Active state

Figure A.3.: Heat storage in active and inactive state.

A.2.3. Plot of Heat storage and CHP

Figure A.4 shows the heat storage for devices four, five and six. On the left scale is the temperature of the water in each storage.The legends on the right of each graph, shows the water temperature at different levels of the tank; bottom, middle and top. On the right scale is the kilowatt output of the CHP. The kilowatt output is same for all three heat storage device since they are connected to one CHP. The Speicherldung angefordert is a signal that is triggered high and then low again to turn the CHP on or of. Any of the three system can be used to trigger this signal.

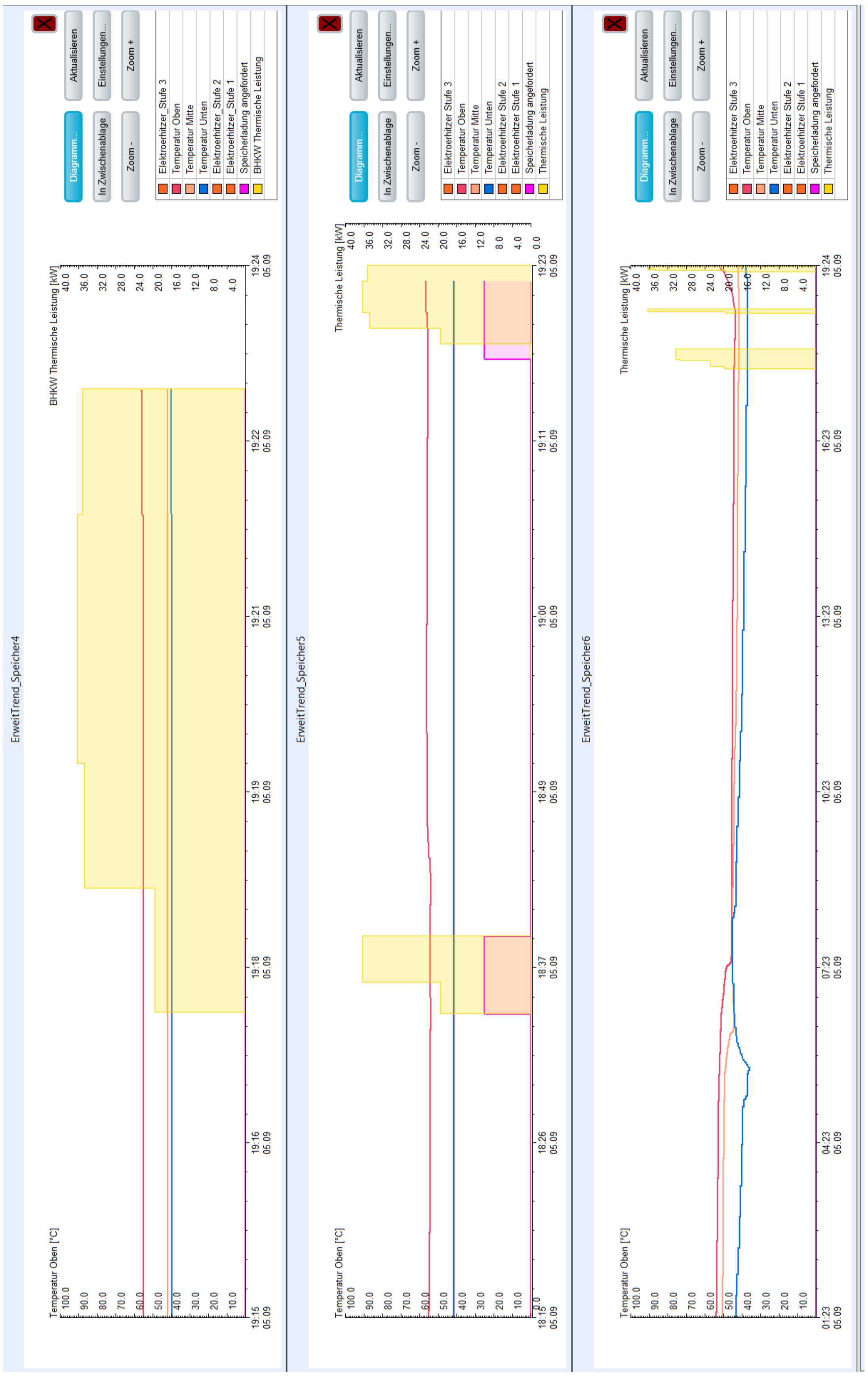


Figure A.4.: Plot showing the cheat storage and chp

A.2.4. CHP configuration yml file

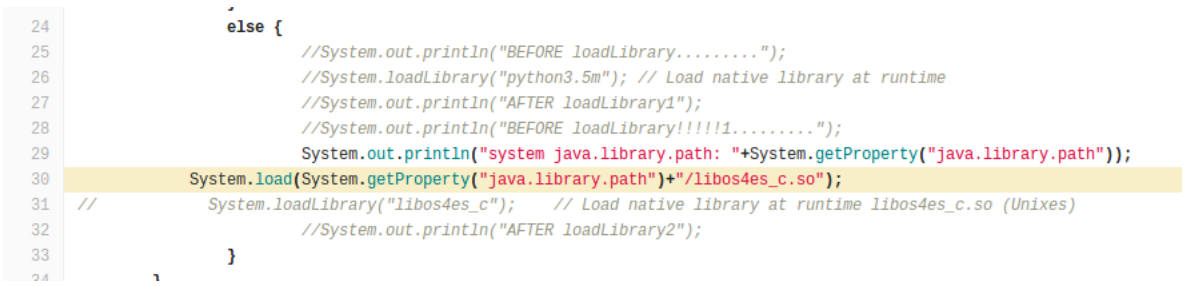
The listing A.2.4 shows the configuration file for the CHP. The file contains the names of the Modbus and MQTT communication data points as well as the addresses, ports and functionality of the protocol used.

```
source:
  driver: bacnet
  connection:
    protocol: bacnet-ip
    localInterface: eth1
    localPort: 47808
    deviceInstance: 12
    deviceAddress: 192.168.241.241
  defaults:
    operation: subscribe
    interval: 5000
    property: present-value
    priority: 7
target:
  driver: mqtt
  defaults:
    operation: write
    topicPrefix: io/cybus/energie-campus/raw/bhkw
mappings:
- source:
    description: Elektrische Istleistung des BHKWs [kW]
    objectType: analog-input
    objectInstance: 131
  target:
    topic: leistung/elektrisch
- source:
    description: Thermische Istleistung des BHKWs [kW]
    objectType: analog-input
    objectInstance: 132
  target:
    topic: leistung/thermisch
- source:
    description: Vorlauftemperatur des BHKWs [ $^{\circ}$ C]
    objectType: analog-input
    objectInstance: 133
  target:
    topic: vorlauf/temperatur
- source:
    description: Reucklauftemperatur des BHKWs [ $^{\circ}$ C]
    objectType: analog-input
    objectInstance: 134
  target:
    topic: ruecklauf/temperatur
- source:
    description: Verbrauchtes Oel des BHKWs [l]
    objectType: analog-input
    objectInstance: 147
  target:
    topic: oelcaption
```

B. Implementation

B.0.1. Problems faced while setting up Java Python Bridge

Setting up the Java Python Bridge on a Linux OS had several complications. One such problem is with the library paths used. The implementation assumes that the library is located on the users PC. Though this problem is small it does not allow portability of the program and would require the user to configure the bridge for each computer. This error occurred in one of the core client files, *<Project Path>src/gr/hypertech/os4es/core/Os4esIec61850ClientBridge.java*, which is needed to run the bridge. The source of th error is shown in the code snippet Figure B.1. The correct solution was implemented using the relative path and should be adapted in other sections of the bridge that requires a path to be specified.



```
24                  else {
25                          //System.out.println("BEFORE loadLibrary.........");
26                          //System.loadLibrary("python3.5m"); // Load native library at runtime
27                          //System.out.println("AFTER loadLibrary1");
28                          //System.out.println("BEFORE loadLibrary!!!!!1.........");
29                          System.out.println("system java.library.path: "+System.getProperty("java.library.path"));
30              System.load(System.getProperty("java.library.path")+"/libos4es_c.so");
31  //            System.loadLibrary("libos4es_c");    // Load native library at runtime libos4es_c.so (Unixes)
32                          //System.out.println("AFTER loadLibrary2");
33                  }
```

Figure B.1.: Java Code Snippet causing the path error

Figure B.1, shows the highlighted line 30 which causes an error due to a broken path to a library that the system can not find.

Correct implementation:

```
System.load(System.getProperty("user.dir")+"/libos4es_c.so");
```

B.1. Battery Simulation Model

Listing B.1 shows the function used to execute the battery simulation.

```
    try:
        # Create the server
        server = modbus_tcp.TcpServer()
        print ("running battery")
        print ("enter 'quit' for closing the server")

        server.start()

        slave_1 = server.add_slave(1)
        slave_2 = server.add_slave(2)

        slave_1.add_block('output', cst.HOLDING_REGISTERS, 0, 100)
        slave_2.add_block('input', cst.HOLDING_REGISTERS, 0, 100)

        old_input_values = slave_2.get_values('input', 0, 7)

        quit_server = [3, 3, 5, 5, 5]
        old_battery = get_battery_from_file(FILE_NAME)
        print_status(old_battery)

        value = [0, 0, 0, 0, 0]
        battery = get_battery_from_file(FILE_NAME)

        while True:
            temp_pv_output = get_convert_state_from_file()
            input_values = slave_2.get_values('input', 0, 7)

            #setting status values to be used
            simulation = temp_pv_output['SIMULATION']
            charge_rate = temp_pv_output["CHARGE_RATE"]
            discharge_rate = temp_pv_output["DISCHARGE_RATE"]
            #(input_values[5] << 8) | input_values[6]
            charge_after_consumption = charge_rate - discharge_rate

            if charge_after_consumption < 0:
                battery['charging_status'] = False

            elif charge_after_consumption == 0:
                battery['charging_status'] = None

            else:
                battery['charging_status'] = True

            adjust_charge(battery, charge_after_consumption)

            time.sleep(SLEEP_TIME)

            battery = get_battery_from_file(FILE_NAME)
            value[0] = int(battery['charged_value'] * 100) / battery['capacity']



            value[3] =  (abs(charge_rate) >> 8) & 0x00FF
            value[4] =  abs(charge_rate)  & 0x00FF

            if charge_after_consumption > 0:
                value[1] =  0
                value[2] =  0

            else:
                value[1] = (abs(charge_after_consumption) >> 8) & 0x00FF
                value[2] = abs(charge_after_consumption)  & 0x00FF

            slave_1.set_values('output', 0, value )

            if input_values[0] != old_input_values[0]:
                old_input_values = input_values
                print ('PV Source: %s' % simulation)
                print ('Charging: %s' % battery['charging_status'])
                print ('Chrge status(%): %s' % value)
                print ('Chrge value: %s' % charge_after_consumption)

            if battery['charging_status'] != old_battery['charging_status']:
                old_battery = battery
                print_status(battery)

            if input_values == quit_server:
                sys.stdout.write('Modbus server disconnected.\r\n')
                break
    finally:
        server.stop()caption
```

References

[1] Agile Alliance. Behavior driven development (BDD). [Online] `https://www.agilealliance.org/glossary/bdd`. last accessed: 07/10/2017.

[2] ARPA-E. Distributed power flow control. [Online] `https://arpa-e.energy.gov/?q=slick-sheet-project/distributed-power-flow-control`. last accessed: 07/10/2017.

[3] A. Banks and R. Gupta. *MQTT Version 3.1.1 OASIS Standard*. OASIS, Burlington, Massachusetts, 10 2014.

[4] L. T. Berger and K. Iniewski. *Smart Grid - Applicacions, Communications and Security*. John Wiley and Sons, 2012.

[5] Cucumber.io. Gherkin. [Online] `https://github.com/cucumber/cucumber/wiki/Gherkin`. last accessed: 07/10/2017.

[6] R. de Diego, J.-F. Martinez, J. Rodriguez-Molina, and A. Cuerva. A semantic middleware architecture focused on data and heterogeneity management within the smart grid. *Energies*, 7(9):5953–5994, 2014.

[7] T. Dethlefs, W. Renz, A. Schröder, J. Benze, A. Lang, and A. Papanikolaou. Open system for energy services (os4es): An eu-funded research project to establish a non-discriminatory, multivendor-capability service delivery platform for smart grid services. In *2015 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pages 593–598, 2015.

[8] Enel S.p.A. Enel Distribuzione: Italy's first Smart Grid in Isernia. [Press release] `https://www.enel.com/content/dam/enel-com/pressrelease/porting_pressrelease/1648124-2_PDF-1.pdf`, 2011.

[9] ENERGY.GOV(US). Recovery act: Smart grid investment grant (SGIG) program. [Online] `https://energy.gov/oe/information-center/recovery-act-smart-grid-investment-grant-sgig-program`. last accessed: 07/10/2017.

[10] EPA.org. What is CHP. [Online] `https://www.epa.gov/chp/what-chp`. last accessed: 07/10/2017.

[11] C. Ghar. Ubuntu 12.04 ipv4 nat gateway and dhcp server. [Online] `https://codeghar.wordpress.com/2012/05/02/ubuntu-12-04-ipv4-nat-gateway-and-dhcp-server/`.

[12] Y. F. Huang, S. Werner, J. Huang, N. Kashyap, and V. Gupta. State estimation in electric power grids: Meeting new challenges presented by the requirements of the future grid. *IEEE Signal Processing Magazine*, 29(5):33–43, Sept 2012.

[13] IEC. IEC 61850-1 : Communication networks and systems in substations - Part 1: Introduction and overview. Technical Report 2, International Electrotechnical Commission, Geneva, 2013.

[14] L. Lampe, A. M. Tonello, and T. G. Swart. *PLC for Smart Grid*. Wiley Telecom, 2014.

[15] C. Lins, L. E. Williamson, S. Leitner, and S. Teske. *The First Decade:2004-2014. 10 years of Renewable Energy Progress*. REN21, Paris, 2014.

[16] MVV Energie. Model city mannheim beacon project. [Online] `https://www.mvv.de/en/mvv_energie_gruppe/nachhaltigkeit_2/nachhaltig_wirtschaften_1/innovationen_1/modellstadt_mannheim_1/moma.jsp`. last accessed: 07/10/2017.

[17] Node-RED.org. What is node-RED. [Online] `https://nodered.org/`.

[18] OS4ES Consortium. Requirement specification for an OS4ES. OS4ES project deliverable, November 2014.

[19] OS4ES Consortium. Analysis of smart grid communication requirements,protocol model development and technological mappings evaluation. OS4ES project deliverable, August 2015.

[20] OS4ES Consortium. Semantic middleware architecture specification. OS4ES project deliverable, July 2015.

[21] OS4ES Consortium. Test specifications for the components of the OS4ES and the communication infrastructure. OS4ES project deliverable, March 2016.

[22] OS4ES Consortium. Laboratory and field test integration and performance results. OS4ES project deliverable, November 2017.

[23] S. Paul, M. S. Rabbani, R. K. Kundu, and S. M. R. Zaman. A review of smart technology (smart grid) and its features. In *2014 1st International Conference on Non Conventional Energy (ICONCE 2014)*, pages 200–203, Kalyani, Oct 2014.

[24] A. Robinson and R. Black. Building the Power Grid of the Future. [Online] `http://www.news.gatech.edu/features/building-power-grid-future`. Last access: 07/10/2017.

[25] M. S. Saleh, A. Althaibani, Y. Esa, Y. Mhandi, and A. A. Mohamed. Impact of clustering microgrids on their stability and resilience during blackouts. In *2015 International Conference on Smart Grid and Clean Energy Technologies (ICSGCE)*, pages 195–200, Offenbach, 2015.

[26] UCLA-SMERC. UCLA smart grid energy research center (SMERC). [Online] `http://smartgrid.ucla.edu/about.html`. last accessed: 07/10/2017.

[27] M. van den Berge, M. Broekmans, B. Derksen, A. Papanikolaou, and C. Malavazos. Flexibility provision in the smart grid era using usef and os4es. In *2016 IEEE International Energy Conference (ENERGYCON)*, pages 1–6, Leuven, April 2016.

[28] www.modbus.org. Modbus messaging on tcp/ip implementation guide v1.0b. [Online] `http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf`.

[29] J. Zhang and C. A. Gunter. *IEC 61850 - Communication Networks and Systems in Substations: An Overview of Computer Science*. University of Illinois at Urabana-Champaign, Urbana, Illinois, 2007.

TABLE OF AUTHORS

Chapters and Sections	Author
1 Introduction	E. Madaha
1.1 Motivation	E. Madaha
1.2 Problem Statement	E. Madaha
1.3 Thesis Overview	E. Madaha
2 The Open System for Energy Services	
2.1 The OS4ES Concept	E. Madaha
2.2 Smart Grid	E. Madaha
2.2.1 Features of a Smart Grid	E. Madaha
2.2.1.0 Reliability	E. Madaha
2.2.1.0 Flexibility	E. Madaha
2.2.1.0 Efficiency	E. Madaha
2.2.1.0 Sustainability	E. Madaha
2.2.1.0 Market Enabling	E. Madaha
2.2.2 Technologies of a Smart Grid	E. Madaha
2.2.3 Further works on the Smart Grid	E. Madaha
2.3 Devices and Roles	E. Madaha
2.3.1 DER-System	E. Madaha
2.3.1.0 DER (Distributed Energy Resource)	E. Madaha
2.3.1.0 PV	E. Madaha
2.3.1.0 CHP	E. Madaha
2.3.1.0 Battery	E. Madaha
2.3.2 Aggregator	E. Madaha
2.3.3 Registry	E. Madaha
2.3.4 Other roles	E. Madaha
2.4 Communication Interface	E. Madaha
2.4.1 IEC 61850	E. Madaha
2.4.1.0 Core Features	E. Madaha
2.4.1.0 Data Model	E. Madaha
2.4.2 Middleware	E. Madaha
2.5 Behavioural-driven Development	E. Madaha
2.6 OS4ES Lab Test	E. Madaha
3 Requirements Analysis	
3.1 Functional Requirements	G. Mckoy
3.1.1 List of Functional Requirements	G. Mckoy
3.1.2 Detailed Description of the Functional Requirements	G. Mckoy
3.1.2.0 Integration into Middleware	G. Mckoy
3.1.2.0 End-to-End Communication	G. Mckoy
3.2 Non-Functional Requirements	G. Mckoy
3.2.1 List of Non-Functional Requirements	G. Mckoy
3.2.2 Detailed Description the Non-Functional Requirements	G. Mckoy
3.2.2.0 Smart Gateway	G. Mckoy
3.2.2.0 MQTT Compatibility	G. Mckoy
3.2.2.0 Simulation of DER Devices	G. Mckoy
3.2.3 Lab Test	G. Mckoy
4 Test Environment and Design	
4.1 Hardware Communication and Networking	G. Mckoy

Chapters and Sections	Author
4.2 Computer Hardware Configuration	G. Mckoy
4.2.1 PC1 Hardware Configuration	G. Mckoy
4.2.2 PC2 Hardware Configuration	G. Mckoy
4.3 Computer Software Configuration	G. Mckoy
4.3.1 PC1 Software Configuration	G. Mckoy
4.3.1.0 Purpose of the Selected Software	G. Mckoy
4.3.2 PC2 Software Configuration	G. Mckoy
4.3.3 Problems Faced During Software Configuration	G. Mckoy
4.3.3.0 Java Python Bridge	G. Mckoy
4.4 Communication Between Smart Gateway and DER Devices	G. Mckoy
4.4.1 Node-RED	G. Mckoy
4.4.2 Integration of Node-RED	G. Mckoy
4.4.3 Modbus-tk	G. Mckoy
4.4.4 Paho-eclipse	G. Mckoy
4.4.5 Definition of Technologies Used	G. Mckoy
4.4.5.0 Node-RED	G. Mckoy
4.4.5.0 MQTT	G. Mckoy
4.4.5.0 Modbus	G. Mckoy
4.5 Raspberry Pis	G. Mckoy
4.5.1 Raspberry Pis	G. Mckoy
4.5.2 Raspberry Pi Communication	G. Mckoy
4.5.3 Raspberry Pi Simulation Models	G. Mckoy
4.6 Design	G. Mckoy & E. Madaha
4.6.1 PV Model Description	G. Mckoy
4.6.2 Test Description	G. Mckoy
4.6.2.0 End-to-End Communication Test	G. Mckoy
4.6.2.0 Gherkin Test	E. Madaha
5 Implementation	
5.1 Simulated DER Devices	G. Mckoy
5.1.1 Communication Module	G. Mckoy
5.1.2 System Behaviour	G. Mckoy & E. Madaha
5.1.2.0 CHP	E. Madaha
5.1.2.0 PV & Converter	G. Mckoy
5.1.2.0 Battery	G. Mckoy
5.2 Middleware Integration	G. Mckoy & E. Madaha
5.2.1 Retrieving the complete DER-Server Model	E. Madaha
5.2.2 Java Python Bridge	G. Mckoy
5.3 DER-System	G. Mckoy & E. Madaha
5.3.1 Smart Gateway	G. Mckoy
5.3.1.0 Node-RED	G. Mckoy
5.3.1.0 Java	G. Mckoy
5.3.2 Forecast and Flexibility	E. Madaha
5.3.2.0 Re-sampling of the Day-ahead values	E. Madaha
5.3.2.0 Heat Demand Aggregation	E. Madaha
5.4 Full System Test	G. Mckoy
5.4.1 Additional Components Needed	G. Mckoy
5.4.1.0 Aggregator	G. Mckoy
5.4.1.0 Registry	G. Mckoy
5.4.2 Implementation Flaws	G. Mckoy

Chapters and Sections	Author
5.5 Future Work	G. Mckoy
5.5.1 Rewriting the Implementation	G. Mckoy
5.5.2 Extending on the Implementation	G. Mckoy
6 Results and Validation	
6.1 Status of the Devices	E. Madaha
6.1.1 Solar Panels (PV)	E. Madaha
6.1.2 Battery	E. Madaha
6.1.3 CHP	E. Madaha
6.2 Operation of the Devices	E. Madaha
6.2.1 Controlling the CHP for the Lab Test	E. Madaha
6.2.2 Key Constraint for the CHP	E. Madaha
6.3 Results of the First Preliminary Test	E. Madaha
6.3.1 Assessment	E. Madaha
6.4 Second Preliminary Test	E. Madaha
6.4.1 Objectives	E. Madaha
6.4.2 Operational adjustments	E. Madaha
6.4.3 Cold Delay	E. Madaha
6.4.4 Warm Delay	E. Madaha
6.4.4.0 Battery	E. Madaha
6.4.5 Results and Analysis of Preliminary Tests	E. Madaha
6.4.5.0 Preliminary Test Conclusion	E. Madaha
6.5 Final Lab Test	E. Madaha
6.5.1 DER-System Registration	E. Madaha
6.5.2 Aggregator requests for a schedule	E. Madaha
6.5.3 Aggregator sends the setpoints	E. Madaha
6.5.3.0 Results from the CHP	E. Madaha
6.5.4 PV & Battery Simulation	G. Mckoy
6.5.4.0 PV-Battery Registration	G. Mckoy
6.5.4.0 Aggregation of PV-Battery	G. Mckoy
6.5.4.0 Booking of PV-Battry Service from DER-Server	G. Mckoy
6.6 Requirements Outlook	E. Madaha
6.6.1 Functional Requirements	E. Madaha
6.6.2 Non-Functional Requirements	E. Madaha
6.6.3 Overview of achieved requirements	E. Madaha
7 Conclusion	G. Mckoy
A Physical Devices	G. Mckoy
A.1 Battery and Converter	G. Mckoy
A.1.1 User Interface of the Battery and Converter	G. Mckoy
A.1.2 Battery configuration yml file	G. Mckoy
A.2 CHP and Heat storage	G. Mckoy
A.2.1 user interface of the CHP system	G. Mckoy
A.2.2 User Interface of the Heat storage	G. Mckoy
A.2.3 Plot of Heat storage and CHP	G. Mckoy
A.2.4 CHP configuration yml file	G. Mckoy
B Implementation	
B.0.5 Problems faced while setting up Java Python Bridge	E. Madaha
B.1 Battery Simulation Model	G. Mckoy

Declaration

I declare within the meaning of section 25(4) of the Ex-amination and Study Regulations of the International De-gree Course Information Engineering that: this Bachelor report has been completed by ourselves inde-pendently without outside help and only the defined sources and study aids were used. Sections that reflect the thoughts or works of others are made known through the definition of sources.

Hamburg, November 7, 2017	
City, Date	sign: Gervais Usha Mckoy
Hamburg, November 7, 2017	
City, Date	sign: Ervin Victor Madaha