Entwicklung und Test eines SmartHome Controllers für Wärmepumpen in Kombination mit Praxos Solardachpfannen in OpenEMS

Bachelorarbeit
Studiengang Elektrotechnik – Erneuerbare Energietechnik
an der Fakultät für Elektrotechnik
der Technischen Hochschule Köln

vorgelegt von:	Enes Sarac
Matrikel-Nr.:	111 06 003
Adresse:	Roggendorfstr. 41
	51061 Köln
	enes.sarac@smail.th-koeln.de

Köln, 25.06.2021

Kurzfassung

Die 100% Nutzung von Erneuerbaren Energien steht im Mittelpunkt der heutigen Zeit. Die in vielen Haushältern installierten PV-Anlagen dienen zur Leistungsgewinnung durch Solare Energie. Es ist aber möglich die Wärmeenergie der PV-Anlage zu nutzen, um den Wärmeverbraucher im Haushalt energieeffizienter zu nutzen. Mit der solarthermischen Anlage, auch PVT-Anlagen genannt, ist es möglich die thermische Energie von der Sonne mit den Solar Modulen von PV-Anlagen zu kombinieren. Um diese Theorie zu testen, bedarf es einer Simulation, daher wird in diesem Projekt eine Simulation auf einer Open-Source Plattform, namens OpenEMS, entwickelt. Es soll ein Haushalt mit einem Haushaltsverbraucherprofil mit PV-Anlage, Batteriespeicher getestet und für eine Erweiterung der Plattform durch eine Wärmepumpe und Wärmespeicher recherchiert werden. In diesem Projekt werden die Grundlagen für das Verständnis der OpenEMS erläutert. Anhand der vorhandenen Komponenten wurde ein Entwurfsmodell für ein Wärmespeicher, der durch einen Controller geregelt wird, erstellt.

Abstract

The 100% use of renewable energy is the focus of today. The PV systems installed in many households, are used for power generation through solar energy. However, it is possible to use the thermal energy of the PV system to make the heat consumption in the household more energy efficient. With the solar thermal system also called PVT systems, it is possible to combine the thermal energy from the sun with the solar modules of PV systems. To test this theory, a simulation is needed, so in this project a simulation is developed on an open source platform called OpenEMS. A household with a household consumer profile with PV system, battery storage will be tested and researched for an extension of the platform with a heat pump and heat storage. In this project, the basics for understanding the OpenEMS are explained. Based on the existing components, a design model for a heat storage system regulated by a controller was created.

Inhalt

Abkürzung

EMS	Energie Management System
AFWCC	Apache Felix Web Console Configuration

Abbildungsverzeichnis

1 Einleitung

Die Förderung der Erneuerbaren Energie, insbesondere der Solarthermie, ist ein wichtiges Ziel im Rahmen der Minderung von Treibhausgasemissionen. Im Jahr 2019 speisten Photovoltaikanlagen 1,7% mehr in das öffentliche Netz ein als im Vorjahr, eine Produktion von 45,5 TWh [1]. Typische Anwendungen von PV-Anlagen dienen zur Erzeugung von Strom durch die Sonneneinstrahlung, jedoch wird die Zunahme an Wärme vernachlässigt und somit eine kostbare Energiequelle nicht genutzt. Das sogenannte PVT Hybridsystem nutzt diese vernachlässigte Wärmeenergie, mit Hilfe von Batterie- und Wärmespeichern bietet das Hybridsystem eine hohe erneuerbare Energiequelle zur Selbstversorgung von Strom und Wärme. Nach Möglichkeiten sollen bestehende Smart Home Anwendungen in der Open Source Plattform OpenEMS untersucht, Regelungen von Controller analysiert und beschrieben werden, um eine Simulation für ein Hybridsystem zu erstellen.

Ziel der Arbeit ist es eine Betriebsstrategie für eine Steuerung und Regelung zum Betrieb einer Photovoltaikanlage in Kombination mit thermischen Komponenten zu entwickeln. Dazu werden eine Wärmepumpe, ein Pufferspeicher, sowie ein Heizstab verwendet. Außerdem wird die Plattform der OpenEMS näher untersucht, um eine Weiterarbeit mit dem OpenEMS zu erleichtern. Am Ende werden basierend auf den Auswertungen der Versuchsdurchläufe die Betriebsstrategien bewertet.

2 Grundlagen

In den Grundlagen werden die Komponenten die für eine Solar Thermische Nutzung benötigt werden näher erläutert. Die Funktionsweise der Komponenten werden näher beschrieben, um den Algorithmus, der für die Simulationsumgebung OpenEMS genutzt wird zu verdeutlichen.

2.1 Funktionsweise einer Wärmepumpe

Die Wärmepumpe ist ein Wärmeerzeuger und funktioniert ähnlich wie ein Kühlschrank, wobei nicht Kälte, sondern Wärme produziert wird. Dabei wird ein niedrigeres Temperaturniveau auf ein höheres verwendbares Temperaturniveau angehoben. Die Außentemperatur, die uns durch die Umwelt zur Verfügung steht, wird genutzt, um dieses Verfahren zu ermöglichen. Die Umwelttemperatur wird über einen Wärmetauscher bzw. Verdampfer an ein Fluid (z.B: Propan) herangeführt, dass bereits in niedrigen Temperaturen verdampft. Anschließend wird durch ein Strombetriebenen Verdichter (Kompressor) das Fluid auf eine Temperatur von über 50°C gebracht. Darauffolgend gibt das Fluid bei einem zweiten Wärmetauscher, die Wärme an einen Wärmeverbraucher vom Haushalt ab. Das Fluid verflüssigt sich nach der Wärmeabgabe und der Ablaufzyklus beginnt von vorne. Die Leistungszahl COP (Coeficent of Performance) bestimmt das Verhältnis zwischen elektrischer und abgegebener Wärmeleistung. [2]

$$\mathbf{COP} = \frac{\mathbf{Q}\mathrm{ab}}{\mathbf{P}\mathrm{el}}$$

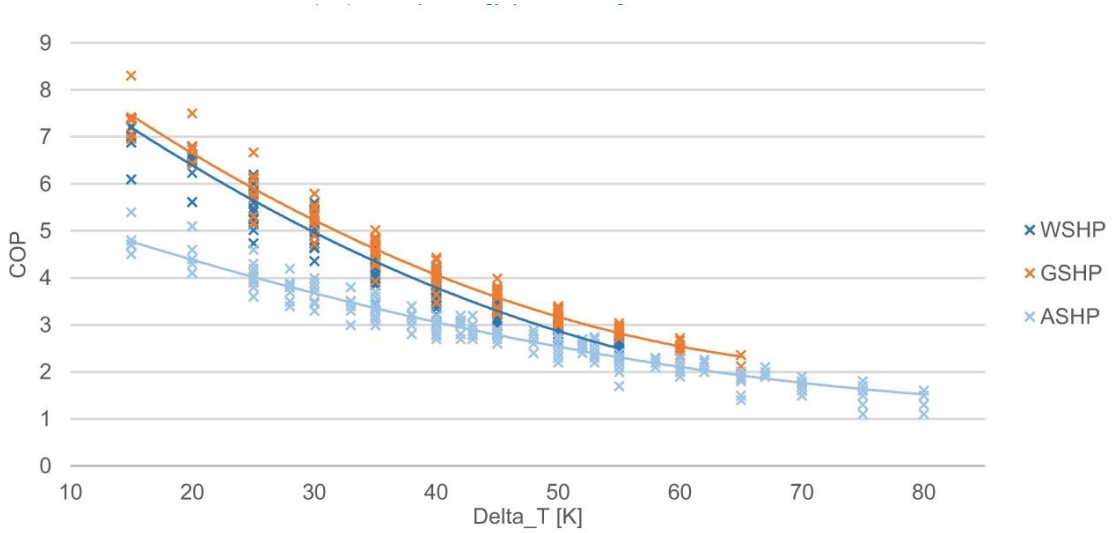
COP = Leistungszahl (**C**oefficient **o**f **P**erformance)

Qab = abgegebene thermische Leistung [kW]

Pel = elektrische Leistungsaufnahme [kW]

In der Abbildung 2.1 sieht man die Temperaturabhängigkeit des COP für den thermodynamisch idealen Prozess, der durch den Carnot-Wirkungsgrad beschrieben wird. In dieser Bachelorarbeit gehen wir näher in den Wärmepumpentyp ASHP ein. Die ASHP nutz die Umwelttemperatur und ist ein modulierbarer, ein & abschaltbarere Wärmepumpe. Die Annäherung zu der unten gelisteten Formel für den COP, wurde durch Anwendung verschiedener Temperaturbedingungen an der Wärmepumpe ASHP durchgeführt und gibt die Funktion des Graphen wieder.

$COP = 6{,}08 - 0{,}09 * \Delta T + 0{,}0005 * (\Delta T)^2$, $(ASHP)$.



Estimation of COP curves. Quadratic regressions are performed on manufacturer data[9] distinguishing between air-source heat pumps (ASHP), ground-source heat pumps (GSHP), and groundwater-source heat pumps (WSHP).

Abbildung 2.1 COP der Wärmepumpe ASHP (Graph aus [4])

2.2 Aufbau und Funktion eines Wärmespeichers

Ein Wärmespeicher, oft auch Pufferspeicher genannt, werden oft als Wasserwärmespeicher eingesetzt, die meist aus Stahl hergestellt sind. Man unterscheidet zwischen durchmischte Speicher und thermisch geschichtete Speicher. Bei unserer Anwendung ist der geschichtete Speicher sinnvoller. Während des Beladevorgang wird unten kaltes Wasser entnommen und außerhalb des Speichers erwärmt und oben wieder eingeschichtet. Die durch diesen Ablaufs Zyklus entstehende thermische Sichtung ist sehr stabil, dadurch ist es möglich auch mit einem teilweise beladenen Zustand des Wärmespeichers Wasser mit maximaler Temperatur zu Verfügung zu stellen. Der Wärmespeicher dient als Komponente einer Heizungsanlage und wird zwischen Wärmeerzeuger (Solaranlage, Wärmepumpe) und Wärmeverbrauchern angeschlossen. Die speicherbare Wärme wird wie folgt berechnet: [3].

$$\boldsymbol{Q = m * c * \Delta T}$$

m = Speichermasse

c = mittlere spezifische Wärmekapazität (bei Wasser beträgt sie 1,16 kWh/(kg*K)

ΔT= nutzbare Temperaturdifferenz (z.B zwischen Vorlauf und Rücklauf)

2.3 Energiemanagement

Die Erklärung zu dem Begriff Energiemanagement wird von der VDI-Richtlinie 4602 [Ver02] hergeleitet. Diese wird wie folgt beschrieben:

Energiemanagement ist die Koordination von Erzeugung, Wandlung, Verteilung und Nutzung von Energie zur Deckung der Anforderungen unter Berücksichtigung ökologischer und ökonomischer Zielsetzungen.

Die Definition beschreibt die Anwendung der Simulatoren, die durch gezielte Steuerung den Energieverbrauch oder die Energieerzeugung regeln. Die Simulatoren werden aber noch unterteilt in verschiedene Klassifikation.

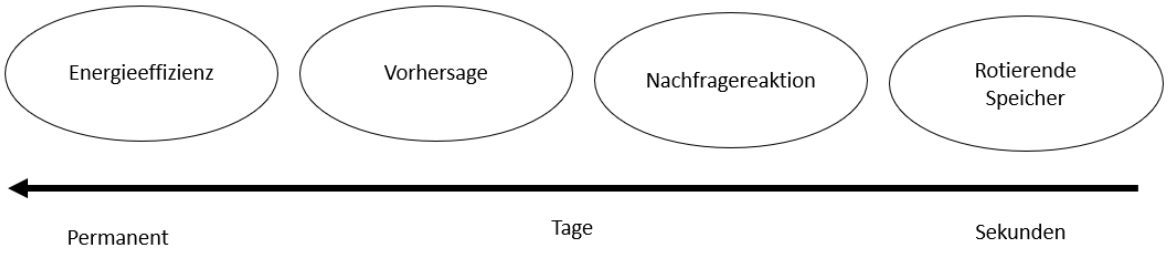


Abbildung 2.2 Klassifikation von Energiemanagementsystemen [5] („Categories of DSM“)

In der Abbildung sieht man die von Palenksy abgeleitete Abb. 2.2 „Categories of DSM“. Die Abbildung beschreibt die zeitliche Reaktion von Simulatoren und die dazugehörigen Steuerungen. Nachfolgend werden alle Kategorien bis auf die Rotierende Speicher beschrieben, da dies nicht zur Bachelorarbeit und zu der Simulationsumgebung OpenEMS eingeordnet werden konnte.

Die Energieeffizienz beschreibt die Simulatoren, die für eine Systemänderung verantwortlich sind. Die energieeffizienten Simulatoren sind Haushaltsgeräte, wie eine Wärmepumpe oder ein Wärmespeicher, die als ein Verbraucher für das EMS dienen oder ein Energieverbrauch eines Haushaltes oder einem Verbraucher simulieren. Wenn neue Verbraucher in das EMS angeschlossen werden oder neue Verbraucher des Haushaltes oder eines Verbrauches vorgenommen werden, zählen diese auch zu den Energieeffizienz Simulatoren, die permanent aktiv sind.

Die Vorhersage ist wie der Name schon sagt für die Vorhersagen des Energiemanagementsystems zuständig. In der Vorhersage werden anders als wie bei den Energieverbrauchern von den Energieeffizienten Simulatoren nicht die Daten direkt übermittelt, sondern in zeitlichen abständen gemittelt und als Prognose verwendet. In dieser Bachelorarbeit wird die Vorhersage (foresight) für die Temperatur genutzt, indem für die nächsten 24h die Temperatur ausgelesen und gemittelt ausgegeben wird. Bei diesen Simulatoren besteht aber die Chance auf fehlerhafte Datenübergaben, wenn die Daten zur Ermittlung der Vorhersage die Kapazität der vorhandenen Daten überschreiten.

Die Nachfragereaktion widerspiegelt den spezifischen Algorithmus der Controller in der Simulationsumgebung OpenEMS. Durch Energieabgaben bzw. Zugaben an den Simulatoren können sich Zustände ändern. Durch eine Zustandsänderung kann als Beispiel

ein größerer Energieverbrauch veranlasst oder eine Zufuhr von Energie gestoppt werden.

Der rotierende Seicher beschreibt die Energienetze durch Kraftwerke deren Drehzahl von der Netzfrequenz abhängt, die aber in der Arbeit nicht näher eingegangen werden, da durch Energieerzeuger wie PV bzw. PVT Anlagen diese rotierenden Speicher mit der Zeit vom Netz genommen werden.

Die verschiedenen Klassifikationen werden miteinander kombiniert, um einen kontrollierten und energieeffizienten Nutzen von den Endgeräten durch ein EMS durchgeführt werden.

3 Stand der Technik

In diesem Kapitel werden der Stand der Technik und die theoretischen Vorarbeiten für diese Bachelorarbeit als Basis für das weitere Vorgehen erläutert. Zunächst wird der aktuelle Stand der Technik von Photovoltaik in Kombinationen mit Batteriespeicher, Wärmepumpe und Pufferspeicher beschrieben. Anschließend werden die weiteren Vorgehensweisen und die Funktionsweisen der einzelnen Komponenten eingegangen.

3.1 Photovoltaik Anlage mit Batteriespeicher

Der erzeugte Strom von PV-Anlagen wird genutzt, um die Haushaltslasten auszugleichen. Sobald mehr Strom erzeugt als verbraucht wird, wird der Strom an den Batteriespeicher geladen, um die Energie effizient zu nutzen. Beim Einspeichern in den Batteriespeicher werden verschiedene Betriebsvarianten genutzt. Die netzdienstlichste Variante ist das Peak Shaving. Der Batteriespeicher wird mit dem überschüssigen erzeugten Strom der PV-Anlage nur geladen, sobald diese auch einen festgelegten Wert überschreiten. Die EEG-Einspeisevergütung für Energie aus PV-Anlagen (ca. 11ct/kWh netto) liegt weit unter den aus dem Netz bezogene Energie bei Privathaushalten (ca. 30ct/kWh brutto) [6]. Der Betrieb der PV-Anlage ist bei selbst Nutzung der erzeugten Energie am wirtschaftlichsten

3.2 Photovoltaik Anlage mit Wärmepumpe

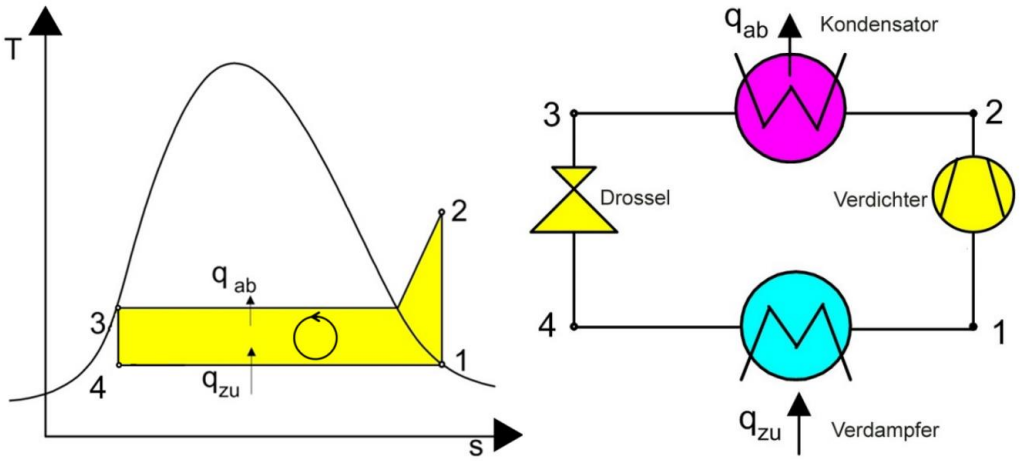


Abbildung 3.1: Das T-s Diagramm (links) und die dazugehörigen Komponenten (rechts) des Wärmepumpen-Kreisprozesses [7]

Die Wärmepumpe erbringt mehr Wärmeleistung, je kleiner die Temperaturdifferenz ist. Wenn die Außenlufttemperatur hoch ist, erbringt die Wärmepumpe auch einen hohen COP Wert. Da die Photovoltaik Anlagen sich durch die Sonnenstrahlen erhitzen, kann diese Wärmeabgabe der Photovoltaik Anlagen genutzt werden, um die Wärmepumpe effizient mit einem hohen COP Wert zu nutzen.

3.3 Photovoltaik Anlage mit einem Heizstab

Ein Heizstab ist ein elektrischer Widerstand durch den Strom fließt und der durch den Stromfluss sich erwärmt, dadurch wird die Umgebung des Heizstabes mit erwärmt. Bei der Zusammenarbeit mit PV-Anlagen ist es empfehlenswert den Heizstab nach der Größe der PV-Anlage einzubauen. Je größer eine PV-Anlage ist, desto größer sollte die Leistung des Heizstabes sein. Eine weitere Möglichkeit ist es den Heizstab in Verbindung mit dem Pufferspeicher einzubauen. Der Heizstab sollte im oberen Fünftel des Pufferspeichers positioniert werden, um die Trinkwarmwasserversorgung sicherzustellen.

4 Einführung in die OpenEMS

Die OpenEMS ist eine Open Source Plattform für Energie Management Anwendungen. Das Energiemanagementsystem vom OpenEMS wurde zur Überwachung, Steuerung und Integration von Energiespeichern mit erneuerbaren Energiequellen entwickelt. Zudem sind verschiedene Dienstleistungen wie Ladestationen für Elektrofahrzeuge, elektrolysiere und zeitliche Stromtarife vorhanden. Viele Endgeräte von verschiedenen Anbietern zur Automatisierung von Gebäuden sind vorhanden und können genutzt werden. In dieser Arbeit konzentrieren wir uns jedoch nur auf die simulierbaren Komponenten und den Controllern, da durch Simulation Kosten in Form von Zeit und Geld erspart werden. Zum Beispiel den Aufbau, Kauf, Installation von Komponenten. Allein nur durch die Simulation ist es in OpenEMS möglich ein Haushalt über das Energiemanagementsystem durch verschiedene Komponenten zu steuern und zu simulieren. Dank dem OpenEMS Asssociation e. V ist es möglich über ein Community Forum sich mit anderen Universitäten, Softwareherstellern, Hardwareherstellern und privaten Eigentümern zu kommunizieren, um Hilfe bei bestehenden Problemen in eigenen Projekten zu suchen. Um mit der Open Source Plattform OpenEMS arbeiten zu können, sind Installation der von FENECON GmbH entwickelten Software notwendig. Die Daten sind vom Urheberrecht FENECON GmbH geschützt, die entwickelten Softwares dürfen nur zur weiter Verteilung und Modifizieren nach den Bedingungen der Eclipse Public License Version 2.0 und GNU Affero General Public License Version 3 freigegeben werden. Nach Einhaltungen der License sind die Daten von der OpenEMS im GitHub frei verfügbar und können durch den Entwicklungstool Eclipse und Visual Studio geändert werden. Nach der Installation der OpenEMS stehen dem Entwickler drei lokale Internetseiten zur Verfügung. Für die Arbeit mit dem Energiemanagementsystem sind die drei lokalen Internetseiten OpenEMS Edge, -Backend und -UI notwendig. Die OpenEMS Edge und OpenEMS backend können nach den Bedingungen der Eclipse Public License Version 2.0 genutzt werden. Die OpenEMS UI wird nach den Bedingungen der GNU Affero General Public License Version 3 geregelt [4].

4.1 Aufbau von OpenEMS

Der Aufbau der OpenEMS wird durch die von FENECON installierte Software vereinfacht. Die aktuelle Version der OpenEMS kann man durch die GitHub von OpenEMS ohne jegliche Kosten downloaden. Die Software ist sehr umfangreich und beinhaltet 3 verschiedene Lokale Webseiten, OpenEMS Edge, -Backend & -UI, die über die jeweiligen Projekte gestartet werden können. Bei den beiden lokalen Webseiten Edge & Backend hilft eine Kommunikation Software namens Apache Felix Web Console Configurationen aus, um die Nutzung des EMS zu vereinfachen. Jedes dieser Elemente verfügt über seine eigenen Funktionen, die als Simulator, Controller, Scheduler und auch als Device genutzt werden können.

4.1.1 OpenEMS Edge

Die OpenEMS Edge ist einer der lokalen Internetseiten, die für die Kommunikation zwischen den Geräten und dem Service zur Verfügung steht. Das OpenEMS Edge wird aktiviert, indem in dem Projekt „Edge.application“, die EdgeApp.Bndrun gestartet wird. In dem Benutzerfeld EdgeApp.Bndrun kann ebenfalls die neu erstellen Projekte der AFWCC hinzugefügt werden. Die Geräte können über die OpenEMS Edge Lokalseite durch Eingabe der Initialwerte bzw. Config-Werte dem System hinzugefügt werden.

4.1.2 OpenEMS backend

Die OpenEMS backend ist eine lokale Internetseite, die auf einem Cloud-Server läuft. Das OpenEMS backend wird aktiviert, indem in dem Projekt „Backend.application“, die BackendApp.Bndrun gestartet wird. OpenEMS backend verbindet das Edge-System und ermöglicht Aggregation, Überwachung und Steuerung über das Internet.

4.1.3 OpenEMS UI

Die OpenEMS UI ist eine Lokale Internetseite, die für den Konsumenten bzw. Nutzer zur Verfügung steht. In der OpenEMS UI werden die Daten in Echtzeit dargestellt, dies kann über einen Webbrowser oder ein Smartphone aufgerufen werden.

.

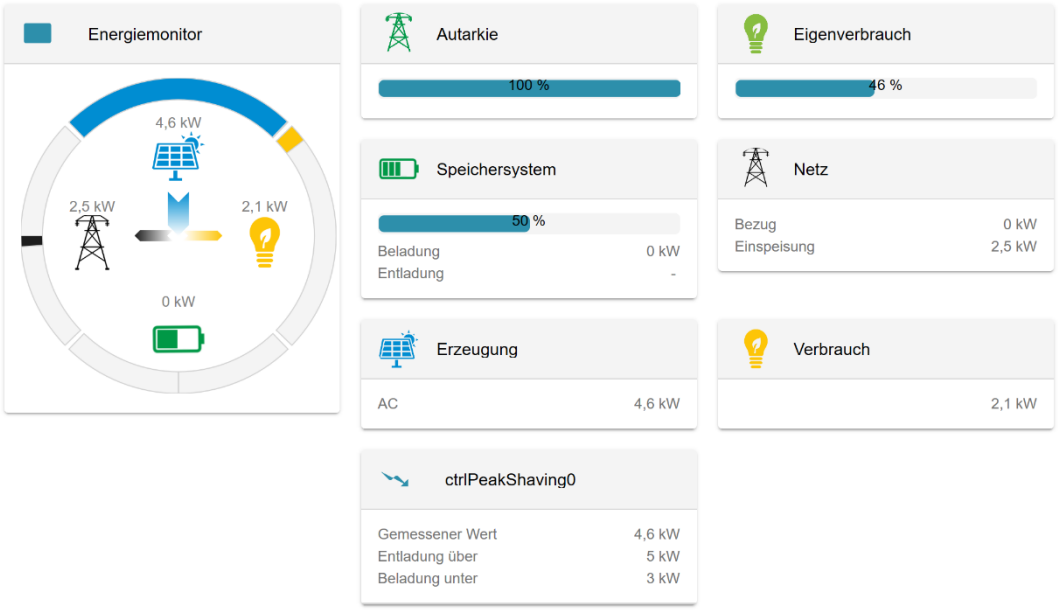


Abbildung 4.1 Darstellung der OpenEMS UI

4.1.4 Apache Felix Web Console Configuration

Die Apache Felix Web Console Configuration ist die Benutzeroberfläche der beiden Lokalen Webseiten Edge & Backend. Die vorhandenen Projekte der OpenEMS sind als Elemente in AFWCC gelistet und können nach Belieben der Energiesystem Modulation hinzugefügt oder entfernt werden. Beim Hinzufügen eines Elements öffnet sich ein Fenster. Die eingetragenen Werte im Fenster gelten als „default“–Werte(siehe Abbildung…)

und können bei Bedarf geändert werden. Durch die Eingabe des Fensters, wird das Element dem System hinzugefügt.

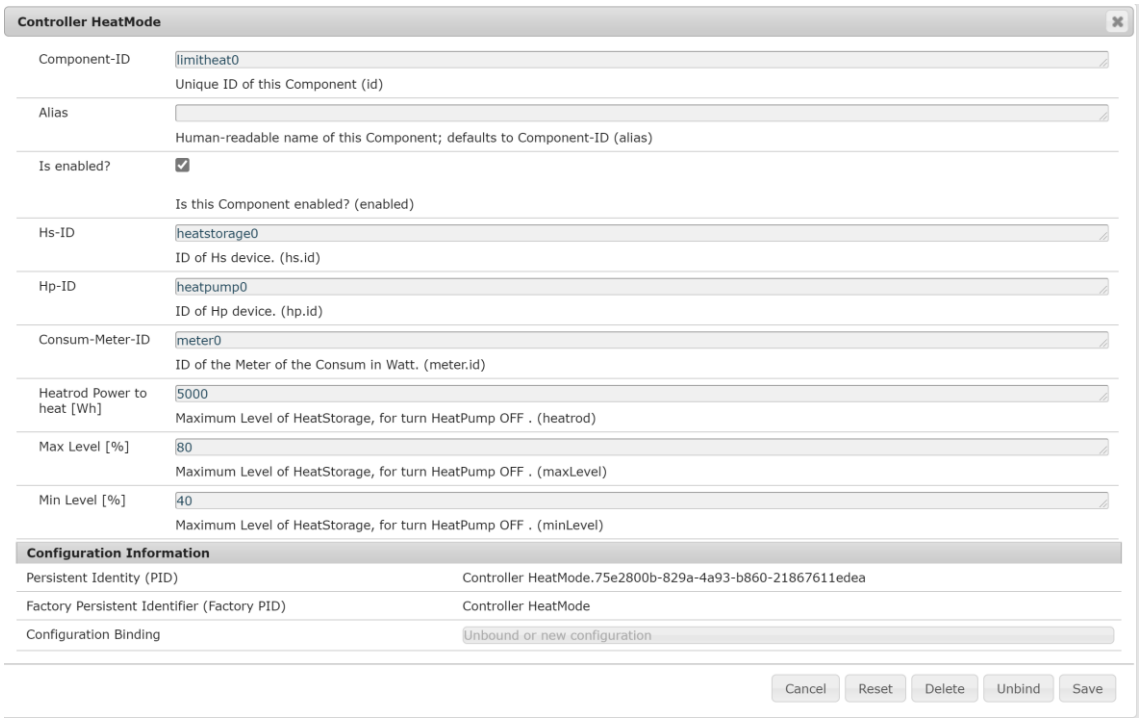


Abbildung 4.2: AFWCC - Eingabefenster (initialisierung der Projekte)

Jedes Projekt in der OpenEMS besitzt eine Config,java Datei, die den AFWCC widerspiegelt. Unten in der Abb. 4.3 sind die Eingabeeinforderungen des Controllers Heatmode zu sehen, die in der Abb. 4.2 in AFWCC dargestellt werden. Die oberste Zeile gibt die Adresse der eigenen Klasse an. In den unteren Zeilen wurden die Adressen der Klassen, HS-ID, HP-ID & Consum-Meter-ID übergeben. Durch die Übergabe ist es möglich auf die Klassen zugreifen und über Ihre ChannelId’s zu kommunizieren.

```java
@ObjectClassDefinition(//
        name = "Controller HeatMode", //
        description = "")
@interface Config {

    @AttributeDefinition(name = "Component-ID", description = "Unique ID of this Component")
    String id() default "limitheat0";

    @AttributeDefinition(name = "Alias", description = "Human-readable name of this Component; defaults to Component-ID")
    String alias() default "";

    @AttributeDefinition(name = "Is enabled?", description = "Is this Component enabled?")
    boolean enabled() default true;

    @AttributeDefinition(name = "Hs-ID", description = "ID of Hs device.")
    String hs_id() default "heatstorage0";

    @AttributeDefinition(name = "Hp-ID", description = "ID of Hp device.")
    String hp_id() default "heatpump0";

    @AttributeDefinition(name = "Consum-Meter-ID", description = "ID of the Meter of the Consum in Watt.")
    String meter_id() default "meter0";

    @AttributeDefinition(name = "Heatrod Power to heat [Wh]", description = "Maximum Level of HeatStorage, for turn HeatPu
    int heatrod() default 5000; //OFF

    @AttributeDefinition(name = "Max Level [%]", description = "Maximum Level of HeatStorage, for turn HeatPump OFF .")
    int maxLevel() default 80; //OFF

    @AttributeDefinition(name = "Min Level [%]", description = "Maximum Level of HeatStorage, for turn HeatPump OFF .")
    int minLevel() default 50; //ON

    String webconsole_configurationFactory_nameHint() default "Controller io.openems.edge.controller.HeatMode [{id}]";

}
```

Abbildung 4.3: Config.java, vom Controller HeatMode

4.1.5 Architektur von OpenEMS

Die OpenEMS hat einen bestimmten Ablaufzyklus, der die verschiedenen Elemente nach der angegebenen Reihenfolge abläuft.

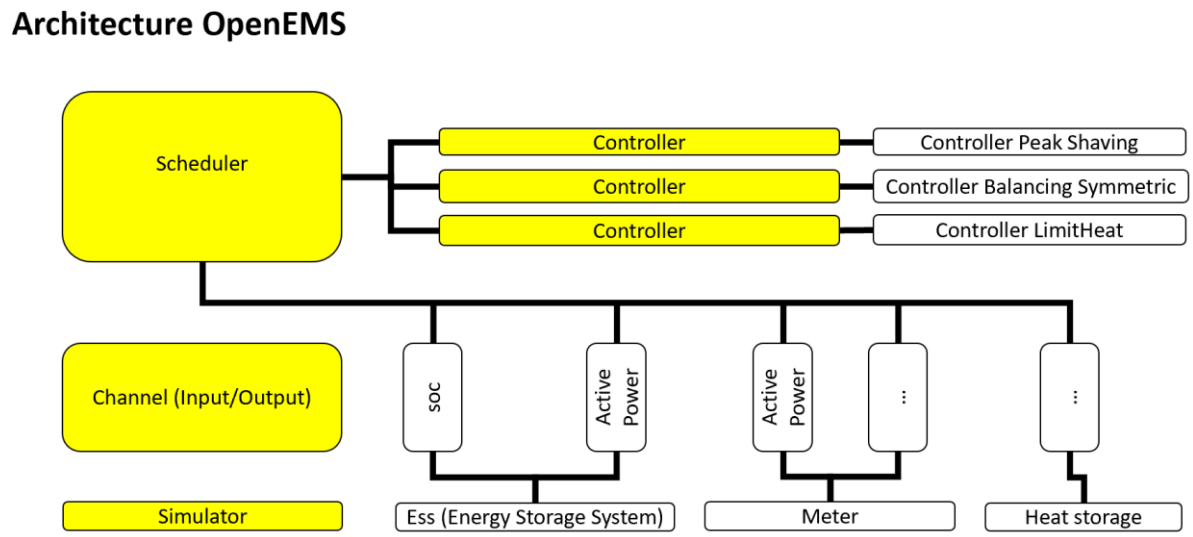


Abbildung 4.4: Architecture OpenEMS

Die Abb. 4.4 „Architecture OpenEMS“ soll den Ablauf von OpenEMS verdeutlichen. Ohne einen Scheduler, der die Reihenfolge des Ablaufes bestimmt, kann keine Simulation gestartet werden. Zuerst werden die Ein- bzw. Ausgänge des EMS ausgeführt. Da wir bei unserer Bachelorarbeit keine Hardwaregeräte genutzt, sondern nur Simulationen

erstellt haben, werden die Simulationen als erstes ausgeführt, die jeweiligen Ein bzw. Ausgänge erstellen. Der Scheduler wird darauffolgend ausgeführt. Abgefragt werden die Controller, die im Energiemanagementsystem aktiviert wurden und nach der gelisteten Reihenfolge ausgeführt. Wichtig ist hier zu beachten das bei einer falschen Angabe einer Reihenfolge es zu ungewollten Ergebnissen der Simulation führen kann. Der Ablauf des Zyklus erfolgt jede Sekunde, die aber im jeweiligen Projekt umgeändert werden kann.

4.1.6 Cylce

Der Zyklus erfolgt jede Sekunde und bestimmt den Ablauf des Programmes. Die Klasse CycleWorker startet eine Stoppuhr und bestimmt anschließend die TOPIC's für den handleEvent und führt diese nacheinander aus. Nachdem alle Simulatoren ausgeführt wurden, überprüft der CylcleWorker, ob ein Schedule definiert wurde, falls das nicht der Fall ist, wird ein Fehler ausgegeben. Zuletzt werden die Controller nach Ablauf der Schedule über ihre „run()“ Funktion ausgeführt.

4.2 Templates

Die Projekte von OpenEMS werden in vier verschiedene Templates unterteilt.

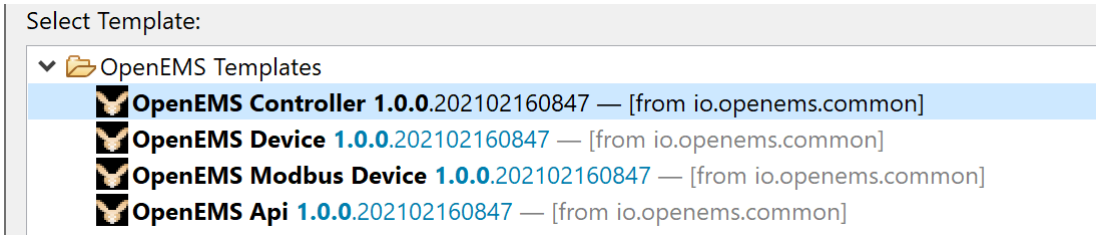


Abbildung 4.5: OpenEMS Templates

Die Templates Controller, Device, Modbus Device und API werden durch Ihre verschiedenen Beziehungen zwischen den inkludierten Bibliotheken unterschieden und vereinfachen die Erstellung von eigenen Projekten.

4.2.1 API

Die API ist ein Interface und dient als Anwendung um Daten zu empfangen diese in Channels zu kategorisieren, um diese im ModbusRecord zu speichern. Anschließend werden die Daten weitergeleitet, interpretiert und erforderliche Maßnahmen werden durchgeführt, um den Benutzer die Daten wieder zurückzuschicken.

ChannelId

```
23      public enum ChannelId implements io.openems.edge.common.channel.ChannelId {
24          MODE(Doc.of(Mode.values())), //
25          COP(Doc.of(OpenemsType.INTEGER) //
26                  .unit(Unit.PERCENT)),
27          ACTIVE_POWER(Doc.of(OpenemsType.INTEGER) //
28                  .unit(Unit.WATT) //
29                  .text(POWER_DOC_TEXT)),//
30          AWAITING_HYSTERESIS(Doc.of(Level.INFO) //
31                  .text("Would change State, but hysteresis is active")), //
```

Abbildung 4.6: Darstellung der Enumeration ChannelId

Die Enumeration ChannelID wird durch die Eingabe eines Namens und einer jeweiligen spezifischen Option definiert. In Zeile 25 wird ein ChannelID mit dem Namen COP, dem Wert Integer und der Einheit Prozent deklariert. Es können auch eigene Enumerationen als spezifische Option definiert, wie in dem Beispiel MODE in Zeile 24, der auf seine eigenen Werte über Mode.values() zugreift. Auch können die Channels einen vorher definierten Text ausgeben, wie in Zeile 31 zu sehen ist.

Getter & Setter Methode

```
114
115     public default Channel<Mode> getModeChannel() {
116         return this.channel(ChannelId.MODE);
117     }
118
119     public default Mode getMode() {
120         return this.getModeChannel().value().asEnum();
121     }
122
123     public default void _setMode(Mode mode) {
124         this.getModeChannel().setNextValue(mode);
125     }
126
127     public default IntegerReadChannel getActivePowerChannel() {
128         return this.channel(ChannelId.ACTIVE_POWER);
129     }
130
131     public default Value<Integer> getActivePower() {
132         return this.getActivePowerChannel().value();
133     }
134
135     public default void _setActivePower(Integer value) {
136         this.getActivePowerChannel().setNextValue(value);
137     }
138
139     public default void _setActivePower(int value) {
140         this.getActivePowerChannel().setNextValue(value);
141     }
```

Abbildung 4.7: Codeverlauf, der Getter-& Setter Methoden

In OpenEMS findet man ein bestimmtes Schema bei dem Aufrufen oder Deklarieren der ChannelID's. Jeder Channel sollte auch eine „getChannel()“ Methode haben, die den

ChannelId zurückgibt. Um die Zugriffsrechte auf die ChannelId's zu umgehen, wird eine weitere get() Methode genutzt, die den Wert im Channel als Rückgabewert besitzt. In Zeile 120 wird der Wert des Channels als Enumeration zurückgegeben und in der Zeile 132 wird nur der Wert zurückgegeben, da der Channel ACTIVE_POWER nur über ein Integer verfügt. Sollte eine ChannelID auch von anderen Klassen aufgerufen werden, um diese zu überschreiben, werden bei den Channels auch eine Setter-Methode definiert, die den getChannel() aufruft und über die Funktion setNextvalue(value) den übergebenen Wert mit eingibt.

ModbusSlaveNatureTable

Der ModbusSlaveNatureTable speichert die vorher definierten ChannelID's mit den jeweiligen Einheiten im ModbusRecordChannel, indem er die Zugriffsrechte, sowie die Bytellänge und die Adresse für den ChannelId bestimmt.

```
105    public static ModbusSlaveNatureTable getModbusSlaveNatureTable(AccessMode accessMode) {
106        return ModbusSlaveNatureTable.of(hp.class, accessMode, 100) //
107                .channel(0, ChannelId.MODE, ModbusType.UINT16) //
108                .channel(1, ChannelId.COP, ModbusType.UINT16) //
109                .channel(2, ChannelId.ACTIVE_POWER, ModbusType.FLOAT32) //
110                .channel(3, ChannelId.HEAT_POWER, ModbusType.FLOAT32) //
111                .build();
112    }
```

Abbildung 4.8: Codeabschnitt, der Methode ModbusSlaveNatureTable

4.2.2 Simulator

Die Simulatoren werden über die handleEvent Methode ausgeführt. Sie werden in Verschiedene TOPIC's unterteilt und können nach Bedarf in dem gewünschten Abschnitten bzw. TOPIC'S eingesetzt und ausgeführt werden.

Activate

Die Activate Methode greift auf die config.java Datei im jeweiligen Projekt zu, der die Werte, die im AFWCC eingegeben werden. Die vom Benutzer bestimmten Werte werden genutzt, um die Simulationen oder die Controller zum Starten zu bringen.

hadleEvent

Die Methode handleEvent, bestimmt in welchem Abschnitt bzw. Prozess die Methoden der Simulatoren aufgerufen werden sollen. In dem unteren Beispiel wird „UpdateChannels()" vor allen anderen Prozessen zum Ablauf gezwungen, und in der Zeile 220 wird die Methode „calculateEnergy()" nach allen Prozessen aufgerufen, um die umgeänderten ChannelId Werte nach Bedarf weiterzubearbeiten und weiterzuleiten. Der Topic Process Image findet vor der Überprüfung einer Schedule, sowie das Ausführen der Controller statt.

```
214     @Override
215     public void handleEvent(Event event) {
216         switch (event.getTopic()) {
217         case EdgeEventConstants.TOPIC_CYCLE_BEFORE_PROCESS_IMAGE:
218             this.updateChannels();
219         case EdgeEventConstants.TOPIC_CYCLE_AFTER_PROCESS_IMAGE:
220             this.calculateEnergy();
221             break;
222         }
223     }
```

Abbildung 4.9 Codeabschnitt, der Funktion handleEvent

UpdateChannel

Die Methode UpdateChannel gilt bei den Simulatoren als Main Funktion. Sie wird in jedem Zyklus aufgerufen und beinhaltet den spezifischen Ablauf der Simulation.

DebugLog

Die Methode DebugLog, dient zur Wiedergabe der Variablen bzw. ChannelId Werte auf der Eclipse Console.

```
252     @Override
253     public String debugLog() {
254     return "E:" + this.energyStorage//
255             + "; Tcurrent:" + this.getCurrentOutsideTemperatureChannel().getNextValue()//
256             + "; MemoryLevel:" + this.memoryLevel//
257             + "; Highvolume:" + this.highvolume//
258             + "; ActivePower:" + this.getActivePowerChannel().value().get()//
259             + "; volume:" + this.volume;//
260     }
```

Abbildung 4.10: Programmauschnitt von der Methode DebugLog

4.3 Controller

Der Controller ist anders als eine Simulation, diese verfügt über kein handleEvent Methode, sondern wird direkt über den CycleWorker ausgeführt und hat somit eine bestimmte Position im Ablauf. Nur die Schedule kann den Ablauf zwischen den einzelnen Controllern umpositionieren.

Activate

Siehe 4.2.2 Simulator -> Activate

Run

Die Run Funktion gilt als main Funktion und beinhaltet den spezifischen Ablauf, den der Controller benötigt, um Werte von anderen Komponenten umzuändern und zurück- bzw. weiterzuleiten.

DebugLog

Siehe 4.2.2 Simulator -> DebugLog

5 Analyse der Elemente

In diesem Kapitel werden die vorhandenen Projekte von der Open Source Plattform OpenEMS näher analysiert.

5.1 Scheduler

Die Scheduler bestimmen in welcher Reihenfolge die Controller oder Simulatoren ausgeführt werden. Dies erfolgt durch die Übermittlung der Liste der IDs der zu bearbeitenden Komponenten.

5.1.1 Scheduler All Alphabetically

Der Scheduler All Alphabetically führt die Controller geordnet nach der eigenen Auflistung aus, die restlichen Controller werden nach dem Alphabet sortiert ausgeführt.

5.2 Simulator

Ein Simulator sind Komponenten, die durch Berechnungen eine Simulation erstellen, damit andere Controller darauf zugreifen können. Über die Methode „HandleEvent()“ entnimmt die Methode den Topic Zustand des EdgeEventConstans, um je nach Topic die Simulation zu dem Arbeitszyklus hinzuzufügen. Dabei werden Simulationen meistens im Topic „CYCLE_BERFORE_PROCESS_IMAGE“ abgespielt. Sobald der Prozess Zustand es zulässt, wird die Methode UpdateChannel() ausgeführt. Die Methode dient als Main Funktion, in der alle Algorithmen für die Funktion des Simulators aufgerufen werden. Zudem besitzen Simulatoren über die Methode „getModbusSlaveNatureTable()“, die für das Erstellen von ChannelIds genutzt wird. Außerdem können auch bestehende ChannelIds über die Funktion OpenemsComponent.getModbusSlaveNatureTable(accessMode) aktualisiert werden. Simulatoren dienen zur Ausführung von CSV-Dateien, um als Verbraucher ausgeführt zu werden, oder ein Simulator, der die Funktion eines Batteriespeichers codiert, beinhaltet. Man kann damit auch eine Simulation für eine Wärmepumpe oder Warmwasserspeicher programmieren.

5.2.1 Simulator Datasource: CSV Predefined

Der Simulator Datasource: CSV Predefined ist geeignet, um CSV-Dateien einzulesen und zu simulieren. In der Spalte Source besteht die Möglichkeit eine CSV-Input Datei auszuwählen, wenn keine vorhanden ist, bietet OpenEMS auch eine Standard CSV-Datei an. Dies kann genutzt werden, um CSV Dateien von Haushalts-Lastprofilen oder von PV-Erzeugungsprofilen zu simulieren. Bei dem Auswahlfenster Faktor ist es möglich die Größenordnung der importierten CSV Datei zu ändern.

5.2.2 Simulator GridMeter Acting

Der Simulator Grid Meter Acting ist eine „Aktive“-Komponente, die eine CSV-Datei benötigt. Die Daten von der CSV-Datei werden als Verbrauch ausgewertet.

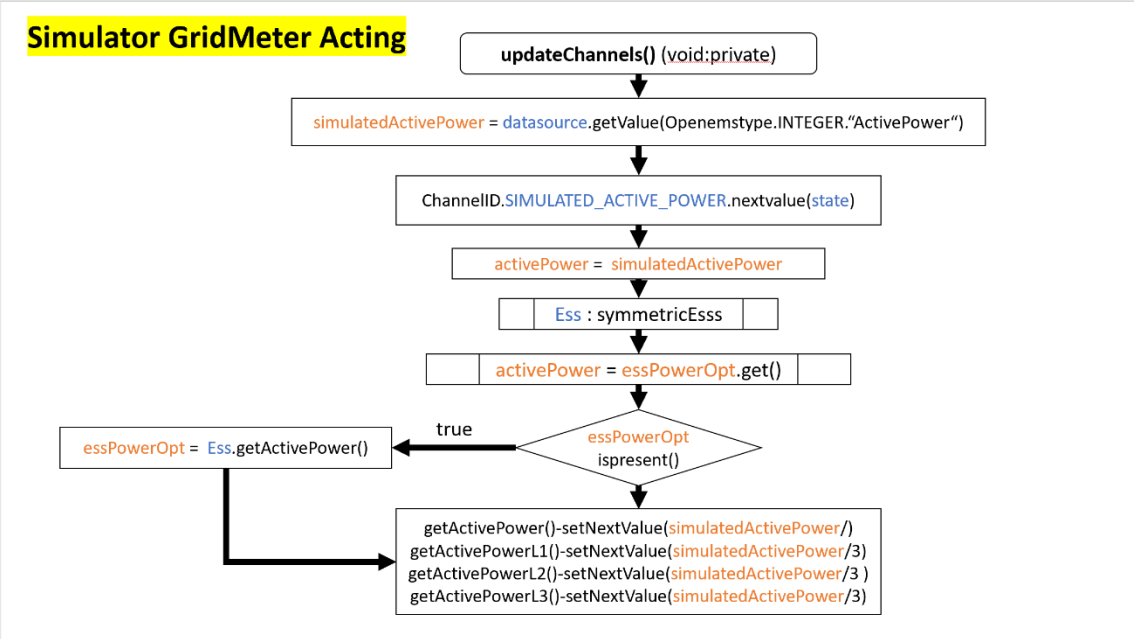


Abbildung 5.1 Programmablaufsplan vom Simulator GridMeter Acting

5.2.3 Simulator GridMeter Reacting

Simulator Grid Meter Reacting, eine „Reagierende“-Komponente, simuliert die Leistung die ins Netz eingespeist bzw. den benötigten Bezug vom Netz erbringen soll.

5.2.4 Simulator ProduktionMeter Acting

Der Simulator ProduktionMeter Acting ist eine „Aktive“-Komponente, die eine CSV-Datei benötigt. Die Daten von der CSV-Datei werden als Produktion ausgewertet.

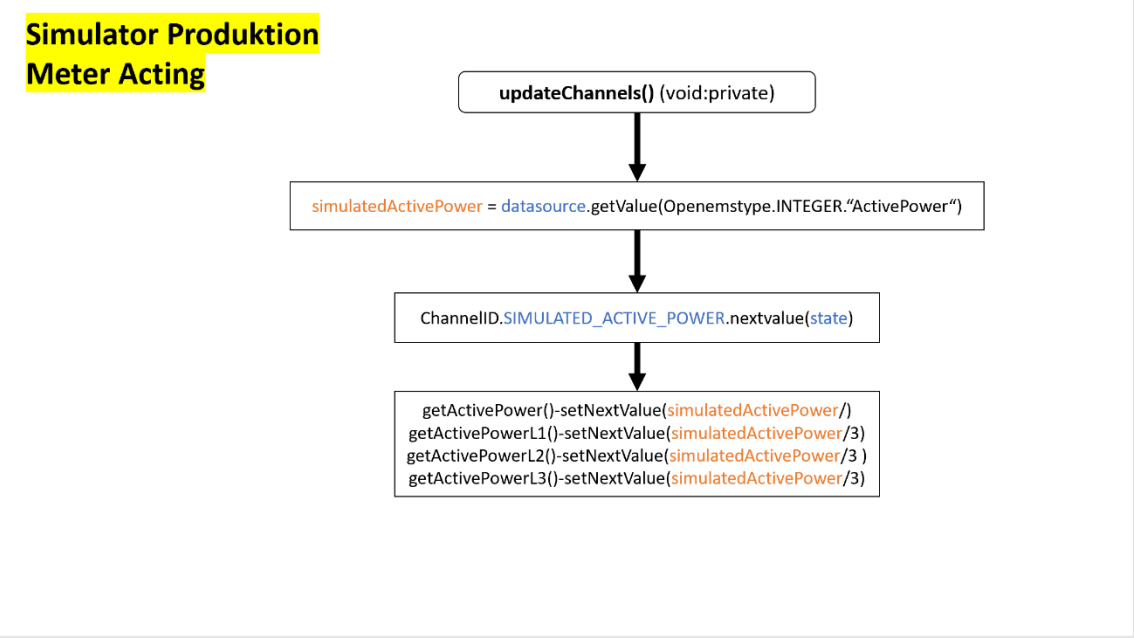


Abbildung 5.2 Programmablaufsplan vom Simulator Produktion Meter Acting

5.2.5 Simulator EssSymmetric Reacting

Der Simulator EssSymmetric Reacting simuliert einen Energiespeichersystem, wobei die Werte der Scheinleistung [VA], die Kapazität [Wh] und der Ladezustand [SoC] einzugebende Werte sind, die bei nicht Eingabe den Default Wert übernehmen.

- Max Apparant Power [VA]
- Capacity [Wh]
- Initial State of Charge [%]

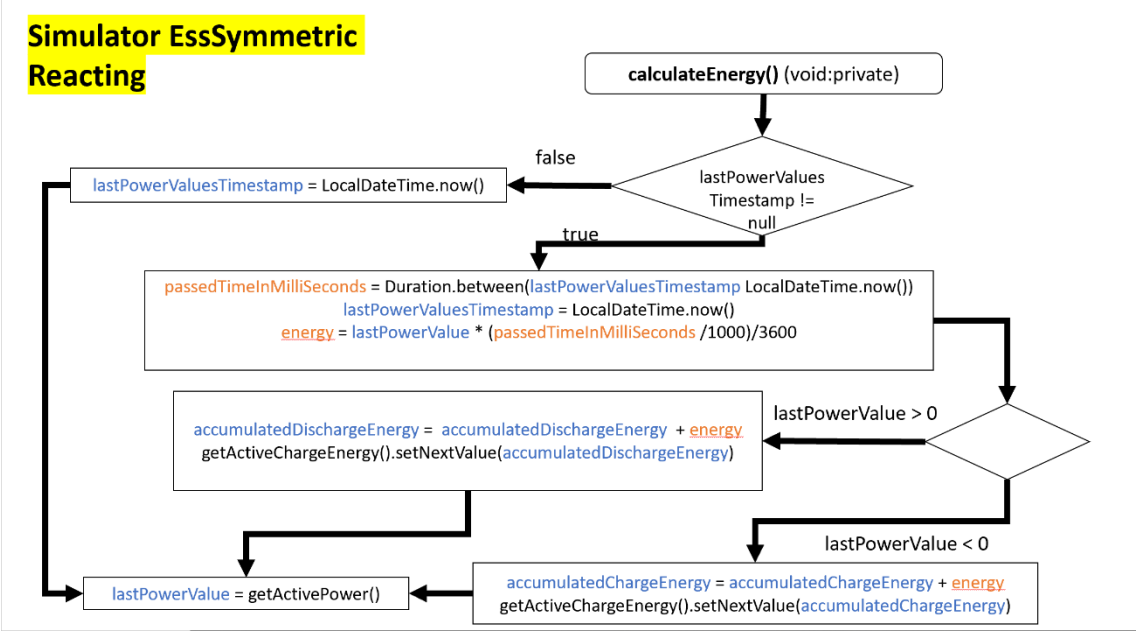


Abbildung 5.3 Programmablaufsplan vom Simulator EssSymmetric Reacting

5.3 Controller

Controller sind Elemente, die auf andere Elemente zugreifen und bei Bedarf diese auch regeln. Über die Klasse ComponentManager wird eine Instanz deklariert, mit der auf andere Komponenten die über dem Config-Menü übergebenen IDs zugreifen kann und den Datentransfer erleichtert. Die Run Methode dient als Main Klasse und wird bei jedem Cycle durch die Klasse AbstractWorker aufgerufen, um alle Controller während der jeweiligen Topics ausführen.

5.3.1 Controller Debug Log

Der Controller Debug Log gibt Informationen über alle genutzten Elemente in der Eclipse Console wieder.

5.3.2 Controller API Websocket

Der Controller API Websocket schaltet den Zugriff auf eine lokale Internetseite frei, dadurch gelangen wir auch in das User Face des OpenEMS in der alle genutzten Elemente in echt Zeit dargestellt werden.

5.3.3 Controller Peak-Shaving Symmetric

Der Controller Peak-Shaving Symmetric regelt die Leistungsübertragung zwischen der PV-Anlage und dem Energiespeichersystem. Wenn die Erzeugte PV-Leistung den Peak-Wert erreicht, wird das Energiespeichersystem so lange Entladen bis die Recharge (wiederaufladbar) Leistung erreicht ist. Die vor dem Ausführen des Controllers erforderlichen Config-Einstellungen:

- Ess-ID
- Meter-ID
- Peak-Shaving power [W]
- Recharge power [W]

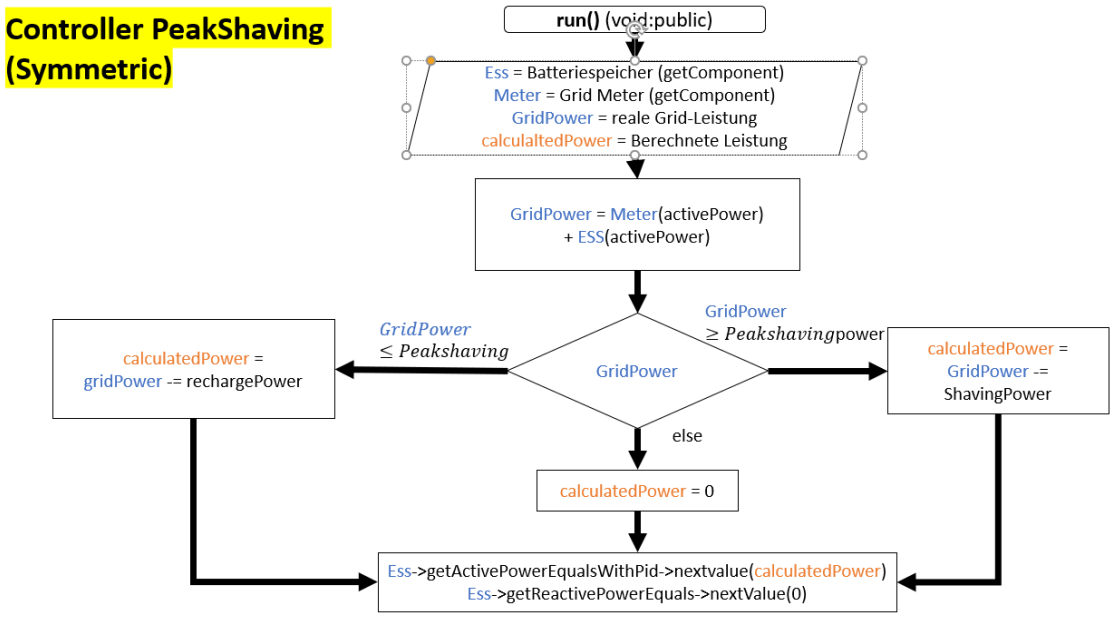


Abbildung 5.4 Programmablaufsplan vom Controller PeakShaving Symmetric

In der Abbildung 2.1 sieht man die Methode „run ()“ von dem Controller PeakShaving. Zuerst werden die durch das Config-Menü eingegebenen IDs in die Komponenten des Batteriespeichers (Ess) und des Grid Meters (Meter) mittels der Funktion „getComponent ()“ übergeben. Anschließend werden von den Komponenten die „getActivePower ()“ Funktion aufgerufen, um die Leistung der beiden Komponenten in die Variable GridPower zu speichern. Wenn die Variable GridPower größer als die PeakshavingPower vom Config-Menü ist, wird die Variable GridPower von der PeakshavingPower subtrahiert und calculatedPower übergeben. Falls die Leistung vom GridPower kleiner als die Leistung vom Peakshaving ist, wird GridPower von rechargePower subtrahiert und in calculatedPower eingespeichert. Die Variable rechargePower ist ebenfalls vorher im Config-Menü definiert worden. Bei keiner eindeutigen Zuordnung von GridPower wird die Variable calculatedPower gleich null gesetzt. Zuletzt werden die Ergebnisse an den Batteriespeicher weitergegeben, indem bei der Variable Ess die Funktion “getSetActivePowerEqualsWithPid ()“ aufgerufen wird, und mit der Funktion „setNextWriteValue ()“ die Variable calculatedPower übergeben wird.

5.3.4 Controller Fix-Active Power Symmetric

Der Controller Fix-Active Power Symmetric regelt die Auf- bzw. Entlade Leistung des Energiespeichersystems.

- Ess-ID
- Discharge/Charge Power [W]

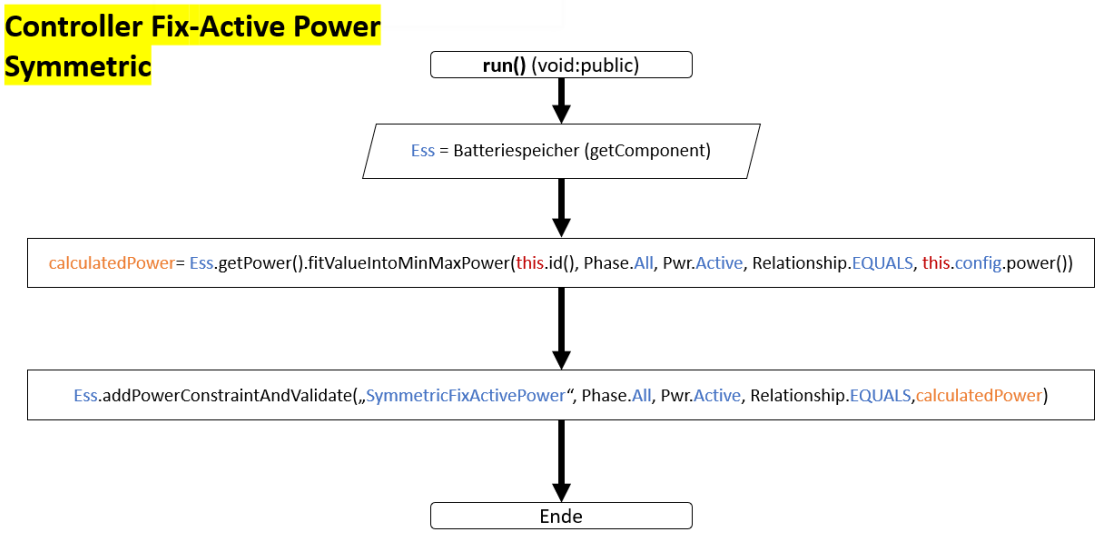


Abbildung 5.5 Programmablaufsplan vom Controller Fix-Active Power Symmetric

Die „run ()“ Methode von dem Controller Fix-Active Power Symmetric überträgt die von dem Config-Menü entnommenen Ess-ID in die Variable Ess mit der Funktion „getComponent ()“. Daraufhin wird die Variable calculatedPower mit der Variable Ess durch den Funktion Aufruf „getPower ()“und „fitVauleIntoMinMaxPower ()“ berechnet. Die Funktion „fitValueIntoMinMaxPower ()“ überprüft die Eingabe vom Discharge/Charge Power, ob die Eingabe im Config-Menü die minimale bzw. maximale Leistung des Batteriespeichers überschreitet. Falls die minimale Leistung unterschritten wurde, wird die minimale Leistung vom Batteriespeicher als Discharge/Charge Power entnommen, andernfalls wenn die maximale Leistung überschritten wurde, wird die maximale Leistung des Batteriespeichers entnommen. Wenn der Wert zwischen dem maximalen und minimalen Wert liegt, wird der Wert vom Config-Menü (this.config.power ()) an calculatedPower übergeben. Als nächstes wird dem Batteriespeicher über die Funktion „addPowerConstrainAndValidate ()“ ein Constraint mit dem calculatedPower in das Interface „Power ()“ hinzugefügt. Constraint ist ein Array der Klasse Constraint, die eine Beschreibung als String, eine double Variable, die Phase und den Status als Enumeration einspeichert.

5.3.5 Controller ESS Minimum Discharge Period

Der Controller ESS Minimum Discharge Period entlädt die Batterie, sobald die Leistung, während dem Ladevorgang, einen bestimmten Wert überschreitet. Die Entladung ist von der Eingabe der Zeitlichen Dauer abhängig.

- Discharge Power activate Value [W]

- Minimum discharge Power [W]
- Discharge Time [min]

.

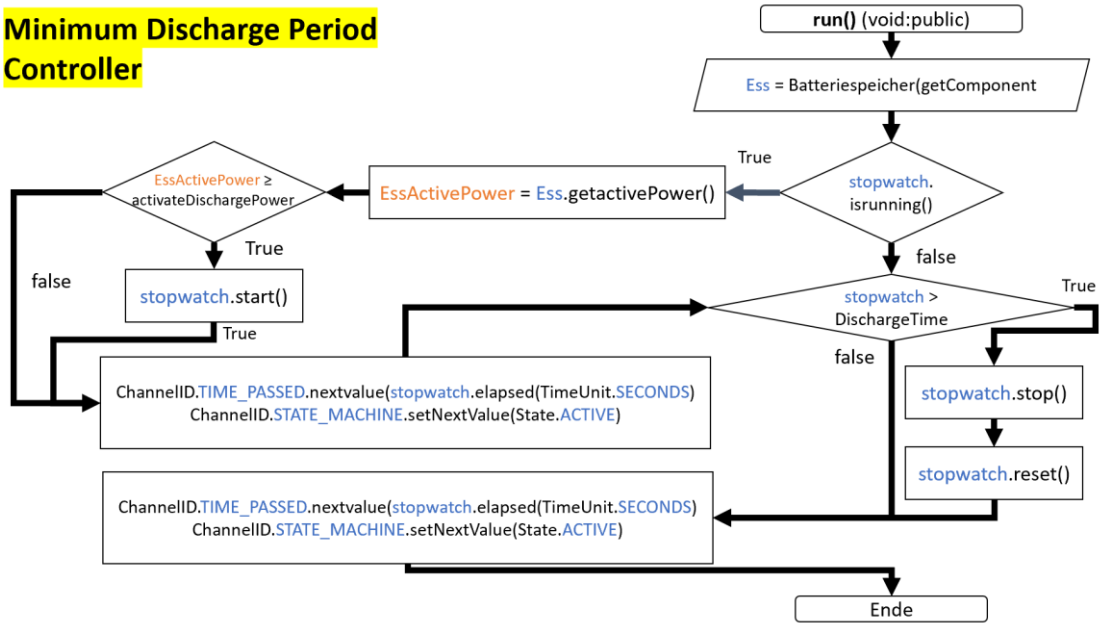


Abbildung 5.6 Programmablaufsplan vom Controller Minimum Discharge Period

In der oberen Abbildung 2.3 ist der Controller Minimum Discharge Period mit der Methode „run ()“ abgebildet. Als erstes wird die Ess-ID vom Batteriespeicher mittels der Funktion „getComponent ()“ an die Variable Ess übergeben. Daraufhin folgt eine Abfrage über die Variable stopwatch mit der Funktion „isRunning ()“, die ein true oder false zurückgibt. Wenn die stopwatch am Laufen ist, wird die Leistung des Batteriespeicher über die Funktion „getActivePower ()“ mit der Variable Ess an die Variable EssActivePower übergeben. Wenn der Wert EssActivePower größer ist als die im Config-Menü eingegebene „Discharge“ Leistung, dann wird die Variable stopwatch mit der Funktion „start()“ auf true als Rückgabewert gestellt. Mit der Funktion channel(ChannelId.TIME_PASSED)“ wird auf den Channel TIME_PASSED zugegriffen und durch die Funktion „nextvalue ()“ wird die errechnete Zeit durch die Funktion „elapsed()“ übergeben. „Elapsed ()“ überprüft den Status der stopwatch und übergibt die aktuelle Zeit wieder.

5.3.6 Controller ESS Limit Total Discharge

Der Controller ESS Limit Total Discharge regelte über die Ladekapazität des Energiespeichersystem die zu erzwingende Leistung des Energiespeichersystem bei zu niedriger Ladekapazität. Zudem wird die Entladung bei einer Bestimmten Ladekapazität blockiert.

- Min SoC [%]
- Force-Charge SoC [%]
- Force-Charge Power [W]

Während des erstellen des Controllers wird durch die Methode „activate()“ aufgerufen. Die Funktion „createUnstarted()“ übergibt die aktuelle Zeit an die variable stopwatch.

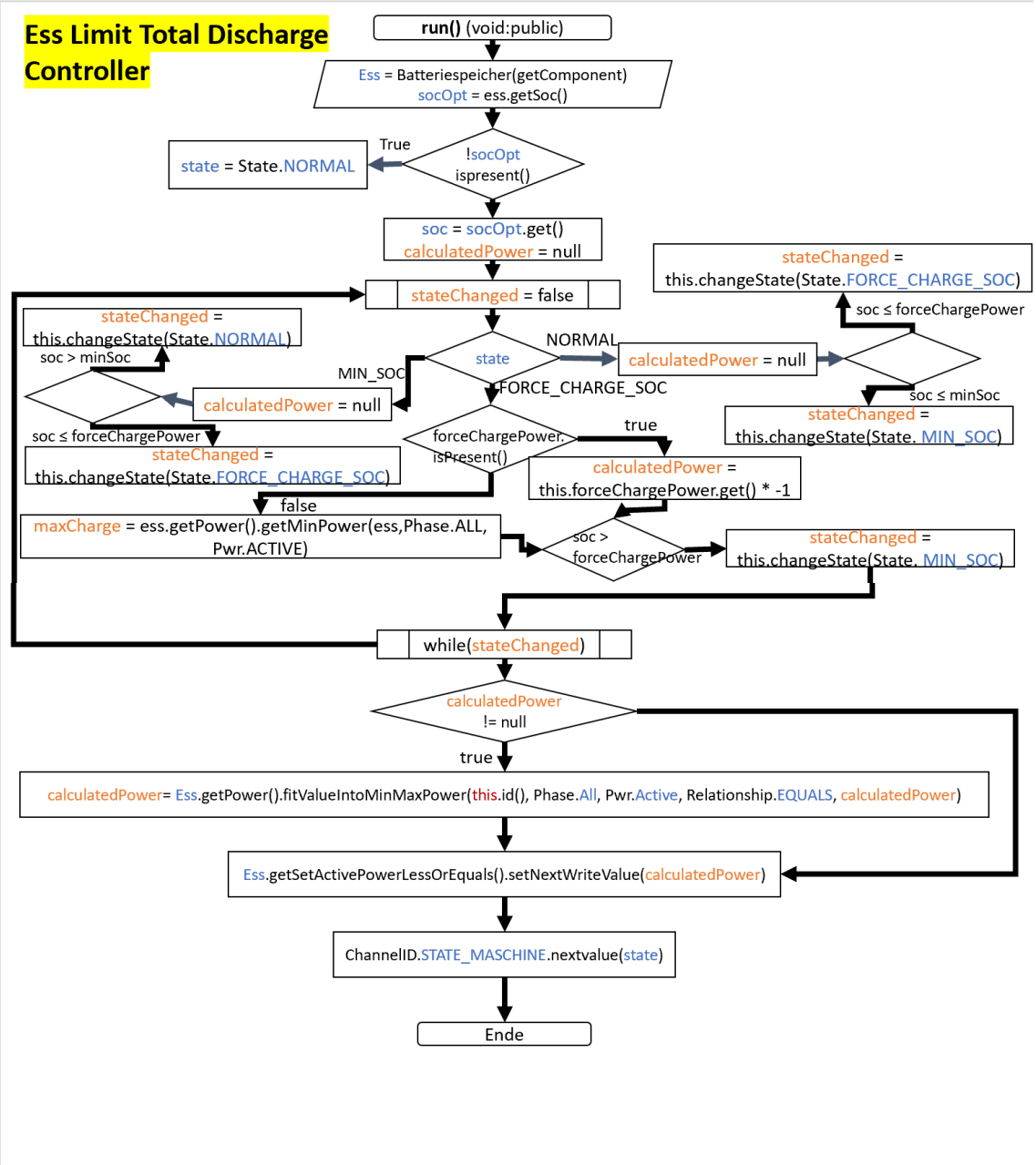


Abbildung 5.7 Programmablaufsplan vom Ess Limit Total Discharge

Der Programmablaufsplan der Methode „run ()“ des Programmes Ess Limit Total Discharge Controller ist in Abbildung 2.4 zu sehen. Als erstes erfolgt die Übergabe der Ess-ID des Batteriespeichers über die Funktion „getComponent ()“, zudem wird der SOC Wert vom Batteriespeicher weitergegeben durch die Funktion „getSoc ()“. Die Variable stateChanged wird am Anfang der Do-While Schleife als false deklariert. Daraufhin wird die Enumeration state in einem switch-case abgefragt. Bei dem Fall „NORMAL“ wird calculatedPower gleich null gesetzt. Falls der soc kleiner oder gleich dem ausgewählten forceChargePower ist, wird die Enumeration stateChanged über die Funktion change-State ()“ in den Status „FORCE_CHARGE_SOC“ geändert. Wenn der soc kleiner oder gleich dem vorher deklarierten minSoc ist, wird der stateChanged auf den Status „MIN_SOC“ geändert. Im Fall „MIN_SOC“ wird die calculatedPower gleich null gesetzt und daraufhin überprüft, ob der soc kleiner ist als der minSoc. Ist der minSoc größer,

wird der stateChanged auf den Status „NORMAL“ geändert. Ist aber der soc kleiner gleich dem forceChargePower, wird der stateChanged auf „FORCE_CHARGE_SOC“ geändert. Im Fall „FORCE_CHARGE_SOC“ wird die Funktion „isPresent ()“ aufgerufen, bei einer true Rückgabe wird die Variable calculatedPower über die Funktion „get ()“ aus der Variable forceChargePower addiert mit dem Inversem. Bei einer false Rückgabe wird die Variabel maxCharge mit der Funktion „getPower ()“ gefolgt von „getMinPower ()“ die als Rückgabe die Leistung, auch der einzelnen Phasen definiert. Sollte der soc kleiner oder gleich der forceChargePower sein, wird die Enumeration stateChanged als „MIN_SOC“ übergeben. Solange sich die Enumeration stateChanged nicht ändert, wird die Do-While Schleife nicht verlassen. Beim Verlassen der Schleife wird überprüft, ob die Variable calculatedPower ungleich null ist, wenn die Bedingung wahr ist, wird calculatedPower über die Funktionen „getPower ()“ gefolgt von „fitValueIntoMinMaxPower ()“ überprüft, ob die minimale bzw. maximale Leistung des Batteriespeichers überschritten worden sind und somit den Wert auf den minimal bzw. maximal Wert anpassen muss. Über die „getSetActivePowerLessorEquals ()“ Funktion wird der Channel der ausgewählt und die Funktion „setNextWrtieValue ()“ überschreibt den Wert im Ess-ID dem Batteriespeicher mit der Variablen calculatedPower. Der Channel „STATE_MASCHINE“ wird über der Funktion „nextValue ()“ der Status der Enumeration state übergeben.

5.3.7 Controller ESS Sell-To-Grid Limit

Der Controller ESS Sell-To-Grid Limit grenzt den Verkauf ins Netz ein, indem der Batteriespeicher mit der überschüssigen Leistung aufgeladen wird. Der Controller kann genutzt werden, um nach den Deutschen Gesetzen die 70% Grenze für die Einspeisung ins Netzt einzuhalten.

- Maximum allowed Sell-To-Grid power [W]

5.3.8 Controller ESS One Full Cyle

Der Controller ESS One Full Cyle Lädt bzw. Entlädt die Batterie nach dem eingegebenen Wert.

- Power [W]

5.3.9 Controller Balancing Symmetric

Der Controller Balancing Symmetric optimiert den Selbs Konsum, indem er versucht den Bezug vom Netz auf null zu halten.

5.3.10 Controller Symmetric Limit Active Power

Der Controller Symmetric Limit Active Power definiert die Auflade- und Entladeleistung unabhängig voneinander.

- Max Charge Power [W]

5.4 Simulation eines realen Haushaltens mit PV-Anlage und Batteriespeicher

Zur Simulation eines realen Haushaltes mit PV–Anlage und einem Energiespeichersystems im OpenEms, muss zuerst ein Scheduler[a] aktiviert werden, da sonst kein Ablauf stattfindet und die Konsole einen Fehler ausgibt. Anschließend wird der Debug Controller[b] aktiviert, der als Konsolenausgabe der Komponenten dient. Beim Hinzufügen des Controllers API Websocket[c] ins System, entsteht eine Verbindung zwischen OpenEMS -UI und -Edge (siehe Abbildung 5.9). Der Simulator Datasource[d] [f] liest CSV-Dateien ein und die Quelldatei(datasource) wird durch den Simulator Produktion Meter Acting genutzt[f], um ein Erzeugerprofil zu erstellen. Der GridMeter Acting [e] ist ein Verbraucherprofil in unserem Beispiel, wobei der GridMeterReacting [i] die Leistung für das Netz regelt. Zudem wurden Controller[j][k][l] genutzt, um die volle Aufladung der Batterie zu verhindern, die Aufladung bei zu hohen Erzeugungsleistung zu stoppen, die Aufladung bei minimaler Batteriekapazität zu erzwingen, sowie die Entladung der Batterie bei zu niedriger Batteriekapazität zu stoppen. Um den richtigen Ablauf zu gewährleisten, musste der Scheduler nochmal neu definiert werden [m]. In Abbildung 3.9 wurde durch das Hinzufügen der Controller-IDs die Ablaufreihenfolge von oben nach unten gefolgt von den nicht genannten Controllern definiert. Wenn der Scheduler nicht neu definiert wird, werden alle Controller nach dem Alphabet ablaufen und es entstehen ungewollte Fehler mit dem Controller [l].

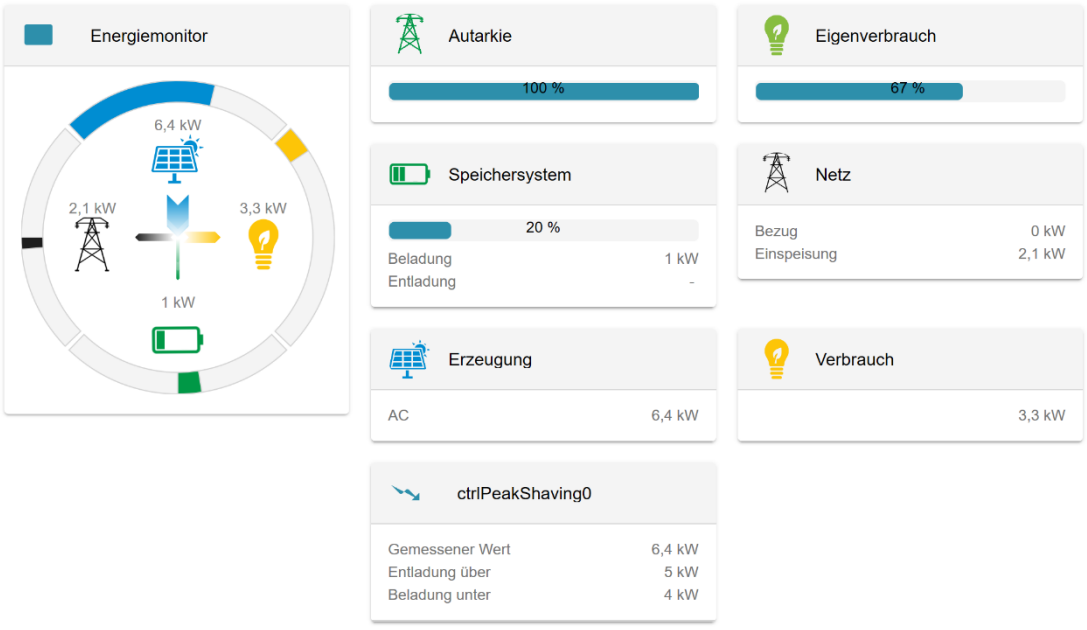


Abbildung 5.8 Nutzerschnittstelle des Energiemanagementsystems

a. Scheduler All Alphabetically [scheduler0]
b. Controller Debug Log [ctrlDebugLog0]
c. Controller Api Websocket [ctrlApiRest0]
d. Simulator Datasource: CSV Predefined [datasourceConsum]
e. Simulator GridMeter Acting [meterConsum]

f. Simulator Datacsource: CSV Direct[datasourcePV]
g. Simulator PrduktionMeter Acting [meterPV]
h. Simulator EssSymmetic Reacting [essEnergy]
i. Simulator GridMeter Reacting [meterGrid]
j. Controller Peak-Shaving Symmetric [ctrlPeakShaving0]
k. Controller Fix-Active Power Symmetric [ctrlFixActivePower0]
l. Controller ESS Limit Total Discharge [ctrlLimitTotalDischarge0]
m. Scheduler All Alphabetically [scheduler0]

Neue Ordnung für den Algorithmischen Ablauf der Controller-IDs:

- ctrlLimitTotalDischarge0
- ctrlPeakShaving0
- ctrlFixActivePower0
- ctrlDebugLog0

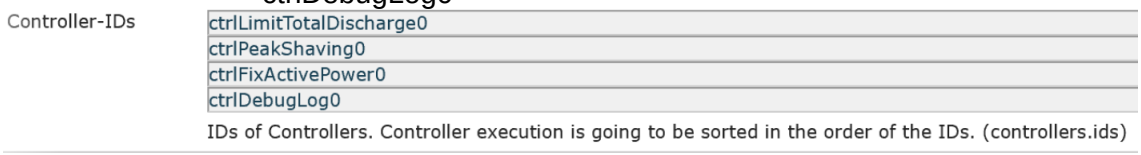


Abbildung 5.9 Algorithmische Ablauf der Controller

6 Entwicklung der Simulationen & Controller

Die Modellierung der Simulation zur Nutzung einer PVT-Anlage in der Open Source Plattform OpenEMS benötigt mehrere Simulatoren. Je mehr Komponenten dem System hinzugefügt werden, desto genauer wird die Analyse für das PVT-Modell.

6.1 Simulator Datasource Predef

Der bestehende Simulator Datasource Predef wurde für den Vorhersage-Effekt umgeschrieben. Nun beinhaltet die Simulation eine extra Variable namens foresight, die angibt, wie viele Werte für eine Prognose ausgelesen werden sollen. Als Default wird der Wert 1 übergeben, der keinen weiteren Wert einließt. Bei unserer Modellierung wird der Wert auf 24 gesetzt, damit ein gemittelter Tageswert übergeben wird. In dem Projekt CsvDatasourcePredefined wird in der methode getData() die Funktion .readcsvFIle-FromResource(..) ausgeführt. Zuerst wurde eine neue Funktion mit demselben Namen erstellt, jedoch mit einem Zusatze „int“ Variable namens „foresight“. Im nächsten Bild sehen wir die alte Funktion ohne Foresight (siehe Abb. 6.1). Die Funktion mit Foresight wird durch die Zeilen 49-52 in der Abbildung 6.2 ergänzt und die Variablen werden für die foresight Kalkulation initialisiert. Zudem wurde in Zeile 52 wird eine Liste in der Größe der Foresight Variable erstellt. Anschließend wurde die Funktion readRecord von einer Void in eine Fload Array umgeändert, da der Wert von der CSV nun zurückgegeben werden muss und in die float array Variable „floatValues“ eingespeichert wird. Nun wird diese temporäre Variable in die vorher erstellte Liste „fifoValue“ über die Funktion „add“ in die Liste fifoValue hinzugefügt und auf die Variable fifoSum aufaddiert. Dieser Ablauf findet so lange statt, bis die Liste die Größe von foresight erreicht hat. Ist die Größe erreicht, wird die Summe der Werte fifoSum mit der Anzahl der Werten foresight dividiert, um den Mittelwert zu berechnen. Dieser wird nun über die Funktion „addRecord“ in ein DataContainer eingespeichert, wie in Abb. 6.1. Zuletzt wird der Wert, der als erstes in die Liste eingefügt wurde, über die Funktion „poll“ entfernt. Somit kann die Schleife fortgeführt werden und wieder über die Funktion “add“ den neuen Wert ans Ende der Liste einfügen.

```
24      public static DataContainer readCsvFileFromResource(Class<?> clazz, String filename, CsvFormat csvFormat,
25              float factor) throws NumberFormatException, IOException {
26          DataContainer result = new DataContainer();
27          boolean isTitleLine = true;
28          String line = null;
29          try (BufferedReader br = new BufferedReader(new InputStreamReader(clazz.getResourceAsStream(filename)))) {
30              while ((line = br.readLine()) != null) {
31                  if (isTitleLine) {
32                      // read titles
33                      readTitles(result, csvFormat, line);
34                      isTitleLine = false;
35                      continue;
36                  }
37                  // start reading the values, parse them to doubles and add them to the result
38                  readRecord(result, csvFormat, factor, line);
39              }
40          }
41          return result;
42      }
```

Abbildung 6.1: Codeabschnitt, der Funktion readCsVFileFromResource

fifoSum = Summe der eingelesenen Werte der CSV Datei

floatValues = Die aktuelle Datei die von der CSV-Datei temporär abgespeichert wird.

fifoValue = Liste der eingelesenen CSV Werte (in der Größe von foresight)

readRecord hat eine Rückgabe wert float Array und übergibt diese in floatValues.

```
44  public static DataContainer readCsvFileFromResource(Class<?> clazz, String filename, CsvFormat csvFormat,
45          float factor, int foresight) throws NumberFormatException, IOException {
46      DataContainer result = new DataContainer();
47      boolean isTitleLine = true;
48      String line = null;
49      int buffer = 0;
50      Float[] floatValues = new Float[buffer];
51      float fifoSum = 0;
52      EvictingQueue<Float> fifoValue = EvictingQueue.create(foresight);
53      try (BufferedReader br = new BufferedReader(new InputStreamReader(clazz.getResourceAsStream(filename)))) {
54          while ((line = br.readLine()) != null) {
55              if (isTitleLine) {
56                  // read titles
57                  readTitles(result, csvFormat, line);
58                  isTitleLine = false;
59                  continue;
60              }
61              // start reading the values, parse them to doubles and add them to the result
62              floatValues = readRecord(result, csvFormat, factor, line, foresight);
63              fifoValue.add(floatValues[buffer]);
64              fifoSum += floatValues[buffer];
65              if(fifoValue.size() == foresight)
66              {
67                  floatValues[buffer] = fifoSum / foresight;
68                  result.addRecord(floatValues);
69                  fifoSum -= fifoValue.poll();
70              }
```

Abbildung 6.2: Codeauschnitt, der Funktion readCsvFileFromResource mit foresight variable

```
155 private static Float[] readRecord(DataContainer result, CsvFormat csvFormat, float factor, String line, int foresight)
156     String[] values = line.split(csvFormat.lineSeparator);
157     Float[] floatValues = new Float[values.length];
158     for (int i = 0; i < values.length; i++) {
159         String value = values[i];
160         if (value == null || value.isEmpty()) {
161             value = null;
162         } else {
163             if (csvFormat.decimalSeparator != ".") {
164                 value = value.replace(csvFormat.decimalSeparator, ".");
165             }
166             floatValues[i] = Float.parseFloat(value) * factor;
167         }
168     }
169     return floatValues;
170 }
```

Abbildung 6.3: Programmausschnitt der Funktion readRecord

6.2 HeatStorage

6.2.1 API

Die API vom HeatStorage besitzt folgende Channels:

- ACTIVE_POWER (Integer/Watt)
    - Beim Entladen negativer Wert, beim Laden positiver Wert
- AMBIENT_TEMPERATURE (Integer/Degree_Celsius)
    - Temperaturdurschnitt der nächsten 24h, eingelesen über CSV-Datei
- CAPACITY (Integer/Watt_Hours)

  - Aktuelle Wärmeenergie des Pufferspeichers
- MAX_CAPACITY (Integer/Watt_Hours)
  - Maximale Wärmeenergie des Pufferspeichers
- MEMORY_LEVEL (Integer/Percent)
  - Aktuelle Ladezustand vom Pufferspeicher

6.2.2 Porgrammablauf des Simulators

Der Simulator HeatStorage besitzt fünf Initialisierung Variablen. Diese werden im AF-WCC durch Eingabe der Config-Werte initialisiert. Die maximale Temperatur (Tnutz) im Wärmespeicher und die minimale Temperatur (Tmin) des Wärmespeichers. Der Füllstand des Wärmespeichers (MemoryLevel), das Volumen (Volumen) in Liter bzw. in Kubikmeter und die CSV-Datei die die Außentemperatur (Datasource) übergibt. Beim Aktivieren des Wärmespeichers wird das Volum (HIghvolume) von der oberen Schicht berechnet. Die obere Schicht beinhaltet die maximale Temperatur und dient zur Wärmeaufnahme bzw. Wärmeabgabe. Nachdem das Volumen der maximalen Temperatur ermittelt wurde, wird die Energie im Speicher ausgerechnet (siehe EnergyStorage in Abb. Simulator HeatStorage).

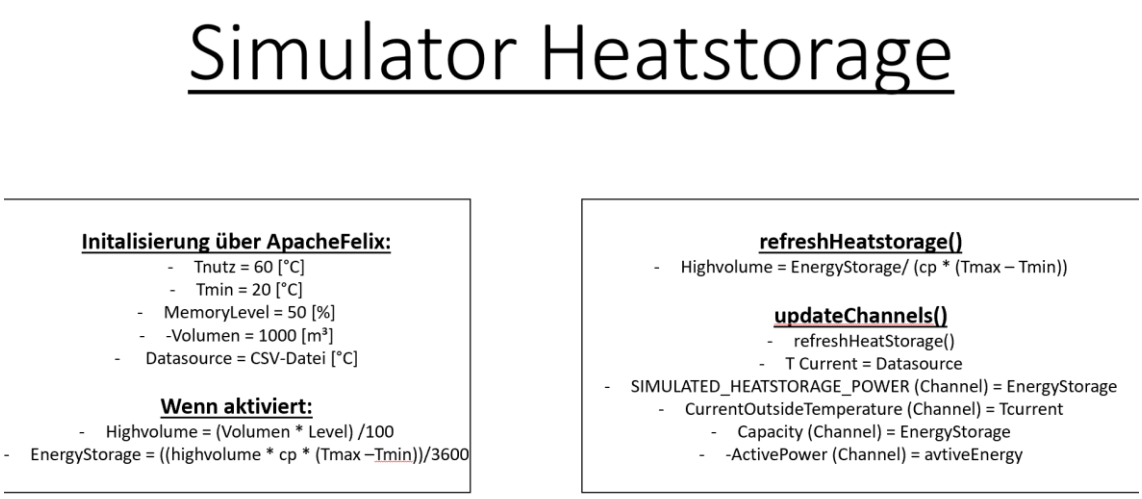


Abbildung 6.4: Simulator HeatStorage

Die Methode updateChannels () wird als Main Methode genutzt und führt die Funktion refreshHeatStorage () aus. Der refreshHeatStorage () errechnet anhand der vorhandenen Energie im Speicher das Volumen (Highvolume) der oberen Schicht. Das Volumen

der oberen Schicht dividiert durch das gesamte Volumen ergibt den Füllstand bzw. Füll-level des Wärmespeichers. Weiter in updateChannels (), wird der Energiegehalt, sowie der Füllstand an die Channels übertragen.

6.3 Simulator Heatpump

6.3.1 API

Die API vom HeatStorage besitzt folgende Channels:

- MODE ([ON/OFF])
    - Der Zustand der Wärmpumpe. Bei „ON“ Abgabe der Wärmeleistung.
- COP (Integer)
    - Aktueller COP Wert
- ACTIVE_POWER (Integer/Watt)
    - Beim Entladen negativer Wert, beim Laden positiver Wert
- HEAT_POWER (Integer/Watt)
    - Erzeugte Wärmeenergie der Wärmepumpe
- AMBIENT_TEMPERATURE (Integer/Percent)
    - Aktuelle Außentemperatur über CSV-Datei eingelesen

6.3.2 Programmablauf des Simulators

Der Simulator Heat pump benötigt 4 variablen zur Initialisierung. Die maximale Temperatur von 60°C die durch das Einschalten der Wärmepumpe als Peak-Wert der zu erhitzenden Temperatur gilt. Die Maximale Leistung von 5000W, die bei der Nutzung der Wärmepumpe verbraucht wird und die verschiedenen Zustände AN oder AUS mittels der Variable MODE. Eine CSV-Datei die die Außentemperatur der nächsten Stunde gemittelt übergibt.

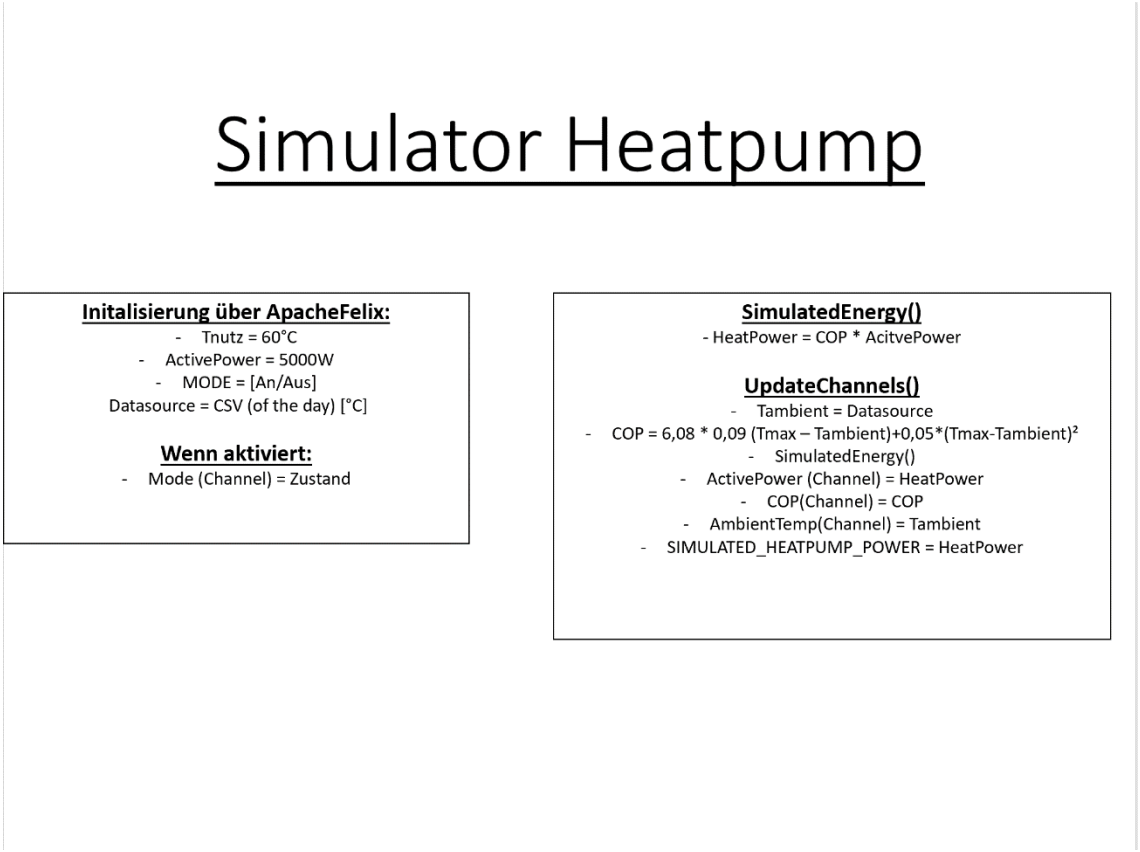


Abbildung 6.5: Simulator Heatpump

Beim Aktivieren der Wärmepumpe wird der Zustand (MODE) in den Channel MODE übergeben, damit andere Komponenten auch Zugriff auf den Zustand der Wärmepumpe haben. Über die Methode UpdateChannels () wird der Integer Wert der CSV-Datei (Datasource) an die Variabel Tambient übergeben. Der COP wird nach der Literatur berechnet [3]. Die Funktion SimulatedEnergy () wird aufgerufen und berechnet die erzeugte Wärmeleistung, indem der COP Wert mit dem ActivePower multipliziert und die Variable HeatPower übergeben wird. Zudem übergibt die Methode updateChannels () auch noch die Variablen MaxLeistung, COP, Tambient und Heatpower an die Channels des Simulators weiter.

6.4 Controller HeatMode

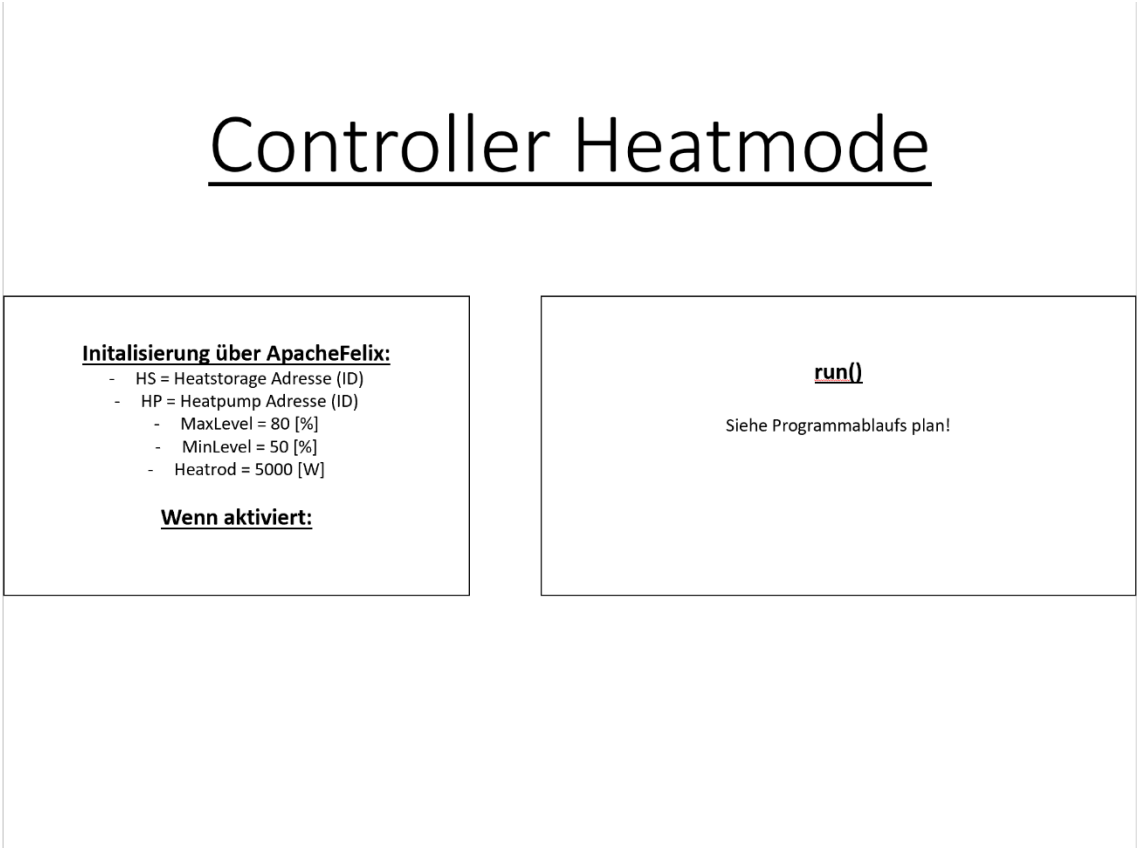


Abbildung 6.6:Controller HeatMode

Bei der Main Methode run() wird überprüft, ob die mittlere Temperatur des Wärmespeichers größer als die aktuell entnommene Außentemperatur der Wärmepumpe ist. Wenn es stimmt, wird überprüft, ob die maximale Grenze der Kapazität vom Controller kleiner ist als die aktuelle Kapazität des Wärmespeichers. Ist dies der Fall, wird dem Wärmespeicher die Verlustleistung des Haushaltes abgezogen. Wenn der Fall nicht eintritt, wird überprüft, ob der untere Grenzwert des Controllers für die Kapazität kleiner ist als die aktuelle Kapazität des Wärmespeichers. Wenn es stimmt, wird die Wärmepumpe eingeschaltet und die erzeugte Wärmeleistung wird mit der Verlustleistung des Haushaltes der Speicherkapazität übergeben. Ansonsten wird der Heizstab mit aktiviert, um den Speicher, der unter der unteren Grenze ist, wieder aufzuladen. Wenn die mittlere Temperatur des Wärmespeichers kleiner ist als die aktuell entnommene Außentemperatur der Wärmepumpe, wird überprüft, ob die aktuelle Wärmekapazität kleiner ist als der untere Grenzwert des Controllers. Stimmt es, wird überprüft, ob die Verlustleistung des Haushaltes größer ist als die erzeugte Wärmeleistung, wenn dies nicht stimmt, wird der Heizstab eingeschaltet. Bei beiden Fällen schaltet sich die Wärmepumpe ein. Wenn die untere Grenze des Controllers kleiner ist, wird überprüft, ob der aktuelle Wärmekapazität Gehalt größer ist als der obere Grenzwert des Controllers, wenn es stimmt, wird die Verlustleistung vom Wärmespeicher abgezogen und die Wärmepumpe bleibt ausgeschaltet. Ist dies nicht der Fall wird die Wärmepumpe eingeschaltet und mit der Verlustleistung zusammen dem Wärmespeicher eingefügt.

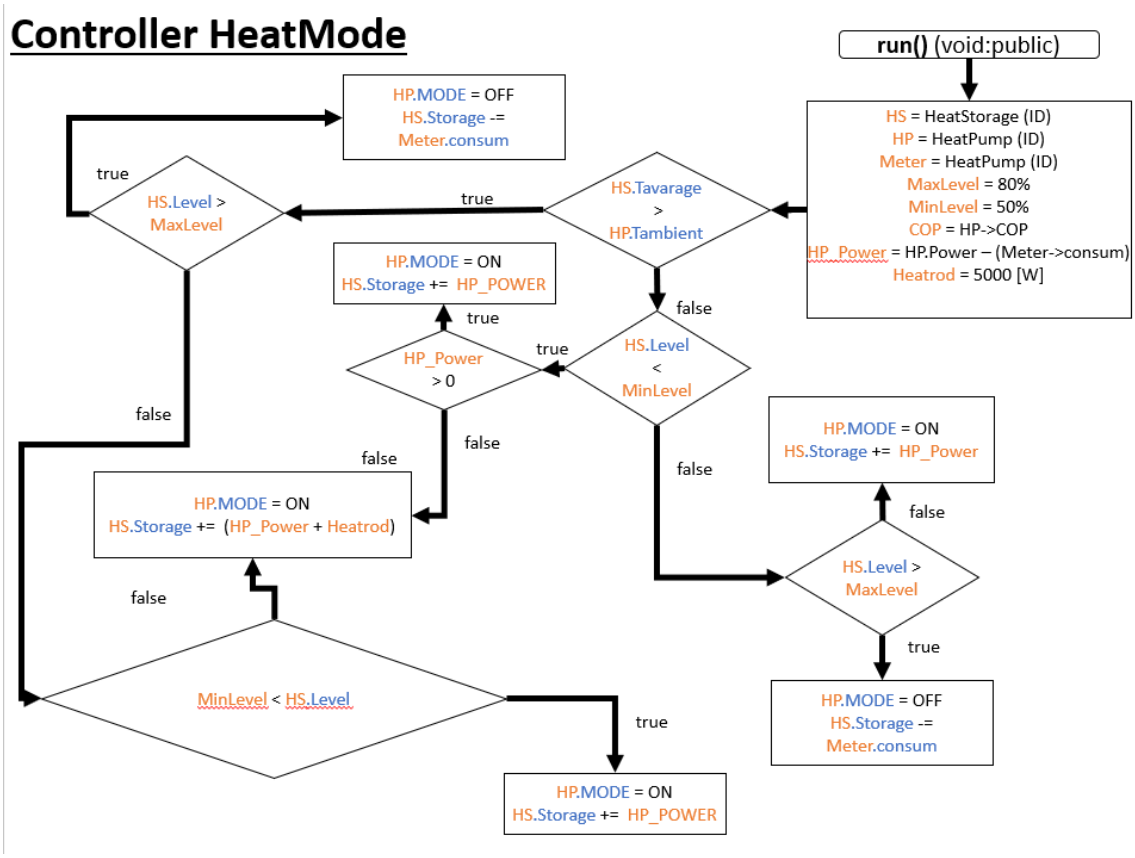


Abbildung 6.7: Controller HeatMode Programmablaufs Plan [1]

7 Auswertung

In dem Kapitel Auswertung werden die Anhand der programmierten Controller und der zur Verfügung stehenden Quelldateien, die als Erzeuger und Verbraucher dienen mit einer Excel Kalkulation Tabelle berechnet und als Diagramm wiedergegeben. Die Diagramme sind alle in der X-Achse zeitabhängig und beinhalten Werte, um eine Simulation bis hin zum Jahr 2040 durchzuführen. In unserem Beispiel wird die X-Achse jedoch in 3 Tagesabständen in einem Monat dargestellt.

- Active Power = Errechnete Leistung durch den Controller
- Heat Power = Eruzeugte Wärmeleistung der Wärmepumpe
- Consum Power = Verlustleistung des Haushaltes (konstant)
- Mode = Der Zustand der Wärmepumpe
- Heatrod = Der Zustand des Heizstabes

7.1 Auswertung mit 0% Kapazität & hohen Temperaturen

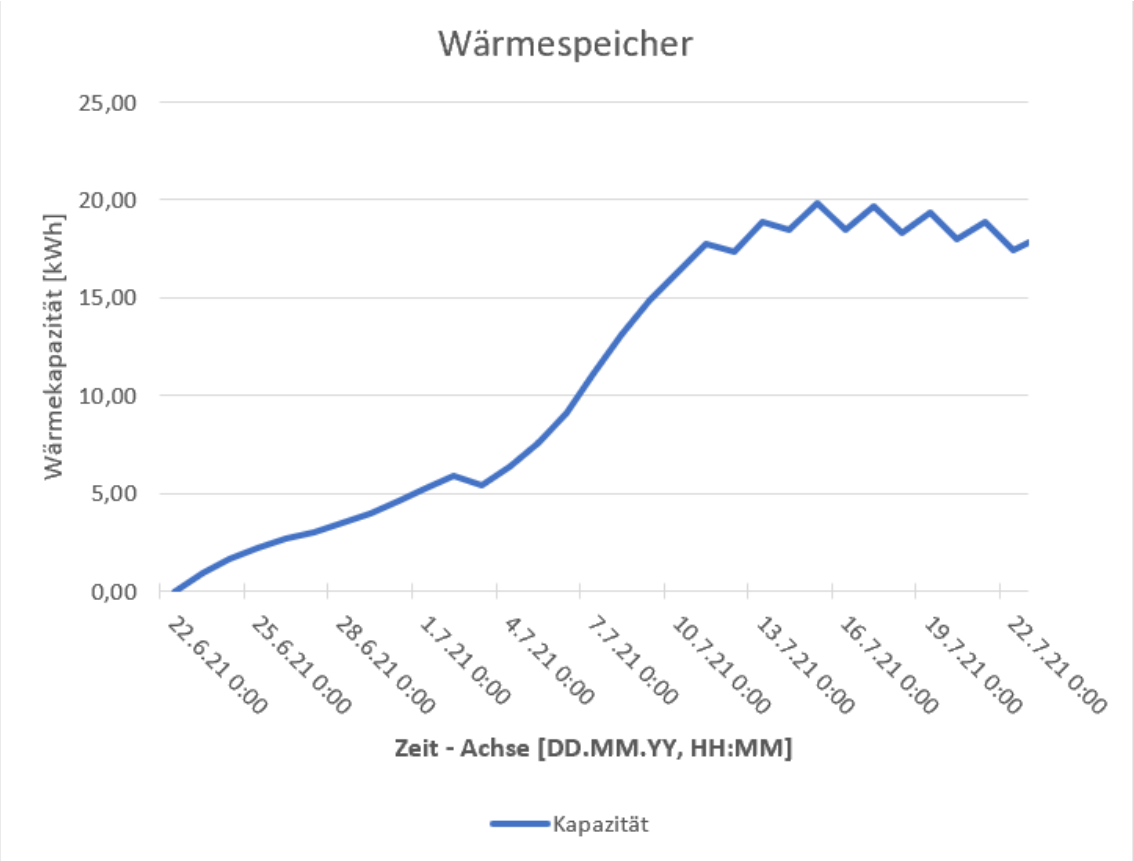


Abbildung 7.1: Liniendiagramm von der Wärmekapazität des Wärmespeichers

Anhand der Abbildung 7.1 erkennt man, dass der Wärmespeicher keine Wärmekapazität abgeben kann und Leistung aufnimmt, der Controller schaltet die Wärmepumpe bis zum 13.Juni komplett an (siehe Abb. 7.4). Mitte des Monats Juni erkennt man, dass der Wärmespeicher das Maximum von 23,23 kWh nicht erreicht, da der Controller nur erlaubt, dass er zu 80% geladen werden darf. Anhand der hohen Temperaturen, erkennbar bei

Abbildung 7.3, ist dementsprechend auch der COP Wert hoch (siehe Abb. 7.5). Der Controller versucht aufgrund der hohen Temperaturen den Pufferspeicher durch die Wärmepumpe zu laden, da es energieeffizienter ist den Pufferspeicher geladen zu halten, wenn die Temperaturen hoch sind. An der Abbildung 7.4 kann man ebenfalls auch den Zustand des Heizstabes erkennen. Der hohe COP-Wert ermöglicht es durch die Wärmepumpe den Eigenbedarf des Haushaltes zu decken.

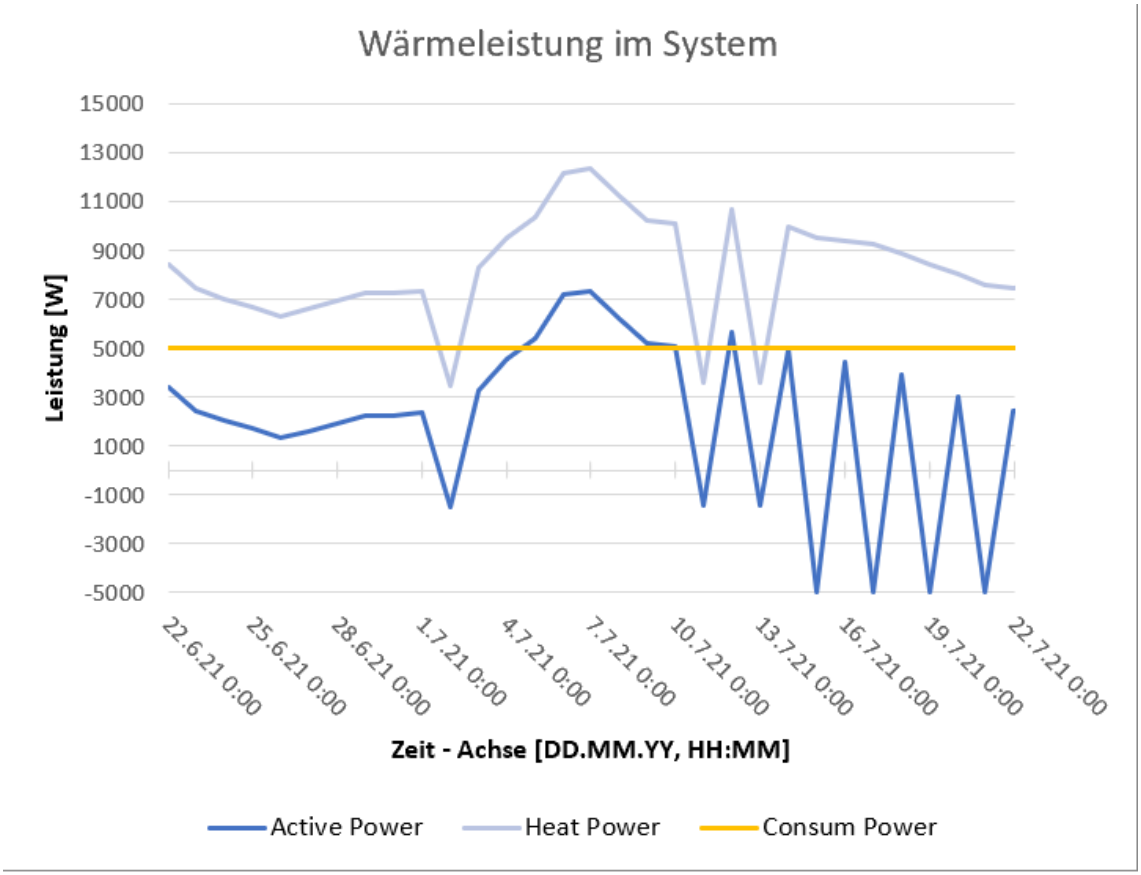


Abbildung 7.2: Liniendiagramm von der Leistung im System

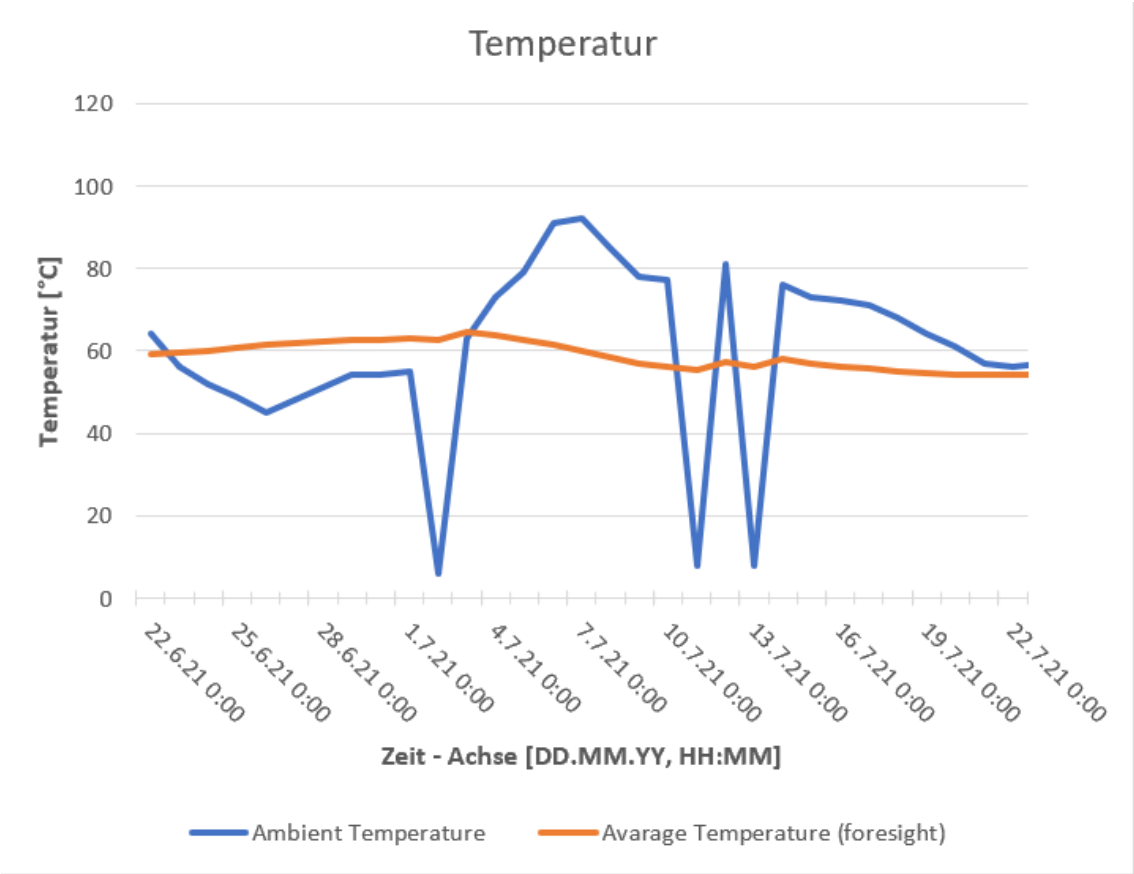


Abbildung 7.3: Liniendiagramm der Temperatur (hohe Temperaturen)

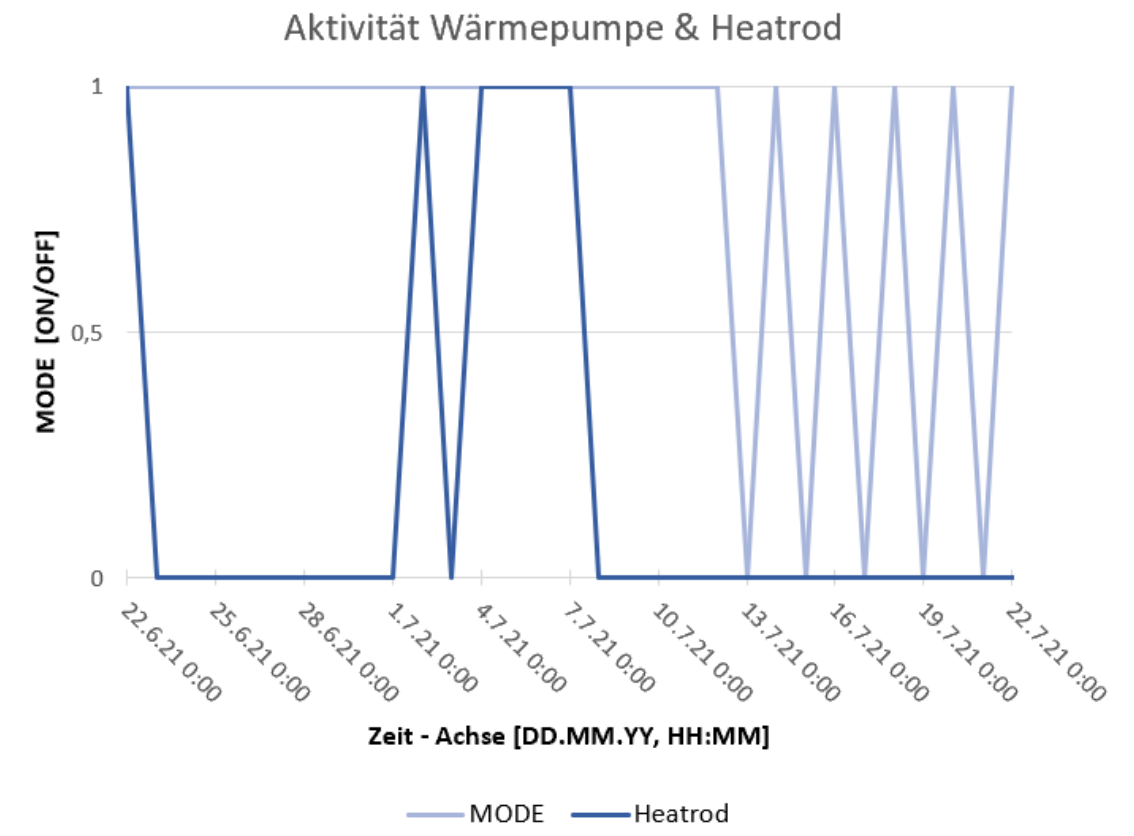


Abbildung 7.4: Liniendiagramm des Zustandes der Wärmepumpe und vom Heizstab

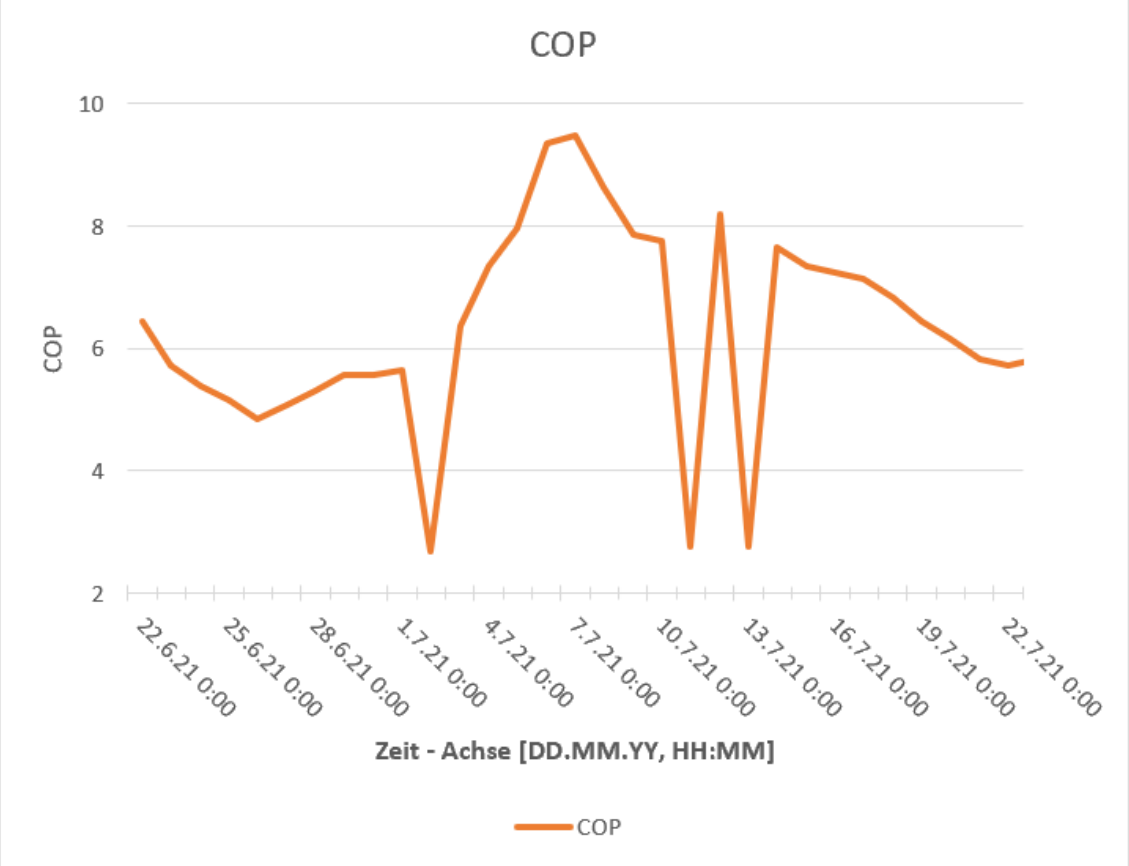


Abbildung 7.5: Liniendiagramm des COP

7.2 100% Kapazität mit hohen Temperaturen

Anhand des maximal befüllten Wärmespeichers verhindert der HeatMode Controller eine weitere Aufladung und zwingt den Wärmespeicher sich zu entladen. Am Anfang der Abbildung 7.6 wird die Verlustleistung vom Wärmespeicher abgezogen, bis die Wärmekapazität unter 80% fällt, ab dem Zeitpunkt wird die Wärmepumpe wieder bei Bedarf angeschaltet (siehe Abb.7.7). Da die Temperaturen sehr hoch sind und der Wärmespeicher über halb der 50% Grenze ist, wird der Heizstab nicht eingeschaltet. In den Abbildungen 7.1 & 7.8 ist gut zu erkennen, der Speicher nachdem er sich be- bzw. entladen hat sich nahe im oberen Grenzwert befindet, da die Temperatur Bedingungen es zulassen den Wärmespeicher mit der Wärmepumpe aufzuladen.

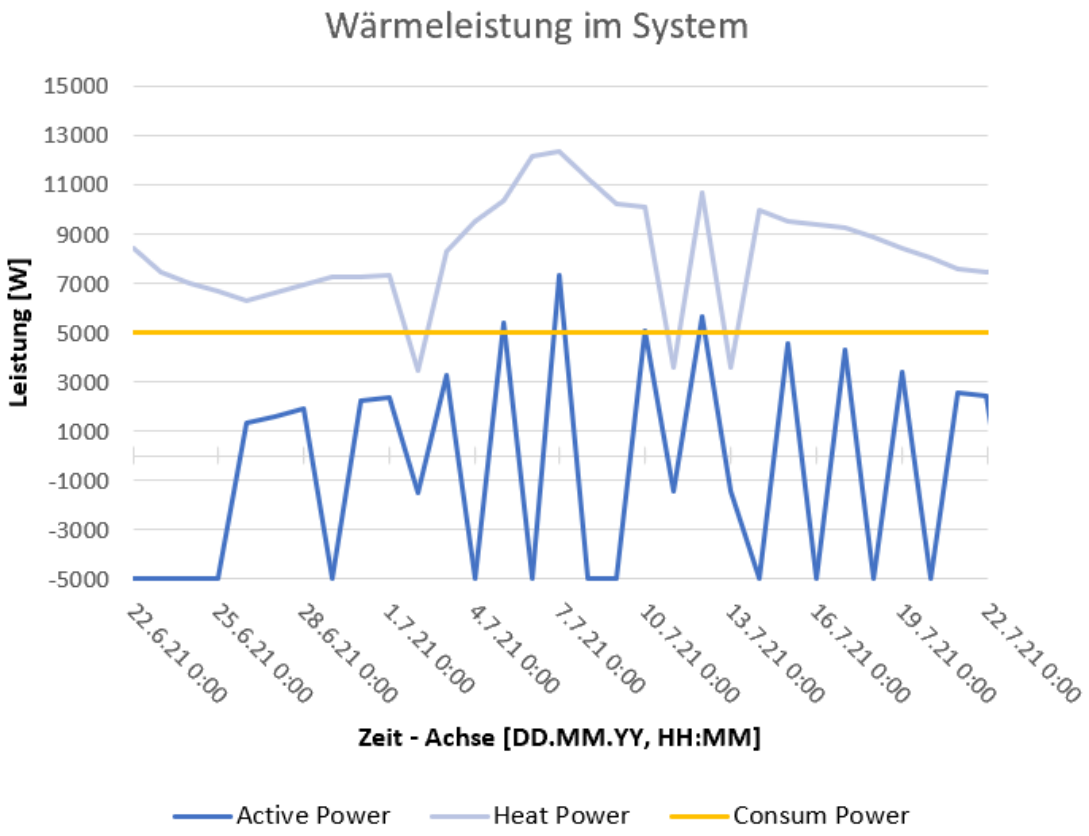


Abbildung 7.6: Liniendiagramm der Wärmeleistung im System

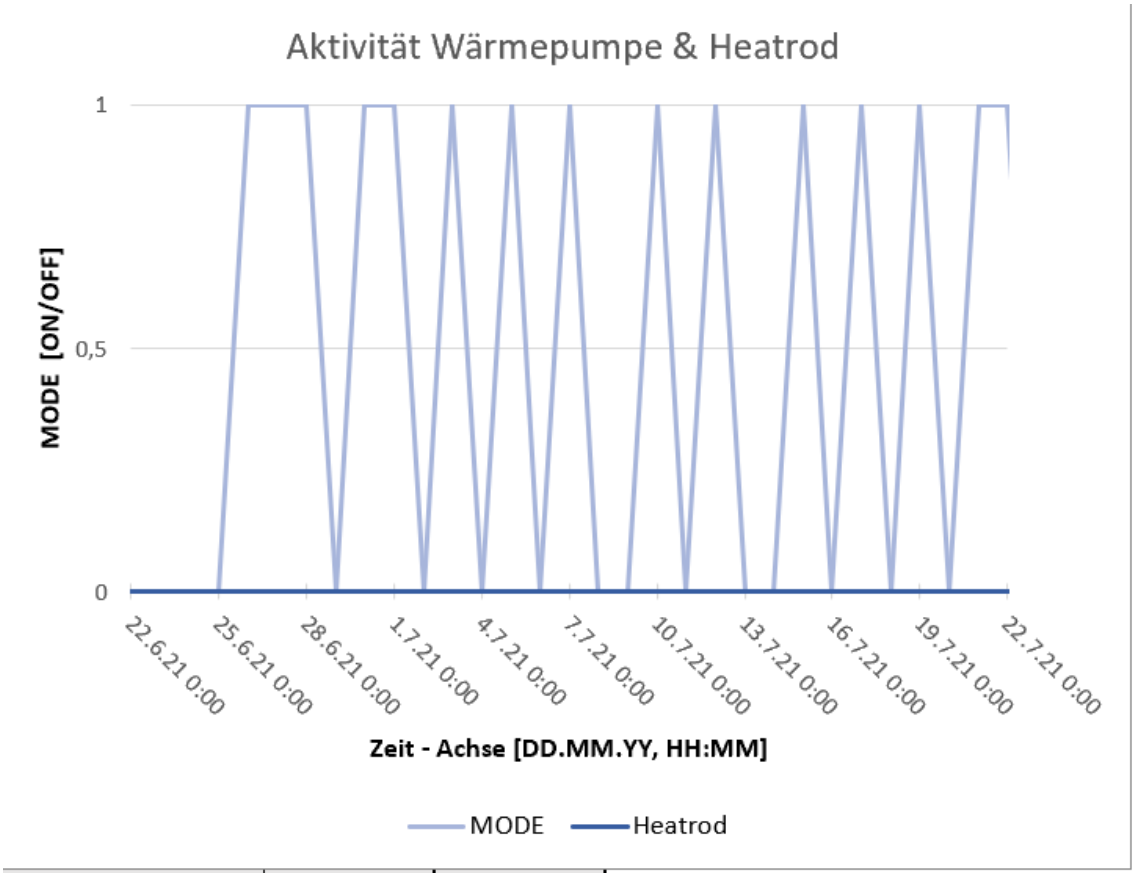


Abbildung 7.7:Liniendiagramm des Zustandes der Wärmepumpe & Heizstab

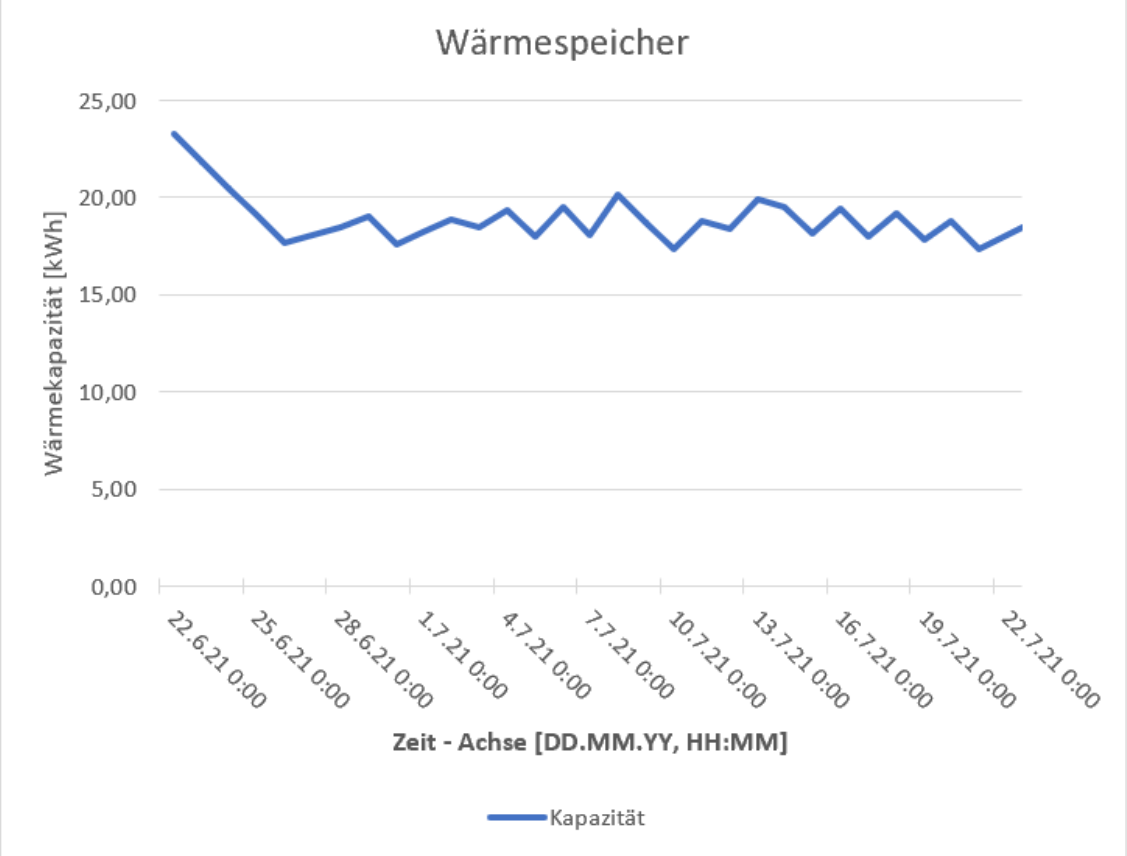


Abbildung 7.8: Liniendiagramm der Wärmekapazität des Wärmespeichers mit 100% anfangs Kapazität

7.3 0% Kapazität mit niedrigen Temperaturen

Bei einer Wärmekapazität von 0% und niedrigen Temperaturen erkennt man in der Abbildung 7.9, wie der Wärmespeicher sich am Anfang schnell durch den Heizstab auflädt (siehe Abb. 711) und später im Verlauf nahe dem unteren Grenzwert von 50% verbleibt. Dies liegt an den niedrigen Temperaturen, da der COP Wert gering ist und die Wärmepumpe nicht so viel Leistung erbringen kann, dazu kommt, dass eine Aufladung des Wärmespeichers bei niedrigen Temperaturen nicht so energieeffizient ist, als wenn es bei höheren Temperaturen der Fall ist. Die Wärmepumpe ist kontinuierlich angeschaltet, da die Verlustleistung des Haushaltes nicht über erzeugte Wärmeleistung der Wärmepumpe nicht gedeckt werden kann und somit sich der Wärmespeicher mit der Zeit weiter entlädt aber der Heizstab ihn davon abhält unter den unteren Grenzwert von 50% zu fallen.

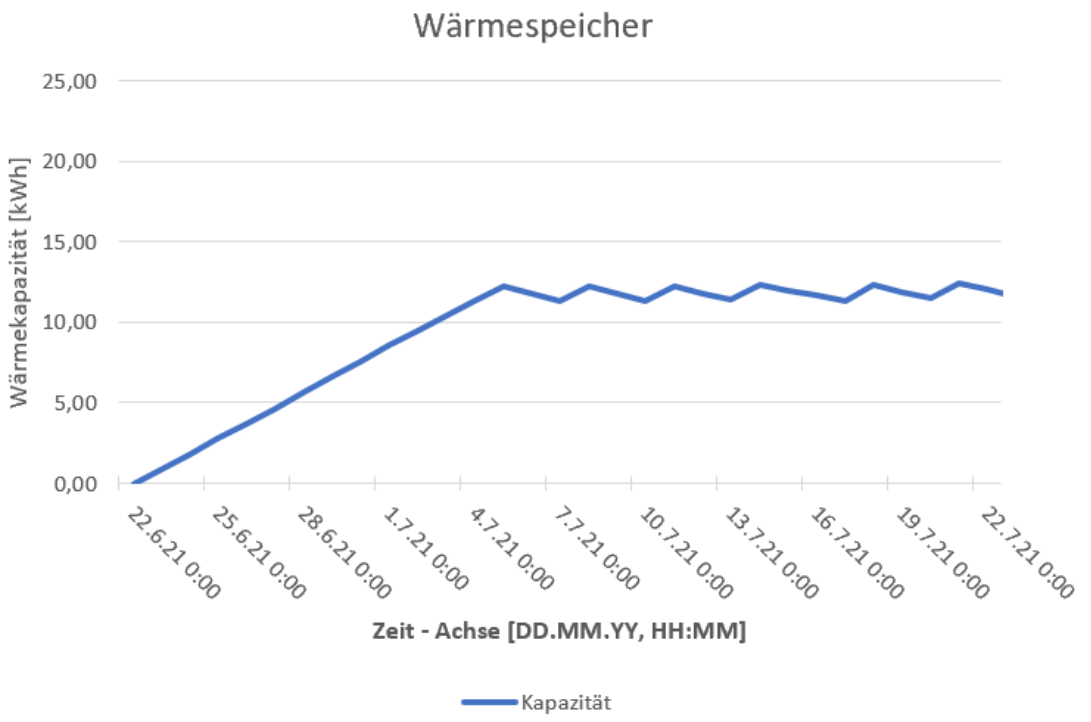


Abbildung 7.9: Liniendiagramm des Wärmespeichers mit 0% Anfangskapazität

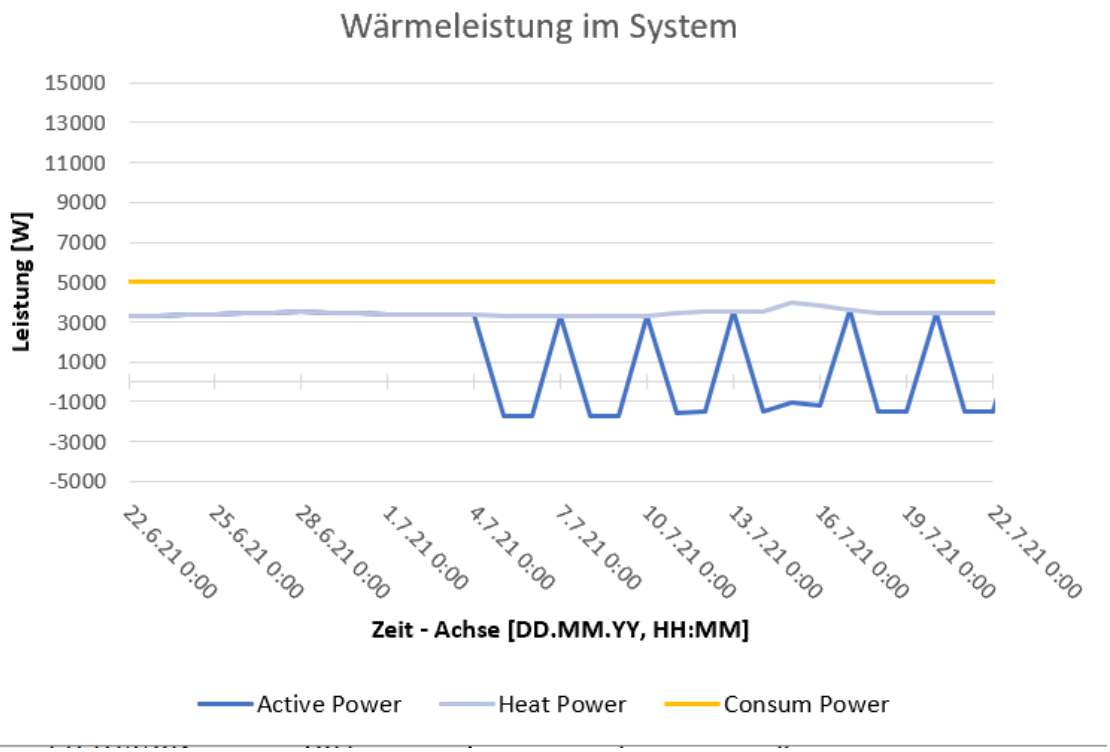


Abbildung 7.10: Liniendiagramm der Wärmeleistung im System

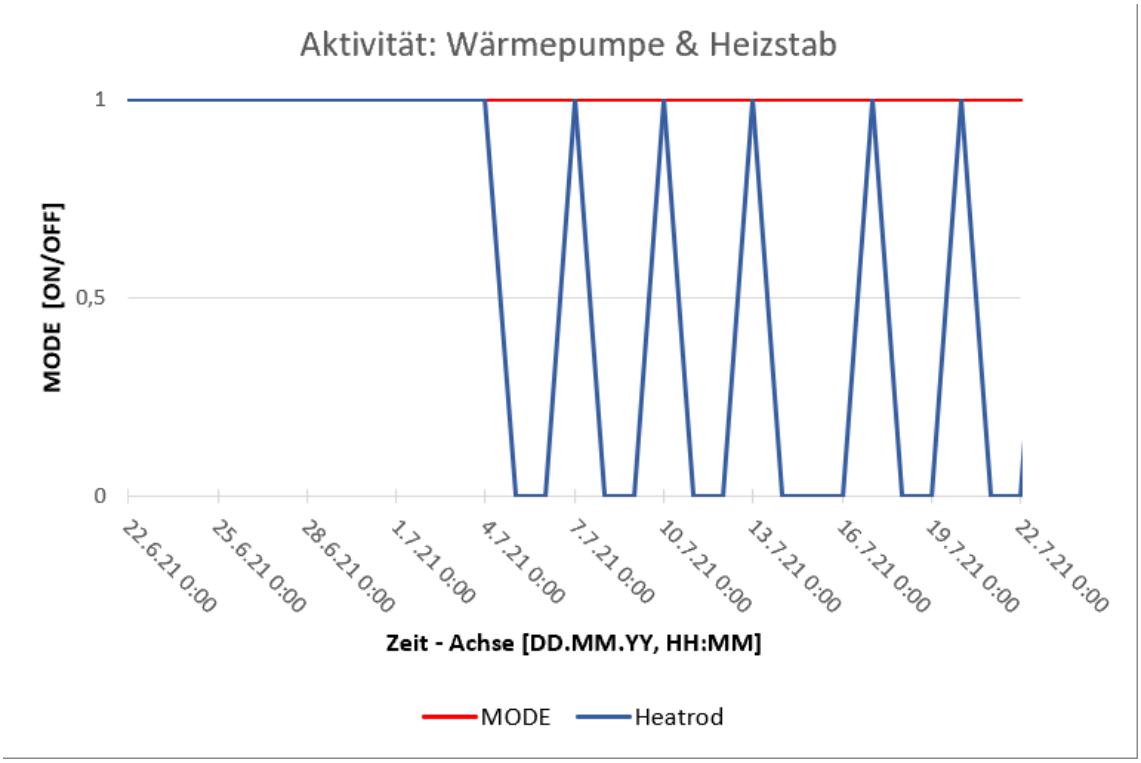


Abbildung 7.11: Liniendiagramm des Zustandes der Wärmepumpe & Heizstab

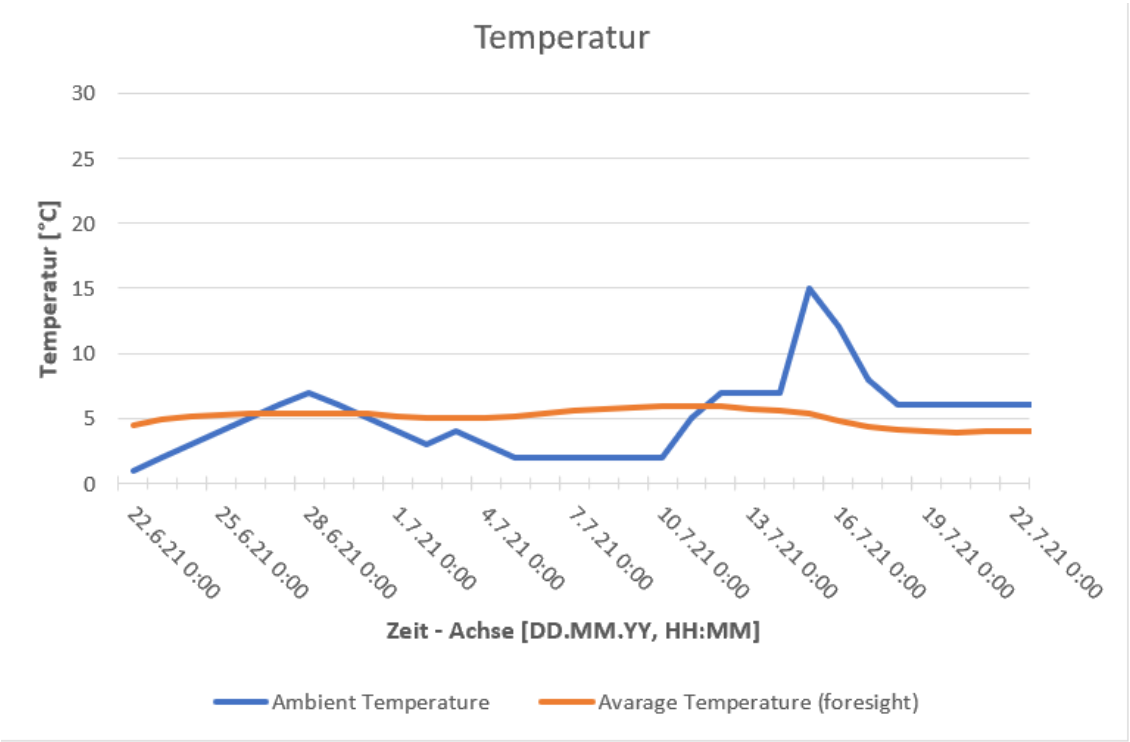


Abbildung 7.12: Liniendiagramm der Temperatur (niedrige Temperaturen)

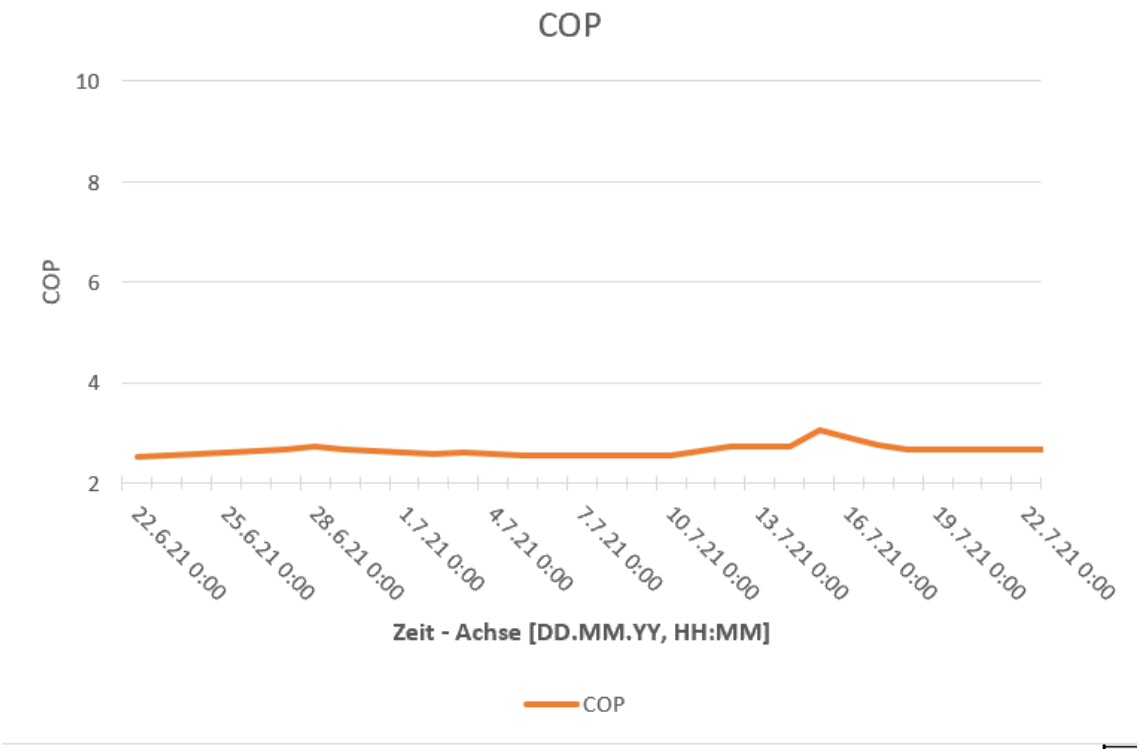


Abbildung 7.13: Liniendiagramm des COP bei niedrigen Temperaturen

7.4 100% Kapazität mit niedrigen Temperaturen

In den letzten Darstellungen erkennt man auf den Diagrammen Abb. 7.16, dass sich der Wärmespeicher bis zu der unteren Grenze von 50% entlädt. Daraufhin schaltet sich Heizstab wieder an (siehe Abb.7.14), um die Verlustleistung des Haushaltes zu kompensieren und den Wärmespeicher aufzuladen. Bei niedrigen Temperaturen verbleibt auch hier der Wärmespeicher nahe der unteren 50% Grenze (siehe Abb. 7.16).

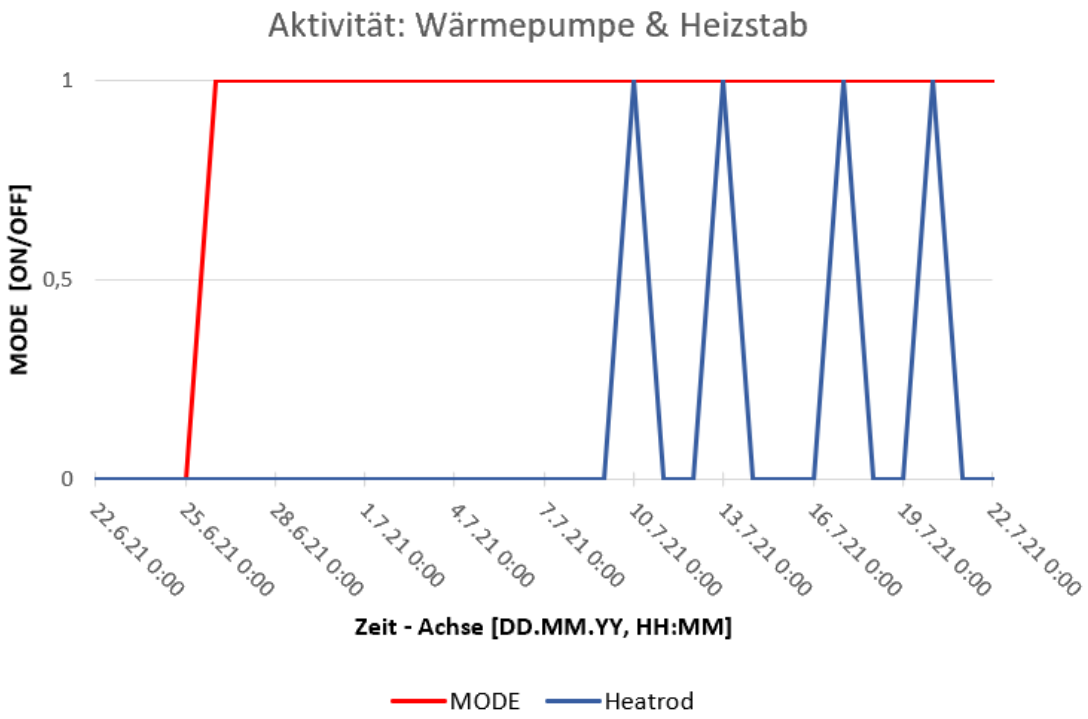


Abbildung 7.14: Liniendiagramm des Zustandes der Wärmepumpe & Heizstab

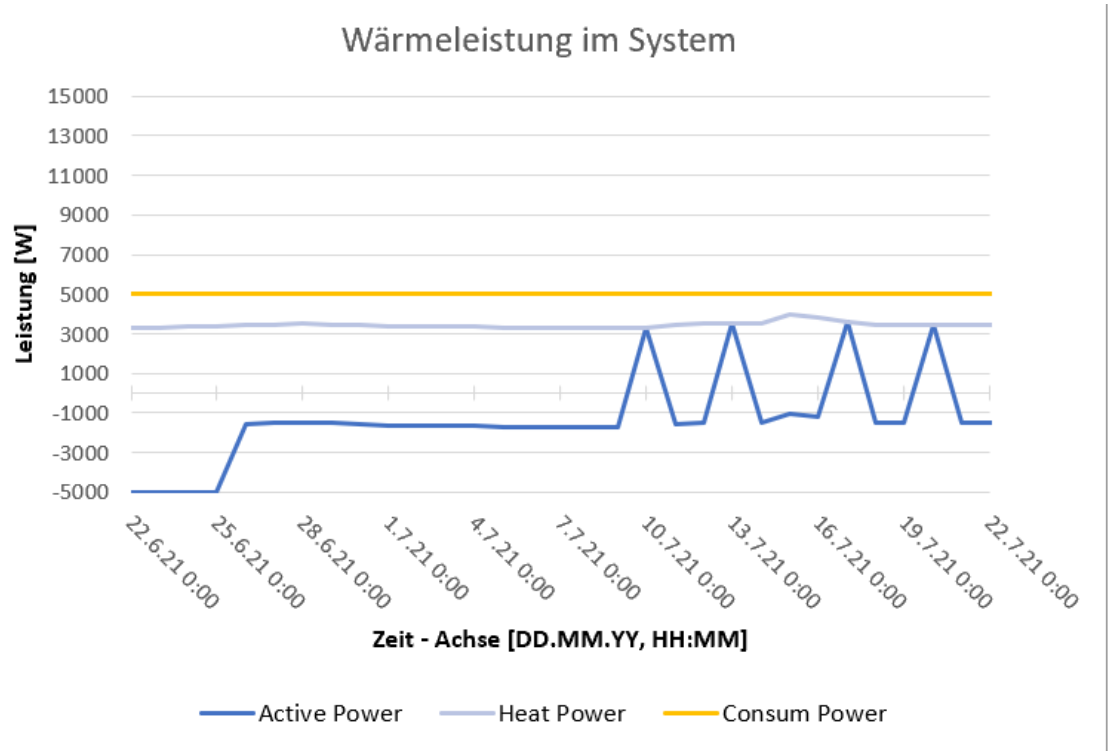


Abbildung 7.15: Liniendiagramm der Leistung im System

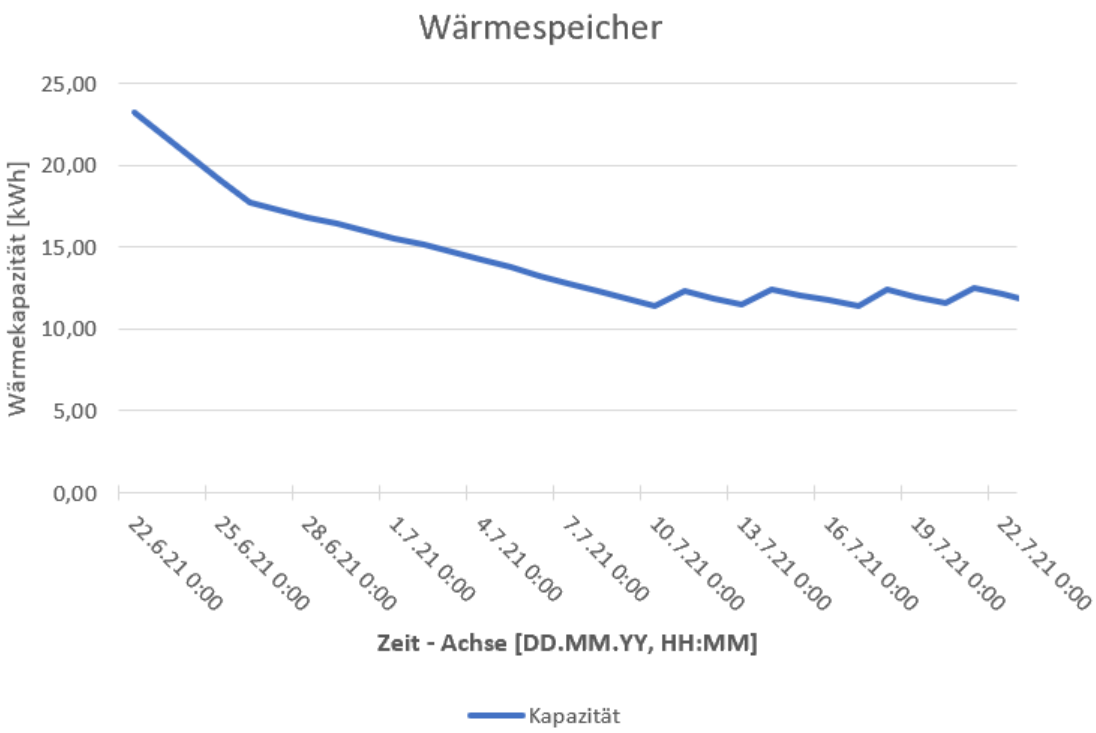


Abbildung 7.16: Liniendiagramm des Wärmespeichers mit 100% Anfangskapazität

Schlussfolgerungen und Fazit

Die Einarbeitung der Open Source Plattform OpenEMS ist sehr aufwendig, da das Anwendungssystem noch neu ist und es noch Fehlern in manchen Programmen vorhanden sind. Zudem wird das Anwendungssystem des Öfteren geupdatet, wodurch Fehler behoben werden oder Abschnitte bzw. Aufrufe verkürzt oder abgeändert werden. Die OpenEMS ist sehr komplex und besitzt neben dem kleinen Simulation Bereich auch noch viele Hardware gebundene Projekte. Die Controller, die für die Simulationen erstellt wurden, können auch bei Projekten mit Hardwarekomponenten eingesetzt werden. Nach den Simulationsresultaten lässt sich herausschließen, dass bei Anschluss eines PVT-Moduls an Stelle der Ausgangstemperaturen energetisch und ökonomisch sinnvoll wäre. Die aufgenommen Wärme durch die Außentemperatur wird durch den Controller energieeffizient eingesetzt, indem der Controller die Wärmepumpe zur Aktivierung zwingt, sobald die Temperaturen und der COP hoch sind.

Ausblick und zukünftige Umsetzung

Bei dem Open Source Plattform OpenEMS wurde das Anwendungssystem in zwei Kategorien aufgeteilt, die Edge mit der in der Arbeit überwiegend gearbeitet wurde und die als Frontend-Development gekennzeichnet ist. Wobei die Backend für die Backend- Development verantwortlich ist und nicht näher in dieser Bachelorarbeit eingegangen wurde. Die grafische Darstellung im UI der entwickelten wäre ins besonders für ein besseres Verständnis des Systemverhaltens von Vorteil. Zudem wäre es möglich den Heizstab als eigene Klasse zu erstellen, um den Zustand gewillt AN & AUS zu schalten, wie es bei der Wärmepumpe der Fall ist. In unserem Fall ist der Heizstab in den Controller HeatMode integriert. Eine weitere Optimierungsmöglichkeit ist die Anbindung der Netzleistung des Grid Meters mit einem weiteren Wärmezähler, der die Wärmeleistungen an das Netz wiedergibt.

8 Literaturverzeichnis

[1] Brosig, C. (kein Datum). Betriebsalgorithmus.

[2] Greenhouse Media GmbH. (10. 03 2020). Pufferspeicher-Technik, Auslegung und Kosten. Deutschland.

[3] Greenhouse Media GmbH. (18. 05 2020). Technik, Auslegung und Betrieb von Wärmepumpen. Deutschland.

[4] Oliver Ruhnau, L. H. (01. Oktober 2019). Time series of heat demand and heat pump efficiency for energy system modeling. Von https://www.nature.com/articles/s41597-019-0199-y abgerufen

[5] Palensky, P. (03. August 2011). Demand SIde Management: Demand Response, Intelligent Energy Systems, and Smart Loads.

[6] Fabien Mink (06.März.2020) Evaluation und Optimierungsansätze für den Eigenverbrauch aus einer häuslichen PV-Anlage

[7] Mark Johannes Windeknecht, Entwicklung einer prädiktiven Regelung für einen Prosumer mit Brennstroffzelle und Wärmepumpe

[8] David Sauter; (03.2020). L-Sol: Effizientes Heizsystem mit PVT und Wärmepumpen

[9] Grupp, Linus (21.10.2015). Entwicklung und Aufbau der Steuerung und Regelung einer Experimentalanlage zur Erforschung des kombinierten Betriebs einer Photovoltaikanlage mit Heizstab und Sole-Waser-Wärmepumpe zur Trinkwasmwasserversorgung

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorgelegte Abschlussarbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Köln, den 25.06.2021

Unterschrift

(Enes, Sarac)