Masterthesis

Ina Herrmann

Smart Heat Grid Hamburg: Konzeption und Simulation einer zentralen Steuerung zum wärmenetzdienlichen Betrieb intelligenter Wärmeübergabestationen

Fakultät Life Sciences *Faculty of Life Sciences*

Ina Herrmann

Smart Heat Grid Hamburg: Konzeption und Simulation einer zentralen Steuerung zum wärmenetzdienlichen Betrieb intelligenter Wärmeübergabestationen

Masterthesis eingereicht im Rahmen der Masterprüfung
im Studiengang Renewable Energy Systems
an der Fakultät Life Sciences
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof.-Dr. Hans Schäfers
Zweitgutachter : Philipp Eike Janßen

Abgegeben am 8. Mai 2019

Ina Herrmann

Thema der Masterthesis

Smart Heat Grid Hamburg: Konzeption und Simulation einer zentralen Steuerung zum wärmenetzdienlichen Betrieb intelligenter Wärmeübergabestationen

Stichworte

wärmenetzdienlicher Betrieb, intelligente Wärmeübergabestationen, Regelungsszenarien für flexible Wärmenetze, Modbus TCP, Speicherprogrammierbare Steuerungen

Kurzzusammenfassung

Diese Arbeit beschäftigt sich mit einer übergeordneten Steuerung für Wärmeübergabestationen. Durch intelligente Regelszenarien sollen Speicher auf der Verbraucherseite als Flexibilität des Wärmenetzes nutzbar gemacht und ein wärmenetzdienlicher Betrieb ermöglicht werden. Die Szenarien werden definiert, in einem Modell umgesetzt und abschließend bewertet. Das Modell enthält eine Erzeugerzentrale, ein Netz und mehrere Wärmeverbraucher inklusive Wärmeübergabestationen und Speichern. Der Schwerpunkt liegt neben hydraulischen Berechnungen auf der Kommunikation zwischen den Steuerungen. Hierfür wird das Protokoll Modbus TCP in die Simulationsplattform Jarvis integriert.

Ina Herrmann

Title of the paper

Smart Heat Grid Hamburg: Conception and simulation of a heat grid beneficial central control for smart substations

Keywords

heat grid beneficial operation, smart substations, control algorithms for flexible heat grids, programmable logic controller, Modbus TCP

Abstract

This thesis describes the conception and simulation of a heat grid beneficial central control for smart substations. By means of the central control, storages can be used to enhance the flexibility of heat grids. For that purpose, control algorithms are defined, implemented and evaluated in a model. The model consists of an energy center, a heat grid and consumers, including substations and storages. The thesis focuses not only on hydraulic calculations, but also on the communication of the different controllers. Therefore, the protocol Modbus TCP is implemented in the simulation platform Jarvis.

Danksagung

An dieser Stelle möchte ich mich für die Betreuung meiner Masterarbeit bei Herrn Philipp Janßen und Herrn Professor Hans Schäfers herzlich bedanken. Mein besonderer Dank geht außerdem an das gesamte SHGH Team für die dauerhafte Bereitschaft, mir bei Fragen zur Seite zu stehen und Probleme zu diskutieren.

Inhaltsverzeichnis

Tabellenverzeichnis iv

Abbildungsverzeichnis v

Quelltextverzeichnis viii

Abkürzungs- und Formelzeichenverzeichnis ix

1. Einleitung 1

1.1. Motivation 1

1.2. Aufgabenstellung und Aufbau der Arbeit 2

1.3. Einordnung der Arbeit 3

2. Grundlagen 5

2.1. Regelungstechnik 5

2.1.1. PID Regler und deren Dimensionierung 5

2.1.2. Speicherprogrammierbare Steuerungen 7

2.2. Kommunikationsnetzwerke und Protokolle 8

2.3. Wärmenetze 13

2.4. Simulationsplattform Jarvis 20

3. Pymodbus Anwendung 24

3.1. Kommunikationstest im Feld 24

3.2. Simulationsumgebung 27

4. Modellierung 34

4.1. Energiebunker als Erzeugermodell 36

4.1.1. Hydraulik und Parametrierung 37

4.1.2. Regelung 41

4.2. Vereinfachtes Last- und Netzmodell 59

4.2.1. Hydraulik und Parametrierung 59

4.2.2. Regelung 61

4.2.3. Kommunikation 66

4.3. Zusammenführung der Modelle 68

5. Konzept der W-SPS Szenarien **72**
5.1. Rahmenbedingungen 72
5.2. Regelszenarien 73

6. Umsetzung und Simulation der W-SPS Szenarien **77**
6.1. Umsetzung 77
6.2. Ergebnisse und Ergebnisbewertung 85

7. Zusammenfassung und Fazit **96**

8. Ausblick **98**

Literaturverzeichnis **100**

A. Anhang **I**

Tabellenverzeichnis

2.1. Function Codes [10] 10
2.2. Szenarien der iWÜST (TWW) [24] 19

3.1. Übersicht über die Funktionen zum Schreiben und Lesen von Daten in Jarvis 33

4.1. Beschreibung der verwendeten Komponenten 36
4.2. Parameter Bestimmung des Bunkers 40
4.3. Auslegung der Rohre im Energiebunker 41
4.4. Einstellschritte der Regelung der Energiezentrale 45
4.5. Einstellschritte nach Anpassung des Regelkonzeptes 46
4.6. Angepasste Parametrierung der Bunkerkomponeten 51
4.7. Hydraulische Berechnungen aller Komponenten 61
4.8. Ermittelte T_{max} und T_{min} für Speicher 1-3 64

5.1. Regelszenarien der W-SPS 74

6.1. Datenpunkte zur Kommunikation zwischen iWÜST, Z-SPS und W-SPS . . 78

A.1. PID-Parameter, Einschaltverzögerungen und Mindestlaufzeiten der Komponenten III

Abbildungsverzeichnis

1.1. Kategorien der benötigten Flexibilität in Wärmenetzen [13] 2

2.1. Blockstruktur des PID-Reglers [17, S.50] 6
2.2. Vorgehen bei der empirischen Dimensionierung [22, S.56] 7
2.3. Netzwerk Schema aus den vier Basis Elementen [20] 8
2.4. Grundlegender Aufbau des Servers [21] 11
2.5. Grundlegender Aufbau des Clients [21] 12
2.6. Vereinfachtes hydraulisches Modell eine Fernwärmenetzes [14] 13
2.7. Übersicht Hausanschluss [2, S.96] 14
2.8. Hausanschluss mit indirekter Wärmeübergabe [2, S.99] 15
2.9. Schema eines Gegenstrom-Wärmeübertragers [2] 16
2.10. Kommunikationsstruktur von Leitwarte zu Erzeugern und Verbrauchern . 18
2.11. *Backend*-Architektur von Jarvis [4] 20
2.12. Struktur eines Projektes in Jarvis 22
2.13. Ausschnitt CoDeSys Programmierung 23

3.1. Überblick über die Einbindung des *heartbeats* 30

4.1. Hydraulisches Schema des Gesamtmodells 35
4.2. Fließschema der Energiezentrale [3] 37
4.3. Schematische Darstellung der Regelungen im Bunker 42
4.4. Angepasste Kesselregelung 44
4.5. Einbruch der Vorlauftemperatur der Last 47
4.6. Einbruch der Vorlauftemperatur auf der Primärseite des Wärmetauschers 48
4.7. Sprunghaftes Ansteigen des Massenstroms von Pumpe 8 48
4.8. Abschaltung BHKW1 aufgrund von zu hohen Rücklauftemperaturen . . . 49
4.9. Abschaltung BHKW2 aufgrund von zu hohen Rücklauftemperaturen . . . 49
4.10. Temperaturspitze an Temperatursensor 17 51
4.11. Vorlauftemperatur Speicher an Temperatursensor 18 52
4.12. Massenstrom Pumpe 8 52
4.13. Temperatur an Temperatursensor 17 bei variablem Massenstrom der Pumpe 53

4.14. Temperatur an Temperatursensor 18 bei variablem Massenstrom der Pumpe . . . 53
4.15. Vorlauftemperatur Primärseite nach Verkleinern der Pumpe bei 5000 kW 54
4.16. Vorlauftemperatur Primärseite nach Verkleinern der Pumpe bei konstanter Last . . . 54
4.17. Vorlauftemperatur Primärseite nach Verkleinern der Pumpe bei veränderlicher Last . . . 55
4.18. Regelung eines Erdgaskessels . . . 56
4.19. Regelung BHKW1 . . . 57
4.20. Regelung BHKW3 . . . 58
4.21. R&I Schema des Last- und Netzmodells . . . 60
4.22. Temperaturregelung am Erzeugervorlauf (Temperatursensor 1) . . . 61
4.23. Vorlauftemperaturen der Lasten . . . 62
4.24. Schichttemperaturverlauf Speicher 1 . . . 62
4.25. Schichttemperaturverlauf Speicher 2 . . . 63
4.26. Schichttemperaturverlauf Speicher 3 . . . 63
4.27. Rücklauftemperaturen der Speicher . . . 64
4.28. Schichttemperaturverlauf Speicher 1 . . . 64
4.29. Schichttemperaturverlauf Speicher 2 . . . 65
4.30. Schichttemperaturverlauf Speicher 3 . . . 65
4.31. SOC Regelung des Speicher 1 . . . 65
4.32. SOC Regelung des Speicher 2 . . . 66
4.33. SOC Regelung des Speicher 3 . . . 66
4.34. Kommunikationsnetz des Verbraucher- und Lastmodells . . . 67
4.35. Verbindungsaufbau zwischen den Servern und Clients zu Beginn der Simulation . . . 68
4.36. Temperaturregelung am Vorlauf der Last . . . 69
4.37. SOC Regelung des Speicher 1 . . . 69
4.38. SOC Regelung des Speicher 2 . . . 69
4.39. SOC Regelung des Speicher 3 . . . 70
4.40. Kommunikationsnetz des gesamten Modells . . . 70

5.1. Datenaustausch zwischen iWÜST und W-SPS für die Funktionsprüfung . 75
5.2. Zeitlicher Unterschied beim Laden der Speicher 1 und 2 . . . 76

6.1. Einhalten eine Profils unter gleichmäßiger Nutzung aller Speicher . . . 82
6.2. Einhalten eines Profils unter ungleichmäßiger Nutzung der Speicher . . . 84
6.3. Ausgabe des Inbetriebnahme Szenarios . . . 85
6.4. Verlauf der Vorlauftemperatur im Szenario Netzkollaps . . . 86
6.5. Verlauf des Massenstroms der primärseitigen Pumpe im Szenario Netzkollaps . . . 87

6.6. Wechsel zwischen Normalbetrieb und Netzkollaps auf iWÜST3 87
6.7. Lastkurven der TWW an iWÜST 1-3 aus der Sicht des Netzes 87
6.8. SOC Verlauf der iWÜST während eines Netzkollapses 88
6.9. Massenströme der Pumpen 1-3 während des Netzkollapses 88
6.10. Ausgabe der iWÜST bei einer Funktionsprüfung 89
6.11. SOC Verlauf auf iWÜST1 90
6.12. Änderung der enthaltenen Energie im Speicher der iWÜST1 90
6.13. Temperaturverlauf der 5 Schichten im Modell 91
6.14. Erwarteter Temperaturverlauf [7, S.123] 91
6.15. Temperaturverlauf der 10 Schichten im Modell 92
6.16. Ausgabe nach Ergänzung der automatisierten Kapazitätsberechnung . . 92
6.17. Verlauf des SOC auf der iWÜST1 93
6.18. Verlauf der Rücklauftemperaturen während der Speicherüberladung . . . 93
6.19. SOC Verlauf der iWÜST1 94
6.20. SOC Verlauf der iWÜST2 94
6.21. Änderung der Energie im Speicher der iWÜST1 95
6.22. Änderung der Energie im Speicher der iWÜST2 95

A.1. Rohrinnendurchmesser in Abhängigkeit von Strömungsgeschwindigkeit und Volumenstrom VIII
A.2. SOC Verlauf der iWÜST1 IX
A.3. SOC Verlauf der iWÜST2 IX
A.4. SOC Verlauf der iWÜST3 IX
A.5. Änderung der Energie im Speicher der iWÜST1 X
A.6. Änderung der Energie im Speicher der iWÜST2 X
A.7. Änderung der Energie im Speicher der iWÜST3 X

Quelltextverzeichnis

3.1. Modbus TCP Server auf Raspberry Pi 24
3.2. Client zur Raspberry Pi Abfrage über Modbus TCP 25

4.1. *temperature* Methode des bestehenden wärmegeführten BHKW 38
4.2. *temperature* Methode des stromgeführten BHKW 39
4.3. *initialize* der W-SPS 67
4.4. *initialize* der iWÜST1 67
4.5. *initialize* der Z-SPS 71

6.1. Umsetzung der Inbetriebnahme im *initialize code* der iWÜST 79
6.2. Umsetzung der Inbetriebnahme im *initialize code* der W-SPS 79
6.3. Umsetzung des Netzkollapses im *main loop code* der Z-SPS 80
6.4. Umsetzung des Netzkollapses im *main loop code* der W-SPS 80
6.5. Umsetzung des Netzkollapses im *main loop code* der iWÜST 80
6.6. Umsetzung der Funktionsprüfung im *main loop code* der iWÜST 81

A.3. *YAML* Datei des bestehenden wärmegeführten BHKW IV
A.4. *YAML* Datei des stromgeführten BHKW V
A.5. Beispiel Modbus TCP Verwendung in Jarvis (*initialize_code*) VI
A.6. Beispiel Modbus TCP Verwendung in Jarvis (*main_loop_code*) VI

Abkürzungs- und Formelzeichenverzeichnis

Abkürzung	Bedeutung
BHKW	Blockheizkraftwerk
FC	function code
iWÜST	intelligente Wärmeübergabestation
LWL	Lichtwellenleiter
NEW 4.0	Norddeutsche Energiewende 4.0
R&I	Rohr- und Instrumentenfließschema
SHGH	Smart Heat Grid Hamburg
SOC	State of Charge
SPS	Speicherprogrammierbare Steuerung
TWW	Trinkwarmwasser
W-SPS	Wärmeübergabestations-Speicherprogrammierbare Steuerung
WÜST	Wärmeübergabestation
Z-SPS	Zentrale-Speicherprogrammierbare Steuerung

Symbol	Bedeutung
A	Fläche des Wärmeübertragers
c_p	Wärmekapazität von Wasser
k	Wärmedurchgangskoeffizient
K_p	Verstärkungsfaktor des proportionalen Anteils eines Reglers
$\dot{m}$	Massenstrom
$\dot{m}_{max}$	maximaler Massenstrom
m	Masse eines Speichers
$\dot{Q}$	Wärmestrom
$\dot{Q}_{max}$	maximaler Wärmestrom

Q_i	Energie pro Speicher i
Q_{max_900s}	maximale Energie pro Viertelstunde
Q_{max_ist}	maximale Kapazität des Speichers zum aktuellen Zeitpunkt
Q_{max}	maximale Kapazität des Speichers
Q_{total}	gesamte Energie für eine Viertelstunde
SOC_{diff_total}	notwendige Änderung des SOC in einer Viertelstunde
SOC_{diff}	notwendige Änderung des SOC pro Sekunde
SOC_{ist}	SOC Istwert
SOC_{soll}	SOC Sollwert
T_n	Nachstellzeit eines Reglers
T_v	Vorhaltezeit eines Reglers
$\Delta\vartheta$	Temperaturdifferenz zwischen Vor- und Rücklauf
$\Delta\vartheta_m$	mittlere Temperaturdifferenz am Wärmeübertrager
$\Delta\vartheta_{max}$	maximale Temperaturdifferenz am Wärmeübertrager
$\Delta\vartheta_{min}$	minimale Temperaturdifferenz am Wärmeübertrager

1. Einleitung

Dieses Kapitel gibt eine Einleitung in die Thesis und deren Motivation. Es ordnet sie in den Kontext der Forschungsprojekte an der HAW Hamburg ein und erläutert die Aufgabenstellung und den Aufbau der Arbeit.

1.1. Motivation

Die Bundesregierung verabschiedete 2016 den Klimaschutzplan 2050, um den im Pariser Abkommen geforderten Klimaschutzstrategien zu folgen. Als Langfristziel wird darin festgelegt, dass Deutschland bis zum Jahr 2050 weitestgehend treibhausgasneutral werden soll. Der Plan umfasst verschiedene Sektoren und bezieht sich im "Fahrplan für einen nahezu klimaneutralen Gebäudebestand" auch auf Heizsysteme. Um einen klimaneutralen Gebäudebestand zu sichern, soll eine schrittweise Abkehr von fossilen Heizungssystemen geschehen. [6]
Auf Grund der mittleren Jahrestemperaturen und der daraus resultierenden Anzahl an Heiztagen im Jahr, spielt die Beheizung von Gebäuden in Deutschland eine wichtige Rolle. Zusätzlich ist die Bereitstellung von Warmwasser eine Notwendigkeit für Wohnkomfort. Da über 50 % des Endenergieverbrauchs in Deutschland auf Wärmeanwendungen entfällt, geht damit auch eine große Verantwortung einher, diese mit geringer Treibhausgas-Emission zu gewinnen. [2, 5]
Aktuelle Studien zeigen, dass Fernwärme in einem zukünftigen nachhaltigen Energiesystem eine wichtige Rolle spielen kann. Allerdings zeigen diese Studien auch, dass die heutigen Fernwärmenetze radikalen Veränderungen unterliegen müssen. Sie müssen dazu beispielsweise in Niedrigtemperatur-Netze umgewandelt werden, die geringe Wärmeverluste haben. Grundsätzlich sollten sie in der Lage sein, erneuerbare Wärmequellen zu integrieren und in einem intelligenten Energiesystem mit den Strom- und Gasnetzen gekoppelt werden. Um diese Ziele erreichen und flexibel auf die fluktuierende Erzeugung durch regenerative Wärmeerzeuger reagieren zu können ist eine Optimierung der Steuerungs- und Regelungssysteme des Fernwärmenetzes notwendig. [16]
Abbildung 1.1 stellt die Kategorien dieser notwendigen Flexibilität dar. Es sollte zwischen kurz-, mittelfristiger und saisonaler Flexibilität unterschieden werden.

Abbildung 1.1.: Kategorien der benötigten Flexibilität in Wärmenetzen [13]

Hierfür werden unterschiedliche Wasserspeicher, eine Sektorenkopplung und anderweitige Speichermedien benötigt. Die Wasserspeicher reichen von beispielsweise Aquifer Speichern über Pufferspeicher in Energiezentralen bis hin zur Nutzung der Infrastruktur und Wärmeübergabestationen samt Speicher für das Trinkwarmwasser. Bisher wurden Gebäude als passive Elemente in der Infrastruktur der Fernwärmenetze gesehen. Aus der Abbildung wird deutlich, dass die Gebäude für kurzfristige Flexibilität eingesetzt werden können und sollten. Ein Element der Nutzbarmachung der Gebäude können extern steuerbare Wärmeübergabestationen sein, die die hausinternen Speicher für kurzfristige Flexibilität verfügbar machen.

1.2. Aufgabenstellung und Aufbau der Arbeit

Diese Arbeit beschäftigt sich mit einer übergeordneten Steuerung für Wärmeübergabestationen, die extern gesteuert werden können und intelligente Regelszenarien befolgen sollen. Ziel ist es, Regelszenarien dieser Steuerung zu konzeptionieren und simulieren. Dadurch sollen die Speicher auf der Verbraucherseite als Flexibilität des Wärmenetzes nutzbar gemacht und ein wärmenetzdienlicher Betrieb der Wärmeübergabestationen

ermöglicht werden. Zunächst werden die notwendigen Szenarien in der Arbeit definiert und konkretisiert. Es wird im Anschluss ein Modell entwickelt, das eine Erzeugerzentrale, ein Netz und mehrere Wärmeverbraucher inklusive Wärmeübergabestationen und Speicher enthält. Der Schwerpunkt des Modells soll neben hydraulischen Berechnungen auf der Kommunikation zwischen den Steuerungen liegen. Dafür muss das Protokoll Modbus TCP als Teil dieser Arbeit in die Simulationsplattform Jarvis integriert werden, um eine möglichst realitätsnahe Kommunikation umzusetzen. Die Szenarien sollen schlussendlich simuliert und bewertet werden.
Die Arbeit beginnt mit einer Beschreibung der Grundlagen. Danach folgt ein Kapitel über die Verwendung von Modbus TCP in Jarvis und in einem Kommunikationstest im Feld. Das Kapitel Modellierung beschäftigt sich mit dem Modellaufbau der Subsysteme und anschließend mit der Zusammenführung dieser. Im darauffolgenden Kapitel wird das Regelungskonzept aufgezeigt und die Rahmenbedingungen für die Konzeptentwicklung beschrieben. Daraufhin werden die entwickelten Szenarien simuliert und die Ergebnisse bewertet. Zum Schluss folgt ein Fazit und ein Ausblick, sowie eine Zusammenfassung.

1.3. Einordnung der Arbeit

Die Masterarbeit entsteht im Rahmen des Forschungsprojektes “Smart Heat Grid Hamburg“ (SHGH) . Außerdem besteht ein Bezug zu dem Forschungsprojekt "Norddeutsche Energiewende 4.0" (NEW 4.0) .

Smart Heat Grid Hamburg

Das Forschungsprojekt SHGH hat sich zum Ziel genommen, innerhalb von vier Jahren intelligente Konzepte für Wärmenetze zu entwickeln und diese in Feldtests im Fernwärmenetz in Hamburg-Wilhelmsburg zu testen. Das Projekt ist eine Kooperation aus der HAW Hamburg, Hamburg Energie und der eNeG Gesellschaft für wirtschaftlichen Energieeinsatz. Es ist ein Nachfolgeprojekt von “Smart Power Hamburg“ und möchte erforschen, wie der Anteil Erneuerbarer Energien an der Wärmeerzeugeung erhöht und die Effizienz gesteigert werden kann. Im Rahmen des Forschungsprojektes werden unter anderem Wärmeübergabestationen erforscht und die Entwicklung von intelligenten Regelungen dieser vorangetrieben. [9]
Zu der Erprobung einer angepassten SPS-Programmiermethodik für intelligente Wärmeübergabestationen wurde bereits eine Masterarbeit durchgeführt[24]. Diese Masterthesis setzt bei der übergeordneten Steuerung der intelligenten Wärmeübergabestationen an.

NEW4.0 - KulturEnergieBunkerAltonaProjekt

NEW 4.0 ist ein großes Verbundprojekt, das eine umweltfreundliche und gesellschaftlich akzeptierte regenerative Stromversorgung von Hamburg und Schleswig-Holstein als übergeordnetes Ziel hat. Es soll bis 2035 umgesetzt werden und auf 100 % Erneuerbaren Energien basieren.[19]
Ein Teilprojekt ist der Energiebunker in Altona. In dem Hochbunker wird von dem Verein "KulturEnergieBunkerAltonaProjekt" eine Kultur- und Energiezentrale geplant. Der Energieteil soll einen wesentlichen Beitrag zur dezentralen Versorgung leisten und durch eine Wärmeeinspeisung in das Fernwärmegroßnetz in Hamburg außerdem eine Öffnung dieses Netzes für Dritte voran treiben. Der Energieteil soll Hand in Hand gehen mit einem Kulturteil, der allen Menschen, insbesondere den Stadtteilbewohner*innen [1], offen steht und durch eine Quersubventionierung von Seiten des Energieteils neue Möglichkeiten für ein Kulturzentrum bietet. [1]
In einem von der HAW Hamburg bearbeiteten Teilprojekt des NEW4.0 Projektes, wird eine Regelung für die geplante Energiezentrale entwickelt. In dieser Arbeit wird, basierend auf einer technischen Machbarkeitsstudie für die geplante Energiezentrale und den Arbeiten aus dem Teilprojekt, ein Erzeugermodell umgesetzt und in das Gesamtmodell eingebunden. [1, 19]

[1] In dieser Masterarbeit sollen alle Geschlechter gleichermaßen angesprochen werden, weswegen eine "gendergerechte Sprache" gewählt wurde. Sollte es sich beispielsweise bei Kommunikationsteilnehmern eindeutig um Gegenstände handeln, dann wird die männliche Form beibehalten.

2. Grundlagen

Das erste Kapitel beschreibt die notwendigen technischen Grundlagen zur Erstellung der vorliegenden Arbeit. Nach einem Einblick in die Regelungstechnik und Speicherprogrammierbare Steuerungen folgen die kommunikationstechnischen Grundlagen von Datennetzwerken und das verwendete Feldbusprotokoll Modbus TCP wird beschrieben. Des Weiteren erfolgt eine Einführung in das Thema Wärmnetze, insbesondere in die Wärmeübergabestation. Das Kapitel schließt mit einer Beschreibung der verwendeten Simulationsplattform.

2.1. Regelungstechnik

In der Regelungstechnik sollen Prozesse von außen beeinflusst werden, sodass sie in der gewünschten Weise ablaufen. Die beeinflussbare Größe muss so gewählt werden, dass das Ziel erreicht werden kann. Da der momentane Grad der Zielerreichung einen Einfluss auf diese Auswahl hat, entsteht ein Regelkreis. Der Regelkreis setzt sich zusammen aus einem dynamischen System (*Regelstrecke*), einer beeinflussbaren Größe (*Stellgröße*) und einer messbaren Größe (*Regelgröße*). Die *Führungsgröße* gibt vor, wie die messbare Größe zeitlich verändert werden soll, während gleichzeitig die Einflüsse einer *Störgröße* minimiert werden sollen. Die Aufgabe der Regelungstechnik besteht also darin, die Stellgröße so vorzugeben, dass das Regelungsziel erfüllt wird.[17]

2.1.1. PID Regler und deren Dimensionierung

Um das Regelungsziel zu erreichen, können unterschiedliche Reglertypen, wie beispielsweise P-, PI oder PID-Regler, genutzt werden. Der PID-Regler findet in vielen Anwendungsfällen Einsatz.

Er enthält einen proportionalen, integrierenden und einen differenzierenden Anteil, den P-, I-, und D-Anteil. Somit gibt es drei Parameter, die einzustellen sind.

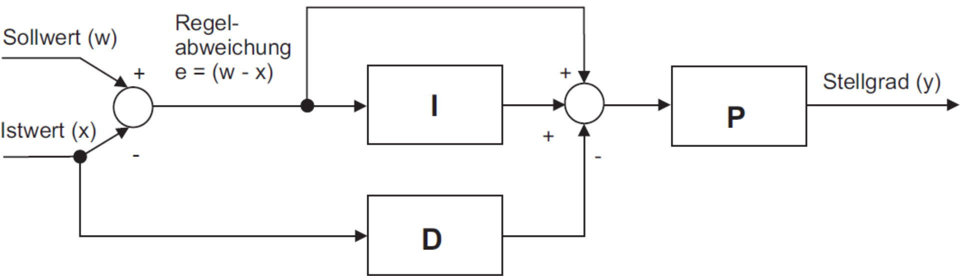

Abbildung 2.1.: Blockstruktur des PID-Reglers [17, S.50]

Der K_p-Wert ist proportional zum P-Anteil. Wird er verringert, dann wird auch die Verstärkung verringert, was zu einem stabileren, aber auch trägeren Verhalten führt. T_n steht für die Nachstellzeit und verhält sich gegenläufig zum I-Anteil. Wird T_n vergrößert, integriert der I-Anteil langsamer auf und dadurch entsteht ebenfalls ein stabileres, aber trägeres Verhalten. Die Vorhaltezeit T_v steht für den D-Anteil und ist proportional zu diesem. Wird er vergrößert, dann wirkt dieser der Änderung des Istwertes stärker entgegen. Abbildung 2.1 zeigt die Blockstruktur eines PID-Reglers. Daraus geht hervor, dass die Änderung des P-Anteils auch das Verhalten des I- und D-Anteils beeinflusst. Dies muss bei der Parametrierung des PID-Reglers bedacht werden. [22]

Empirische Dimensionierung

Neben heuristischen Methoden, wie dem Faustformelverfahren, zur Einstellung von Reglern, gibt es auch empirische Methoden zur Ermittlungen der Regelparameter. Die empirische Dimensionierung wird häufig in der Praxis angewendet, da sie komplett ohne die Berechnungen auskommt. Abbildung 2.2 zeigt das Vorgehen dieser Methode.

Zunächst wird der Proportionalbereich (P) relativ groß eingestellt und immer weiter reduziert. Wenn der Istwert nach zwei bis drei Schwingungen einen stabilen Endwert erreicht hat, dann wird als nächstes der D-Anteil dazu genommen. Dieser wird immer weiter vergrößert, bis ein Endwert mit möglichst kleiner Schwingung erreicht ist. Ziel ist es, in den ersten beiden Schritten eine möglichst kleine Schwingung und somit einen stabilen Wert zu erreichen. Eine Regelabweichung wird dabei zunächst in Kauf genommen. Bleibt nach dem zweiten Schritt eine Regelabweichung, dann kann diese im letzten Schritt mit Hilfe des I-Anteils ausgeglichen werden. Zuletzt wird die Nachstellzeit dazu genommen, die in der Regel mit einem vierfachen Wert der Vorhaltezeit in einem günstigen Bereich liegt.[22]

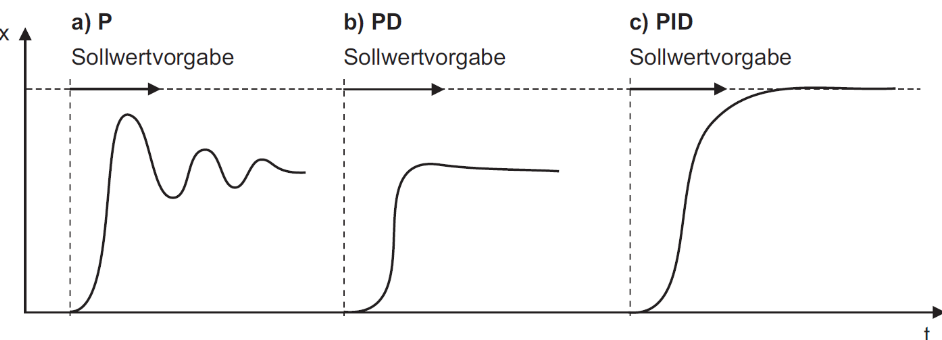

Abbildung 2.2.: Vorgehen bei der empirischen Dimensionierung [22, S.56]

2.1.2. Speicherprogrammierbare Steuerungen

Bei Speicherprogrammierbaren Steuerungen (SPS) handelt es sich um programmierbare Steuergeräte, die modular aufgebaut sind. In dieser Arbeit wird auf SPS der Firma WAGO Kontakttechnik Bezug genommen, die mittels der Software CoDeSys 2.3 programmiert werden. Sie können je nach Anwendungsfall anhand von Einzelmodulen frei konfiguriert werden. Sie bestehen aus einer CPU und einer beliebigen Anzahl an Eingangs- und Ausgangsbaugruppen. Dabei kann es sich um digitale und analoge Ein- und Ausgänge handeln oder auch um Schnittstellenerweiterungen und Kommunikationsbaugruppen. Diese Klemmen werden von einer Endklemme abgeschlossen, die die offenen Kontakte der letzten Klemme bedeckt. Durch die Modularität gibt es keine ungenutzten Baugruppen und das System kann leicht erweitert werden. [12]

CoDeSys

Die Software CoDeSys basiert auf Aufrufen von voneinander unabhängigen Bausteinen und Programmen. Die Programme können entweder Bausteine aus bestehenden Bibliotheken enthalten oder werden eigenständig programmiert. Sie können in verschiedenen Programmiersprachen umgesetzt werden, die in der Norm IEC 661131-3 für SPS Sprachen standardisiert wurden. Die Adressierung von Variablen richtet sich ebenfalls nach der Norm. So kann eine Variable in CoDeSys einem Register zugewiesen werden und dieses über Modbus TCP von anderen Geräten im Netzwerk gelesen oder geschrieben werden.
In CoDeSys wird die Kommunikation über Modbus TCP über die Steuerungskonfigurationen erstellt. Dort können im *Modbus Master* Slaves mit IP-Adresse und Port festgelegt

und diesen Register zugewiesen werden. Der eigentliche Kommunikationsaufbau erfolgt nicht sichtbar für den User im Hintergrund des Programms.[10]

2.2. Kommunikationsnetzwerke und Protokolle

Ein Netzwerk ist grundsätzlich eine Gruppe miteinander verbundener Systeme, die untereinander kommunizieren können. Es kann mittels vier Basiselementen beschrieben werden:

- Rechnern oder Knoten (z.B. SPS von Erzeugern oder Verbrauchern)
- Infrastrukturkomponenten (z.B. ein Switch)
- Verkabelung
- Protokolle (z.B. Modbus TCP)

Ein Beispiel für eine Verknüpfung dieser vier Elemente zeigt Abbildung 2.3.

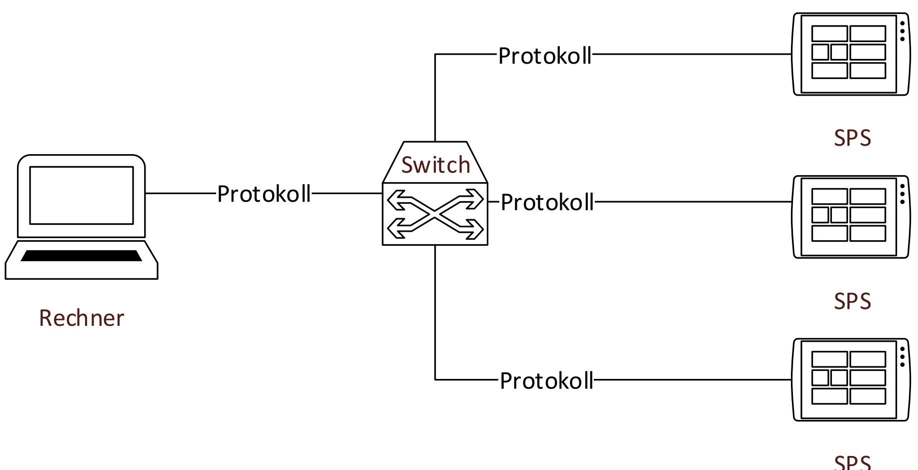

Abbildung 2.3.: Netzwerk Schema aus den vier Basis Elementen [20]

Die Rechner oder Knoten sollen verbunden werden, während die Infrastrukturkomponenten den Anschluss und die Kopplung der Rechner leisten. Sie übertragen Datensignale und erlauben Datenströme anhand von Sicherheitsrichtlinien oder aber unterbinden diese. Die Verkabelung stellt die physikalische Verbindung zwischen den einzelnen

Elementen her, wobei auch drahtlose Alternativen existieren. Ein Protokoll legt die Regeln für den Datenaustausch fest. [20]
Es handelt sich um ein großes Themengebiet, das hier auf Ethernet basierte Netzwerke und Modbus beschränkt wird, da diese für das Verständnis der Arbeit relevant sind. Ethernet ist eine Datennetztechnologie, die auf Datenrahmen (*Frames*) basiert. Durch sie können Geräte mit jedem anderen Gerät im Netz Daten in Form von *Frames* oder Paketen austauschen. Es kann die Basis für verschiedene Protokolle wie TCP/IP (Transmission Control Protocol/ Internet Protocol) bilden und ist in einer IEEE-Norm standardisiert. [18]
Eine oft verwendete Technik der Vernetzung ist das Client-Server-Prinzip. Es existiert eine Aufgabenteilung, in der von einem oder mehreren Rechnern als Servern zentral Dienstleistung oder Ressourcen bereit gestellt werden. Die übrigen Geräte des Netzwerkes können als Clients darauf zugreifen. Prozesse zwischen Server und Client könne sowohl synchron als auch asynchron verlaufen. Bei synchronen Anfragen wartet der Client auf eine Antwort des Servers, bevor der Prozess weiter arbeitet. Bei asynchronen Anfragen dagegen läuft der Prozess bereits weiter und überprüft regelmäßig, ob eine Antwort vom Server erhalten wurde. [11]

Modbus

Modbus hat sich in der Automatisierung als de facto Standardprotokoll für die Übertragung von Maschinendaten entwickelt und dient als universelle Kommunikationsschnittstelle. Es wird klassischerweise unterschieden zwischen Modbus-TCP, Modbus-ASCII und Modbus-RTU. [10]
Modbus TCP verwendet TCP/IP und Ethernet zur Übertragung von Daten in Form des Modbus Protokolls. Das heißt Modbus TCP vereint ein physikalisches Netz (Ethernet) mit einem Netzwerkstandard (TCP/IP) und einer Standardmethode zur Darstellung von Daten (Modbus). Es handelt sich um eine Client-Server-Schnittstelle. Ein Modbus-Server stellt Daten zur Verfügung, welche dann vom Client abgerufen werden. Dabei kann ein Kommunikationsgerät Server und Client in sich vereinen und sowohl Daten zur Verfügung stellen als auch Daten bei anderen Geräten abrufen. Für die Adressierung zwischen Kommunikationspartnern wird die IP-Adresse verwendet und die Kommunikation erfolgt über Port 502. Neben der IP Adresse und dem Port können noch *unit identifier (unit ids)* festgelegt werden. Diese sind allerdings nur relevant, wenn es um Brücken zu seriellen Netzwerken geht. Ansonsten ist die *unit id* auf einen Standardwert, meist 0, festgesetzt.
Modbus-RTU (Remote Terminal Unit) verwendet einen ähnlichen Aufbau, allerdings findet die Übertragung über die serielle Schnittstelle RS-232 oder RS-485 statt. [8]
Die verschiedenen Modbus Varianten haben ein einheitliches Anwendungsprotkoll und

sind durch Grunddatentypen vereint. Es handelt sich im wesentlichen um vier Datentypen.

- Discrete Inputs (Digitale Eingänge)
- Coils (Digitale Ausgänge)
- Input Register (Analoge Eingangsdaten)
- Holding Register (Analoge Ausgangsdaten)

Sie können grundsätzlich in zwei Kategorien eingeteilt werden. Die digitalen Dienste, die sich auf digitale Daten beziehen und die Register Dienste, welche sich mit analogen Ein- und Ausgangsdaten beschäftigen. Ihr Funktionsweisen werden in sogenannten *function codes* (FC) zusammengefasst. Tabelle 2.1 stellt die wichtigsten FC dar und beschreibt diese.[10]

Tabelle 2.1.: Function Codes [10]

FC	**Name**	**Beschreibung**
1	read coils	Lesen mehrerer Eingangs- und Ausgangsbits
2	read inputs discretes	Lesen mehrerer Eingangsbits
3	read multiple registers	Lesen mehrerer Eingangsregister
5	write coil	Schreiben eines Ausgangsbits
6	write single register	Schreiben eines Ausgangsregisters
16	write multiple registers	Schreibt mehrere Ausgangsregister

Pymodbus

Pymodbus ist eine Modbus Protokoll Implementation für die Programmiersprache Python und läuft ab Python Version 2.7 und Version 3 aufwärts. Es funktioniert ebenfalls auf Basis des Client-Server-Modells. Pymodbus kann sowohl für synchrone als auch asynchrone Clients und Servers genutzt werden. Dabei decken beide Versionen die Funktionalität der FCs ab. Sowohl die Benutzung von Modbus TCP als auch eine serielle Verbindung über Modbus RTU ist möglich.
Innerhalb dieser Arbeit soll ein synchroner Server die Register einer realen SPS ersetzen. Auf dem Server wird die Anzahl der Register festgelegt und diese initial beschrieben. Der Server ist der passive Part der Kommunikation und wartet auf eine Anfrage des Clients. Der Aufbau des Servers in Pymodbus folgt immer dem gleichen grundlegenden Prinzip, das in Abbildung 2.4 dargestellt ist. [21]

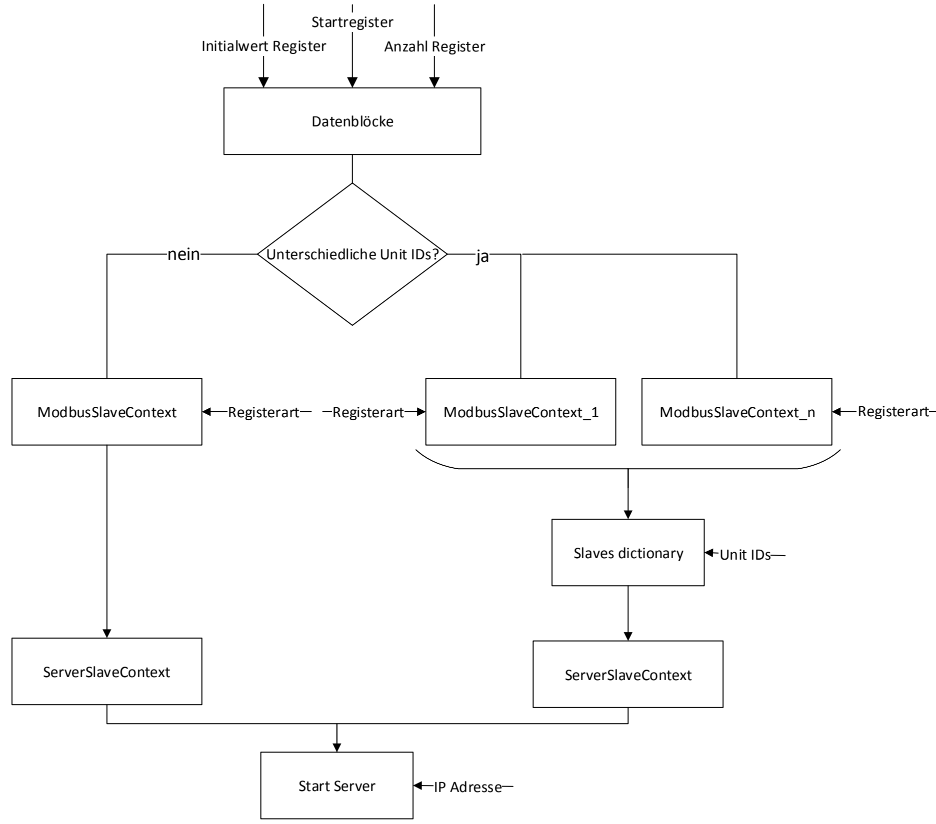

Abbildung 2.4.: Grundlegender Aufbau des Servers [21]

Zunächst werden Datenblöcke bestimmt, die eine Vorgabe für die Anzahl der Register und die Initialwerte dieser bekommen. Sollen verschiedene Clients unterschiedliche *unit ids* erhalten, dann wird mit mehreren sogenannten *ModbusSlaveContext* gearbeitet, in denen die Datenblöcke gemeinsam mit ihren Registerarten abgespeichert werden. Die *ModbusSlaveContexte* werden mit ihren jeweiligen *unit ids* in einem Dictionary zusammengefasst und in einem *ServerSlaveContext* gespeichert. Dieser wird beim Start des Servers zusammen mit dessen IP Adresse eingelesen.
Sollen keine unterschiedlichen *unit ids* verwendet werden, dann wird der *ModbusSlaveContext* direkt als *ServerSlaveContext* gespeichert. Der Start des Servers erfolgt wie bereits beschrieben.
Nach dem der Server die Anzahl der Register festgelegt und diese initialisiert hat, kann ein Client Register abfragen oder schreiben. Er bildet den aktiven Part der Kommunikation.
Synchrone Clients sollen in dieser Arbeit zum einen auf virtuellen SPS (Soft-SPS) in

der Simulation laufen und die Kommunikation mit Servern aufbauen, die ebenfalls auf Soft-SPS laufen. Zum anderen sollen die Clients dazu dienen, die Kommunikation mit Hardware SPS zu ermöglichen. Dies kann aus der Simulation heraus geschehen oder aber in einem extra Programm, um beispielsweise einen Kommunikationstest durchzuführen. In allen Fällen ist der grundsätzliche Aufbau ähnlich. Er wird in Abbildung 2.5 dargestellt.

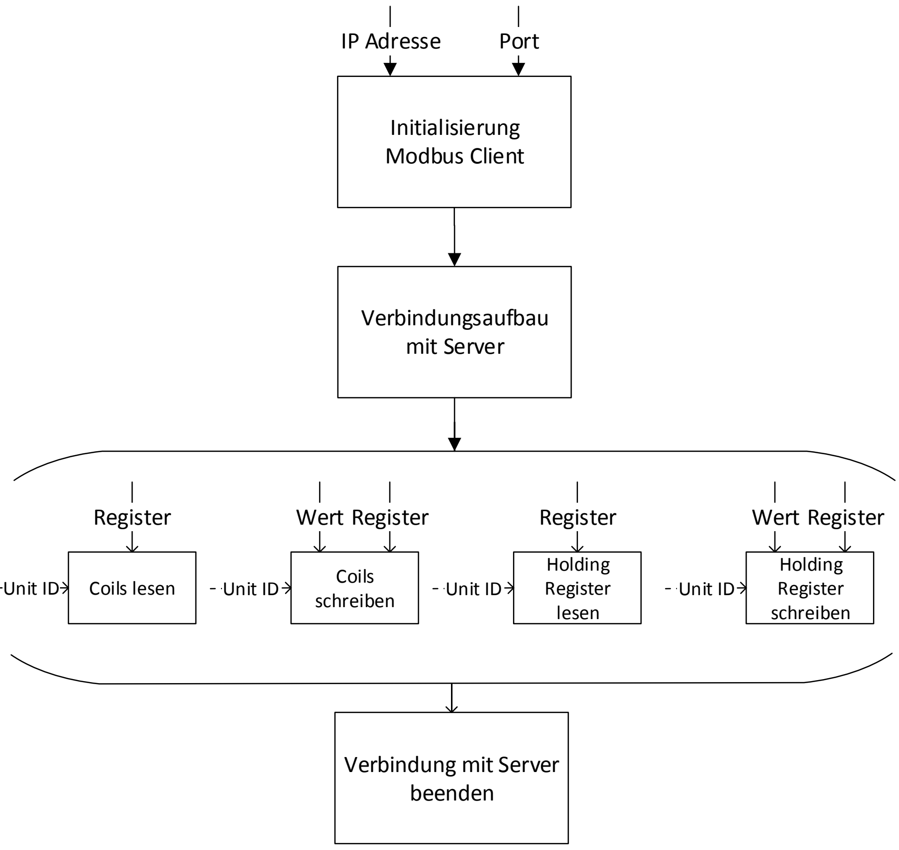

Abbildung 2.5.: Grundlegender Aufbau des Clients [21]

Zunächst muss der Modbus Client im ersten Schritt initialisiert werden und ihm die IP Adresse des Servers und der Port übergeben werden. Im nächsten Schritt wird die Verbindung zu dem Server aufgebaut. Es können nun Register gelesen und geschrieben werden. In der Regel werden nur *Coils* und *Holding Register* als Registerarten benötigt. Dabei muss die Registerart, die gelesen oder geschrieben werden soll, der vorgegeben Registerart auf dem Server entsprechen. Sie muss vorher im *ModbusSlaveContext*

entsprechend angelegt worden sein. Den Funktionen zum Lesen muss nur das Register übergeben werden, die schreibenden Funktionen benötigen zusätzlich noch den zu schreibenden Wert. Die *unit id* ist optional. Wenn sie nicht gesetzt wird, wird der default Wert 0 verwendet. Zum Schluss wird die Verbindung mit dem Server wieder beendet.

2.3. Wärmenetze

Als Wärmenetz werden Transport- und Verteilnetze zur Wärmeversorgung von externen Verbrauchern beschrieben, die sich in der Regel auf öffentlichem Boden befinden. Dabei wird häufig differenziert zwischen Nah- und Fernwärmenetz, wobei in Deutschland Übertragungen zwischen 50 kW und wenigen Megawatt als Nahwärme bezeichnet werden. Der Unterschied ist vor allem durch die Projektplanung und -umsetzung geprägt. In dieser Arbeit wird zusammenfassend der Begriff Fernwärme verwendet, da der Übergang zwischen beiden fließend ist und sich der Aufbau nicht grundlegend unterscheidet. [2]

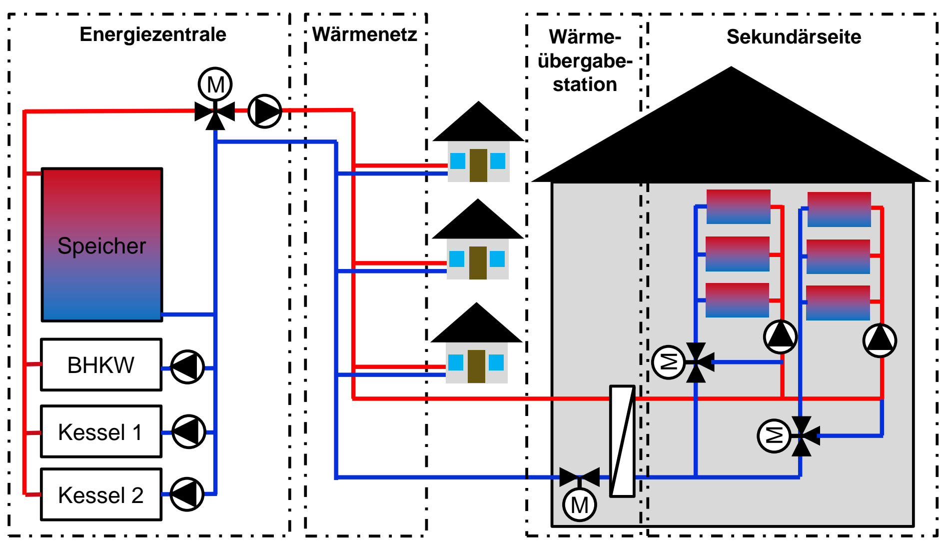

Abbildung 2.6.: Vereinfachtes hydraulisches Modell eine Fernwärmenetzes [14]

Abbildung 2.6 stellt den grundsätzlichen Aufbau eines Fernwärmenetzes dar. Ein oder mehrere Erzeuger erwärmen das Wärmeträgermedium, wobei es sich in der Regel um flüssiges Wasser handelt. Das erhitze Wasser wird über Rohrleitungen im sogenannten

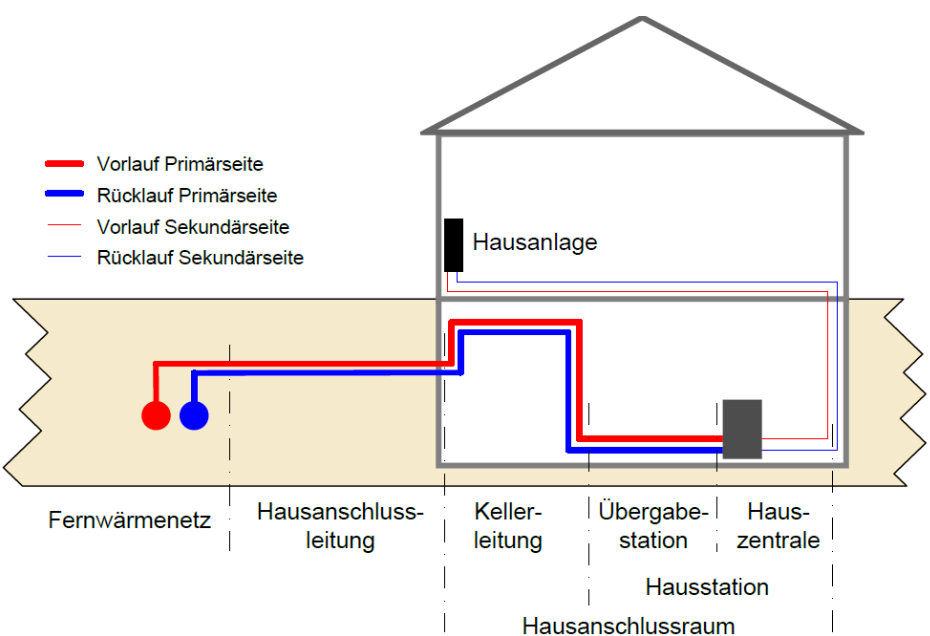

Abbildung 2.7.: Übersicht Hausanschluss [2, S.96]

Vorlauf in Richtung der Verbraucher*innen gepumpt, die über Heizkreise mit Wärme versorgt werden. Pumpstationen sorgen für einen Druckausgleich innerhalb des Netzes. An einer Wärmeübergabestation wird die Wärme übertragen. Das abgekühlt Wasser des Fernwärmenetzes wird über den Rücklauf zurück zu der Erzeugeranlage gepumpt, um dort erneut erhitzt zu werden.
Im folgenden wird auf die Netzanbindung der Wärmeabnehmer*innen eingegangen und das Vorgehen bei dem Auslegen von Wärmeübertragern beschrieben. Außerdem wird die Pumpenauslegung und Dimensionierung von Rohren beschrieben. Danach folgt eine Erläuterung der sogenannten intelligenten Wärmeübergabestation und den Unterschieden zu einer regulären Übergabestation.

Netzanbindung von Wärmeabnehmer*innen

Abbildung 2.7 zeigt die Anbindung einer Hausanlage an ein Fernwärmenetz. Die Hausanschlussleitung verbindet das Fernwärmenetz mit der Übergabestation. Als Wärmeübergabestation ist die Verbindung zwischen Hausanschlussleitung und Hauszentrale definiert. Sie stellt klassischerweise die Eigentumsgrenze zwischen Wärmelieferant und Wärmeabnehmer*innen dar, sodass die Übergabestation in der Regel Eigentum der Abnehmer*innenseite ist. Die Anbindung an ein Fernwärmenetz kann direkt oder indirekt

erfolgen. Bei direktem Anschluss fließt das Wärmeträgermedium des Erzeugers auch durch die Hausanlage. Das Wasser kommt somit in direkten Kontakt mit den Hausanlagen und kann dort verunreinigt werden. Der Wärmelieferant muss die Wasserqualität überwachen und einstellen, weswegen ein direkter Anschluss oftmals vermieden werden soll, um eine gute Wasserqualität einfacher gewährleisten zu können. Außerdem ist ein direkter Anschluss nicht möglich, wenn Druck und Temperatur des Fernwärmenetzes nicht für die Hausanlage geeignet sind. [2]
Bei indirekter Anbindung trennt deswegen ein Wärmeübertrager das Verbrauchernetz und das Netz des Versorgers hydraulisch. Die zwei Heizkreise werden als Primär- und Sekundärseite bezeichnet. Bei der Primärseite handelt es sich um die Netzseite, während die Sekundärseite hausseitig durchströmt wird. Trinkwarmwasser wird aus hygienischen Gründen indirekt angebunden, wobei es über das Fernheizwasser oder das Heizwasser der Verbraucheranlage erwärmt werden kann. In der Regel wird es primärseitig angebunden, da es sonst zu erhöhten Energieverlusten kommt. Trinkwarmwasser soll der Verbraucher*in bestenfalls jederzeit zur Verfügung stehen, weswegen oftmals Trinkwarmwasserspeicher verwendet werden. [2]

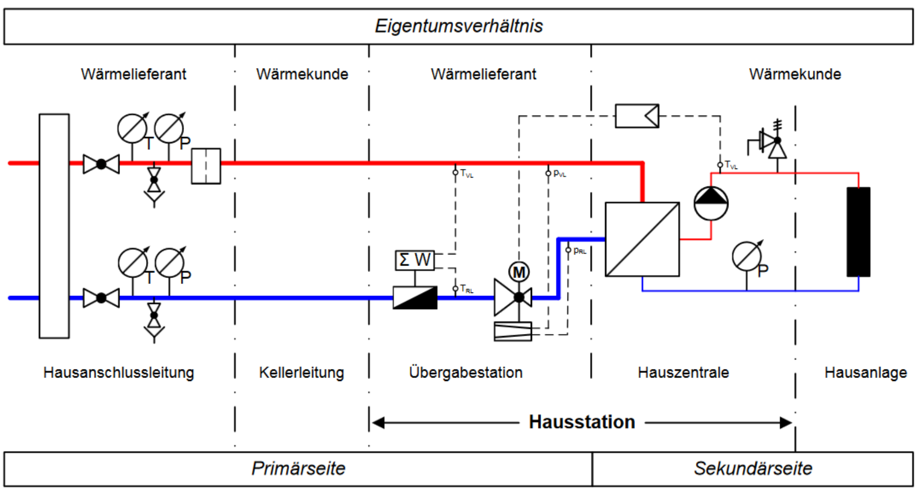

Abbildung 2.8.: Hausanschluss mit indirekter Wärmeübergabe [2, S.99]

In dieser Arbeit werden nur indirekte Anschlüsse betrachtet und davon ausgegangen, dass ein Speicher vorhanden ist. Die Wärmeübergabestation (WÜST) enthält somit einen Wärmeübertrager und deren Regelung und Regeleinrichtung, wie z.B. einen Differenzdruckregler. Sie sorgt dafür, dass die Wärme bestimmungsgemäß in Bezug auf Druck, Temperatur und Volumenstrom primärseitig verfügbar ist und an die Hauszentrale übergeben werden kann. Abbildung 2.8 zeigt einen Hausanschluss mit indirekter

Wärmeübergabe. Die Wärme wird über einen Wärmeübertrager an die Verbraucher*in gegeben. Der Volumenstrom auf der Primärseite wird über ein motorgesteuertes Ventil im Rücklauf auf die im Liefervertrag vereinbarte Leistung begrenzt. Die sekundärseitige Vorlauftemperatur kann außentemperaturabhängig über die lastseitige Pumpe reguliert werden. Die Ladung des Speichers sowie die restliche Übergabestation wird intern von der Hauszentrale geregelt und es kann im Normalfall nicht von dem Wärmelieferanten auf sie zugegriffen werden. [2]

Auslegung Wärmeübertrager

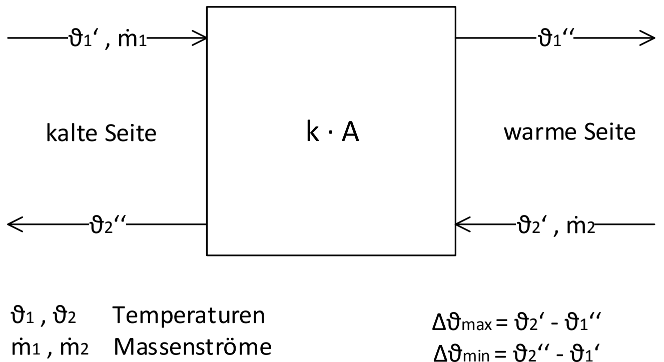

Abbildung 2.9.: Schema eines Gegenstrom-Wärmeübertragers [2]

Wärmeübertrager können unterschiedlich ausgeführt werden, es gibt beispielsweise Platten-, Rohrbündel- oder Spiralwärmeübertrager. Neben der Bauart spielt die Durchströmungsrichtung eine Rolle. Bei Gleichstrom-Wärmeübertragern strömen beide Flüssigkeiten in die gleiche Richtung, wohingegen die Fluide beim Gegenstrom-Wärmeübertrager in entgegengesetzte Richtungen strömen. Der Eintrittsort des einen Fluid befindet sich dann am Austrittsort des anderen Fluid.

Abbildung 2.9 zeigt ein Schema eines Gegenstrom-Wärmeübertragers mit den wichtigsten Massenströmen und Temperaturen. Für die Auslegung von Wärmeübertragern ist der Wärmedurchgangskoeffizient k multipliziert mit der Fläche A des Wärmeübertragers wichtig. Formel 2.1 zeigt die Berechnung, wobei es sich bei $\dot{Q}$ um den übertragenen Wärmestrom und bei $\Delta\vartheta_m$ um die mittlere Temperaturdifferenz handelt. Diese wird über den logarithmischen Mittelwert (Formel 2.2) von $\Delta\vartheta_{max}$ und $\Delta\vartheta_{min}$ berechnet. Hierbei

bezeichnen $\Delta\vartheta_{max}$ und $\Delta\vartheta_{min}$ die örtlichen Temperaturdifferenzen an den beiden Enden des Wärmeübertragers. $\Delta\vartheta_{max}$ ist die Temperaturdifferenz der warmen Seite und $\Delta\vartheta_{min}$ die der kalten Seite. Daraus ergibt sich schließlich Formel 2.3. [23]

$$k \cdot A = \frac{\dot{Q}}{\Delta\vartheta_m} \tag{2.1}$$

$$\Delta\vartheta_m = \frac{\Delta\vartheta_{max} - \Delta\vartheta_{min}}{\ln(\frac{\Delta\vartheta_{max}}{\Delta\vartheta_{min}})} \tag{2.2}$$

$$k \cdot A = \frac{\dot{Q}}{\frac{\Delta\vartheta_{max} - \Delta\vartheta_{min}}{\ln(\frac{\Delta\vartheta_{max}}{\Delta\vartheta_{min}})}} \tag{2.3}$$

Pumpenauslegung und Rohrdimensionierung

Für den Umlauf des Wassers im Fernwärmenetz werden Umwälzpumpen verwendet. Diese werden von einem Elektromotor angetrieben und ihre Drehzahl ist in der Regel stufenlos regelbar. Ungeregelte Pumpen passen ihren Volumenstrom und die Förderhöhe nicht an und sind deswegen für sich verändernde Lastzustände wenig geeignet. Geregelte Umwälzpumpen dagegen passen je nach Last ihre Förderhöhe an. Eine volumenstromabhängige Differenzdruckregelung und hierfür eine Differenzdruckmessung an den entscheidenden Verbraucher*innen (oftmals die am weitesten entfernten) ist für die Regelung notwendig. [2]
In der Modellierung dieser Arbeit wird auf die Druckberechnung verzichtet. Deswegen ist vor allem interessant, welchen maximalen Massenstrom $\dot{m}_{max}$die Pumpen erzeugen müssen und Druckdifferenzen können vernachlässigt werden. Formel 2.4 zeigt die Berechnung des maximalen Massenstroms aus dem maximalen Wärmestrom $\dot{Q}_{max}$, der Temperaturdifferenz $\Delta\vartheta$ und unter Berücksichtigung von der spezifischen Wärmekapazität von Wasser (c_p). [23]

$$\dot{m}_{max} = \frac{\dot{Q}}{\Delta\vartheta \cdot c_p} \tag{2.4}$$

Sowohl für den Aus- und Umbau von realen Fernwärmenetzen als auch für die Parametrierung von Modellen in der Simulationsplattform muss außerdem die Größe der Rohrinnendurchmesser bestimmt werden. In Anhang A.1 kann der Rohrinnendurchmesser auf Grundlage des maximal herrschenden Volumenstroms und den optimalen Strömungsgeschwindigkeiten für Rohrleitungen in mm abgelesen werden. Rohrinnendurchmesser

orientieren sich an genormten Nennweiten. Der Rohrinnendurchmesser in mm wird deswegen übertragen auf eine Nennweite, wobei im Zweifel die kleinere Nennweite gewählt werden kann, da die maximalen Volumenströme nur selten auftreten und sich dafür die zusätzlichen Materialkosten der Rohre im Vergleich zu der ansonsten erhöhten Pumpleistung nicht rechnen. [2]

Kommunikation mit Leitwarte

Abbildung 2.10 zeigt das Kommunikationsnetz im betrachteten Wärmenetz und an welcher Stelle die WÜST SPS (W-SPS) agiert.

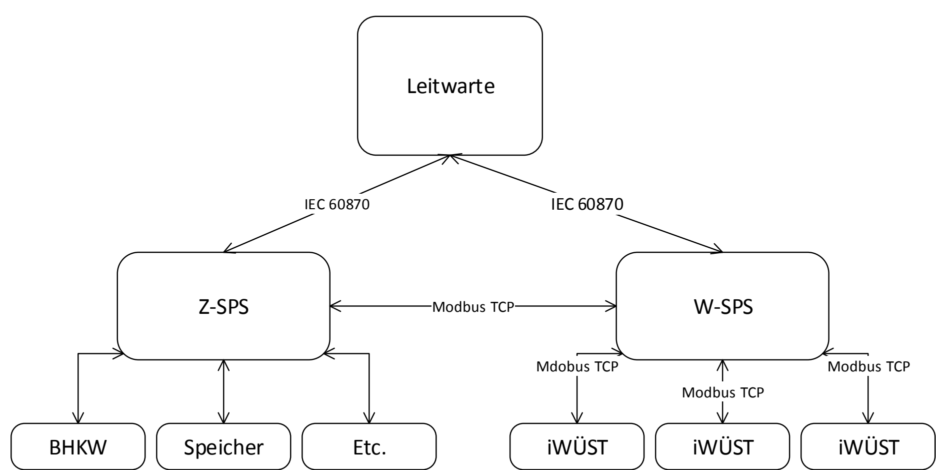

Abbildung 2.10.: Kommunikationsstruktur von Leitwarte zu Erzeugern und Verbrauchern

Es handelt sich um ein Ethernet Netzwerk unter Verwendung von unterschiedlichen Protokollen. Eine Leitwarte kommuniziert mit einer zentralen SPS (Z-SPS), der die Komponenten der Energiezentrale untergeordnet sind. Auf der anderen Seite gibt es die W-SPS, die mit einer Vielzahl an iWÜST verbunden ist. Die Z- und W-SPS kommunizieren sowohl mit der Leitwarte als auch direkt untereinander. Die Kommunikation zwischen Z-SPS und W-SPS und mit den iWÜST Unterstationen findet mittels des Protokolls Modbus TCP statt. Sollen Marktentscheidungen getroffen werden, dann kommunizieren W-SPS und Leitwarte miteinander. Bei Themen der Betriebssicherheit, wie einem Netzkollaps, soll die Z-SPS direkt mit der W-SPS kommunizieren.

Intelligente Wärmeübergabestation

Im Rahmen des SHGH Projektes an der HAW Hamburg soll eine intelligente WÜST (iWÜST) entwickelt werden. Diese soll im Vergleich zu der konventionellen WÜST extern steuerbar sein und die Regelung in den Aufgabenbereich des Wärmelieferanten fallen. Die Besonderheit der iWÜST ist außerdem eine hydraulischen Verschaltung des Wärmespeichers. Des Weiteren wird eine große Anzahl an Daten erfasst und durch eine intelligente Regelung und Kommunikation mit einer übergeordneten Steuerung soll auf verschiedene Situationen im Netz reagiert und ein wärmenetzdienlicher Betrieb ermöglicht werden.

Tabelle 2.2.: Szenarien der iWÜST (TWW) [24]

Szenario	**Beschreibung**
Inbetriebnahme	Prüfung aller Modulfunktionen, danach erfolgt Betriebsfreigabe
Minimale Temperatur	Sicherstellen einer minimalen TWW-Temperatur
Kommunikationsausfall	Keine Kommunikation mit der Leitwarte möglich; erhalten eines Basisbetriebs durch Vorgabe eines minimalen Speicherfüllstands
Netzkollaps	Das Netz ist instabil, d.h. die Sollvorlauftemperatur wird nicht erreicht. Netzstabilisierung durch minimale Abnahme
Funktionsprüfung	Sobald Mindestfüllstand überschritten ist, kann von der Energiezentrale jederzeit eine Funktionsprüfung für die Motorventile gestartet werden
Speicher überladen	Der Wärmespeicher wird maximal gefüllt, der Netzrücklauf wird möglichst aufgeheizt
Füllstandsbegrenzung	Begrenzung des Speicherfüllstandes, sobald die Temperaturdifferenz zwischen Speichervor- und -rücklauf zu klein wird
Leistungsbegrenzung	Leistungsbegrenzung, wenn vertraglich vereinbarte Leistungsabnahme überschritten
Rücklauftemperaturbegrenzung	Die Rückführung der Zirkulationstemperatur in den Rücklauf des Fernwärmenetzes wird mit einer Rücklaufbegrenzung verhindert
Normalbetrieb	Normalbetrieb; die Leitwarte sendet einen Füllstandsollwert, der kontinuierlich geregelt wird

Außerdem verschiebt sich bei der iWÜST im Gegensatz zu Abbildung 2.8 die Eigentumsgrenze des Fernwärmebetreibers, sodass dieser Zugang zu den Daten der Übergabestation und des Speichers hat und auf sie einwirken kann, um mehr Flexibilität im Netz zu realisieren. Im Rahmen des SHGH Projektes wurde die Notwendigkeit von bestimmten Regelszenarien erkannt und ein Konzept zur Umsetzung entwickelt. Dabei werden die Szenarien zwischen Trinkwarmwasser (TWW) und den Heizkreisen unterschieden. Tabelle 2.2 zeigt eine Übersicht der Regelszenarien des Trinkwarmwassers, die denen der Heizkreise stark ähneln. Eine detaillierte Regelbeschreibung sowie die Regelszenarien der Heizkreise findet sich in [24].

2.4. Simulationsplattform Jarvis

Jarvis ist eine Simulationsumgebung, die derzeit im Forschungsprojekt SHGH an der HAW Hamburg entwickelt wird. Es basiert auf der Programmiersprache Python und ist als verteiltes System konzipiert. Abbildung 2.11 zeigt den Aufbau der *Backend*-Architektur.

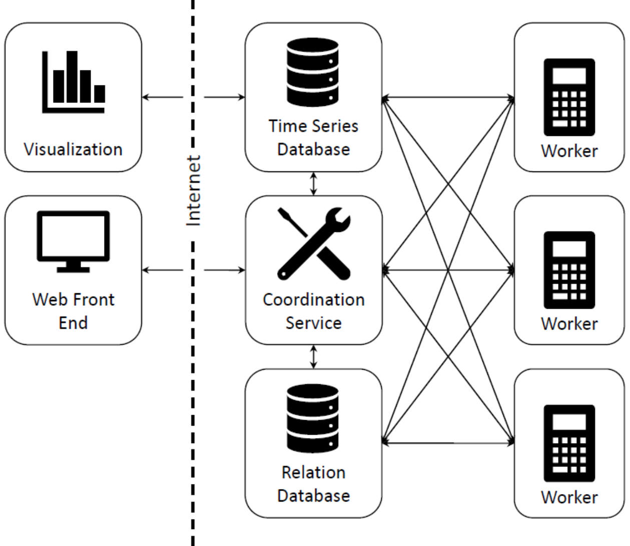

Abbildung 2.11.: *Backend*-Architektur von Jarvis [4]

Eine zentrale Einheit koordiniert eine Vielzahl an Diensten für Berechnungen, die sogenannten *Worker*. Als *Frontend* wird zunächst eine graphische Benutzeroberfläche verwendet, die allerdings im Laufe des Projektes durch eine webbasierte Oberfläche

abgelöst werden soll. Die graphische Benutzeroberfläche kann bereits eingeschränkt verwendet werden, bietet allerdings noch nicht alle benötigten Funktionalitäten. Eine relationale Datenbank speichert die Modellkomponenten, während die Ergebnisse und Eingangsdaten in eine Zeitreihen Datenbank geschrieben werden. Mit Hilfe von der Software Grafana können die Ergebnisse dann visualisiert werden. Die Simulation von gleichzeitig großen Modellen mit hohem Detailgrad kann durch ein verteiltes System auf mehreren Rechnern und Prozessoren ermöglicht werden. [4]
Die Simulationsumgebung ermöglicht sowohl Berechnungen anhand von Massenströmen als auch unter Berücksichtigung von Drücken im System. Monolithen sind im allgemeinen Elemente der Simulation, die im Hintergrund Berechnungen in verschiedenen Bereichen durchführen. Es handelt sich um die Bereiche Massenstromberechnung, Druckberechnung sowie einen SPS Monolithen zur Signalberechnung und -übertragung. Wird auf die Berechnung von Drücken verzichtet, dann müssen Ventile durch Pumpen und T-Stücke ersetzt werden, da für die Ventile eine Druckberechnung notwendig ist. Die Massenströme werden auf Basis der Graphentheorie und einer *Kanten-Knoten-Matrix* berechnet. Dabei bilden die T-Stücke Knoten und alle dazwischen liegenden Teile Kanten. Für die drucklose Berechnung darf keine Masche im System bestehen. Eine Pumpe sorgt für eine Unterbrechung der Kante, damit die Berechnung funktioniert. [15]
Für jeden Wasserkreislauf muss ein Massenstrom Monolith verwendet werden. Bei einem druckbehafteten System, wird die Berechnung von einem Druckmonolithen übernommen, der auf Basis eines interativen Verfahrens Drücke und die daraus resultierenden Massenströme berechnet. [24]

Modellaufbau und SPS Komponenten

Modelle in Jarvis können entweder in der webbasierten graphischen Oberfläche oder direkt in einer *YAML* Datei erstellt werden. *YAML* ist eine Datenserialisierung für verschiedene Programmiersprachen, die von Menschen leicht lesbar ist. Sie ist auch unter Python anwendbar. [25]
Wird das Modell im *Frontend* erstellt, dann kann eine *YAML* Datei aus dem grafischen Modell exportiert werden. Dies ist allerdings nicht erforderlich, um die Simulation zu starten. Abbildung 2.12 zeigt die grundlegende Struktur eines Projektes. Es enthält die *YAML* Datei, das *project creation script* sowie den Ordner *PLC Code*. Handelt es sich um ein rein hydraulisches Modell ohne Regelung, entfällt der *PLC Code*.

Die *YAML* Datei enthält das Grundgerüst des Modells. Enthalten sind alle verwendeten Komponenten sowie deren Parameter und Verbindungen untereinander. Die Komponenten müssen in ihr sowohl hydraulisch, als auch auf Signalebene verknüpft werden.

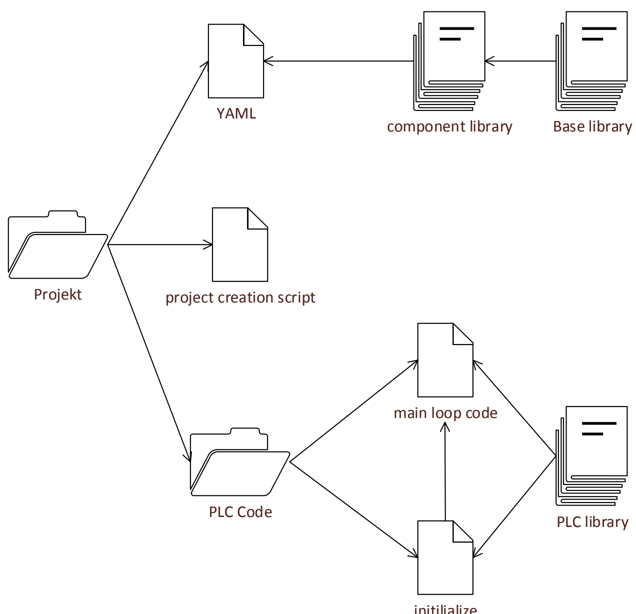

Abbildung 2.12.: Struktur eines Projektes in Jarvis

Es können nur Komponenten verwendet werden, die in der *component library* enthalten sind. In der *component library* sind hydraulische und regelungstechnische Komponenten definiert und deren Methodencode zur Berechnung von Temperaturen, Massenströmen, Druck und Weitergabe von Signalen abgelegt. Alle hydraulischen Komponenten enthalten mindestens die Methode *mass flow* und *temperature*. Die Berechnungen beruhen auf physikalischen und thermodynamischen Modellen. Die Komponenten verfügen zudem über Parameter, die innerhalb der *YAML* Datei angepasst werden können.
Regelungstechnische Komponenten empfangen Signale und geben diese ohne weitere Berechnungen an die angeschlossenen Komponenten weiter. Die Regelungskomponenten sind so aufgebaut, dass sie bezüglich der Anzahl der Kanäle sowie der Signalart so genau wie möglich abgebildet werden. Die Komponenten werden als *elements* in der relationalen Datenbank (PostgreSQL) gespeichert. Die *component library* nutzt die *base library* als Basis. In der *base library* sind Datentypen, Größen, Einheiten und Verbindungstypen sowie Monolithen und Element Klassen definiert.
In dem *project creation script* wird festgelegt, welche Daten eingelesen und in der Zeitreihen Datenbank abgespeichert werden sollen. Diese Zeitreihen können dann von Modellkomponenten während der Simulation eingelesen werden.
Der *PLC Code* besteht aus zwei Skripten, der *initialize* und dem *main loop code*. Diese sind mit der *PLC Library* verbunden. Um deren Funktion zu verstehen, muss zunächst

Umsetzung der virtuellen SPS in Jarvis erklärt werden. Die SPS soll möglichst realitätsnah aufgebaut sein. Deswegen besteht sie aus mehreren Komponenten: der CPU, verschiedenen analogen und digitalen Klemmen und einer Endklemme. Die Klemmen werden dann mit den Signalein- und ausgängen von anderen Komponenten innerhalb des Modells verbunden.
Abbildung 2.13 zeigt einen Ausschnitt aus der Programmierung einer realen SPS in CoDeSys. Im oberen Teil werden Variablen und Bausteine definiert. Diese können dann im unteren Teil für die Programmierung verwendet werden.

```
0001 PROGRAM Erzeuger
0002 VAR
0003     gen_pressure_flow_1_signal:REAL;
0004     gen_pressure_flow_1: REAL;
0005     gen_pressure_flow_2_signal: REAL;
0006     gen_pressure_flow_2: REAL;
0007
0008     gen_heartbeat_timer: TON;
0009     gen_heartbeat_timer_in: BOOL;
0010     gen_heartbeat_timer_q: BOOL;
0011
0012     gen_power_diff: WORD;
0013 END_VAR

0001 (*-----------------------------------measurement------------------------------------*)
0002
0003 gen_pressure_flow_1_signal:=WORD_TO_REAL(gen_pressure_flow_1_block);
0004 gen_pressure_flow_1:=(gen_pressure_flow_1_signal/32767)*10;
0005
0006 gen_pressure_flow_2_signal:=WORD_TO_REAL(gen_pressure_flow_2_block);
0007 gen_pressure_flow_2:=(gen_pressure_flow_2_signal/32767)*10;
0008
0009 gen_temp_flow_1:=(DINT_TO_REAL(gen_temp_flow_1_block))/10;
0010 gen_temp_flow_2:=(DINT_TO_REAL(gen_temp_flow_2_block))/10;
```

Abbildung 2.13.: Ausschnitt CoDeSys Programmierung

Basierend auf dieser Struktur sind auch die *initialize* und der *main loop code* aufgebaut. In der *initialize* werden Instanzen der Klassen aus der *PLC Library* erzeugt und Variablen initiiert, die dann im *main loop code* zur Programmierung verwendet werden können. In Anlehnung an CoDeSys Bibliotheken existiert die *PLC Library* in Jarvis. In ihr sind beispielsweise PID Regler als Bausteine enthalten.
Einstellungen, die in CoDeSys unter Kommunikationsparametern oder der Steuerungskonfigurationen vorgenommen werden, haben auf der virtuellen SPS ihr Pendant im Code der PLC Monolith Klasse. Die Skripte *initialize* und *main loop code* werden zunächst in der *initialize* Funktion der PLC Klasse als Parameter eingelesen. Die *process* Funktion liest dann zu jedem Zeitschritt die Signale der Klemmen ein, führt den *main loop code* aus und schickt die Ausgangssignale an die Klemmen.

3. Pymodbus Anwendung

Dieses Kapitel beschreibt die Anwendung von Pymodbus in einem Feldtest mit einem Raspberry Pi und dessen Integration in die Simulationsumgebung Jarvis.

3.1. Kommunikationstest im Feld

Im betrachteten Fernwärmenetz werden Lichtwellenleiter (LWL) verlegt, um in Zukunft Wärmeübergabestationen zur Kommunikation mit einer Leitwarte anzuschließen. Die Verlegung der LWL und der Anschluss der WÜST-Steuerung (SPS) kann zeitlich auseinander liegen. Deswegen soll eine Möglichkeit entwickelt werden, die Kommunikation über die LWL direkt nach deren Verlegung zu testen. Zu diesem Zweck sollen Raspberry Pi's als Stellvertreter für die WÜST fungieren und mit ihrer Hilfe ein Kommunikationstest durchgeführt werden.
Um den Kommunikationstest umzusetzen, werden fünf Raspberry Pi's verwendet. Vier davon sollen über ein Ethernet Netzwerk und Modbus TCP angebunden werden, der fünfte Raspberry Pi ist als Datenlogger vorgesehen und soll aus Sicherheitsgründen seriell angeschlossen werden und über Modbus RTU kommunizieren.
Jeder Raspberry Pi wird mit einem Python Skript versehen, das auf Basis der Pymodbus Bibliothek einen Server aufbaut. Durch diesen Server können Register gesetzt werden, die dann testweise von dem Laptop ausgelesen werden können. Quelltext 3.1 zeigt das Skript des ersten Raspberry Pi's. Dieses Skript wird mit leichten Veränderungen, wie z.B. Anpassung der *rasp_number*, auf die ersten vier Raspberry Pi's gespielt und beim Start dieser direkt ausgeführt.

Quelltext 3.1: Modbus TCP Server auf Raspberry Pi

```
1  from pymodbus.server.sync import StartTcpServer
2  from pymodbus.device import ModbusDeviceIdentification
3  from pymodbus.datastore import ModbusSequentialDataBlock
4  from pymodbus.datastore import ModbusSlaveContext, ModbusServerContext
5  from threading import Thread
6
7  rasp_number = 1
8  n_boolean = 1
9  n_floats = 2
10 port = 502
11
12 store = ModbusSlaveContext(hr=ModbusSequentialDataBlock(1, [rasp_number]*n_floats),co=ModbusSequentialDataBlock(1,
       [False]*n_boolean))
13 context = ModbusServerContext(slaves=store, single=True)
```

```
14 identity = ModbusDeviceIdentification()
15
16 def modbus_server_thread():
17     StartTcpServer(context, identity=identity, address=("", port))
18
19 thread = Thread(target=modbus_server_thread)
20 thread.start()
21 print("Server thread started")
```

Zuerst werden Teile der Pymodbus Bibliothek importiert und Parameter definiert. Danach werden die Datenblöcke (*ModbusSequentialDataBlock*) festgelegt und ihnen ein Initialwert, eine Anzahl der Register und ein Startregister zugewiesen. Der erste *ModbusSequentialDataBlock* hat beispielsweise das Startregister 1, den Initialwert *rasp_number* und eine *n_floats* große Anzahl an Registern. Die Datenblöcke werden dann im *ModbusSlaveContext* mit ihren Registerarten abgespeichert. Die Abkürzung *hr* steht für *holding registers* und *co* für *coils*. Der *ModbusSlaveContext* wird anschließend in den *ModbusServerContext* eingelesen und *single* auf *True* gesetzt, um festzulegen, dass es nur eine *unit id* gibt.

Der Server ist nun bereit und kann in einem Thread unter Zugabe der Adresse und des Ports gestartet werden. Als IP Adresse wird die statische IP des Raspberry Pi's verwendet, weswegen nur Anführungsstriche eingegeben werden müssen. Es ist notwendig, den Server als Thread zu starten, damit andere Prozesse nebenher weiterlaufen können, während der Server auf eine Anfrage wartet.

Als Gegenstück wird der Client benötigt, der die vier Server der Raspberry Pi's abfragt. Quelltext 3.2 zeigt den Aufbau.

Quelltext 3.2: Client zur Raspberry Pi Abfrage über Modbus TCP

```
1  from pymodbus.client.sync import ModbusTcpClient as ModbusClient
2  import os
3
4  port = 502
5  hostname_dict = {"10.100.47.173": 3, "10.100.47.174": 4, "10.100.47.175": 2, "10.100.47.176": 1}
6  hr_set_value = 17
7  co_set_value = True
8  hr1_error = True
9  hr2_error = True
10 co1_error = True
11
12 for hostname, rasp_number in hostname_dict.items():
13     client = ModbusClient(hostname, port=port)
14     client.connect()
15
16     # Read first holding register
17     succeeded = client.connect()
18     if succeeded:
19         print("\n Connection with {} successfully established.".format(hostname))
20         hr1 = client.read_holding_registers(0)
21
22         if hr1.isError():
23             print("Oops, something went wrong. Register 1 of {} could not be read.".format(hostname))
24         else:
25             if hr1.registers[0] == rasp_number:
26                 print("{} - Raspberry Pi Number: {}".
27                   format(hostname, hr1.registers[0]))
28                 hr1_error = False
29             else:
30                 print("Oops, something went wrong. The register 1 of {} does not contain the expected value. Please
                        check whether you are reading the right register.".format(hostname))
31                 hr1_error = True
32
33     # Write second holding register and read back
34
35         client.write_register(1, hr_set_value)
36         hr2 = client.read_holding_registers(1)
```

```
37
38          if hr2.isError():
39              print("Oops, something went wrong. Register 2 of {} could not be read.".format(hostname))
40          else:
41              if hr2.registers[0] == hr_set_value:
42                  hr2_error = False
43              else:
44                  print("Oops, something went wrong. The register 2 of {} does not contain the expected value. Please
                        check whether you are writing and reading the right register.".format(hostname))
45                  hr2_error = True
46
47      # Write coil in third register and read it back
48
49          client.write_coil(2, co_set_value)
50          co1 = client.read_coils(2, 1)
51
52          if co1.isError():
53              print("Oops, something went wrong. Register 3 of {} could not be read.".format(hostname))
54          else:
55              if co1.bits[0] == co_set_value:
56                  co1_error = False
57              else:
58                  print("Oops, something went wrong. The register 3 of {} does not contain the expected value. Please
                        check whether you are writing and reading the right register.".format(hostname))
59                  co1_error = True
60
61              if not hr1_error and not hr2_error and not co1_error:
62                  print("Hooray! Test for {} terminated successfully. \n".format(hostname))
63
64      else:
65          print("Connection with {} failed. Please check your connection.".format(hostname))
66          continue
67
68      client.close()
```

Zuerst wird ebenfalls ein Teil der Pymodbus Bibliothek importiert und Parameter werden angelegt. Es wird ein *dictionary* mit den IP Adressen und Nummern der Raspberry Pi's erzeugt, das in einer Schleife durchlaufen wird, sodass mit jedem Raspberry Pi auf die gleiche Weise kommuniziert wird. Zunächst wird ein Verbindungsaufbau zu dem Server des Raspberry Pi's versucht. Ist dieser erfolgreich, wird dies der Benutzer*in unter Angabe der IP mitgeteilt. Im nächsten Schritt wird das *holding register* mit der Raspberry Pi Nummer ausgelesen, um zu testen, ob es sich um das angenommene Gerät handelt. Danach wird ein weiteres *holding register* und ein *coil* geschrieben und anschließend zur Überprüfung gelesen. In jedem Schritt erfährt die Benutzer*in, ob der Test erfolgreich war und bekommt abschließend eine Nachricht über den Erfolg aller Testschritte.
Nun fehlt noch die Kommunikation über die serielle Schnittstelle für den fünften Raspberry Pi. Quelltext A.1 im Anhang zeigt die Programmierung des Servers. Er soll neben der Kommunikation über Modbus RTU über einen sogenannten *heartbeat* verfügen, der von dem Client abgefragt werden kann. Der *heartbeat* liegt auf einem Register, dass mit einer festgelegten Frequenz zwischen *True* und *False* wechselt. Der Client fragt diesen ab und erkennt, dass ein Kommunikationsfehler vorliegt, wenn der Wert des Registers sich nicht innerhalb der vorgegeben Zeit verändert. Ansonsten ist der Aufbau des Servers ähnlich zu dem für Modbus TCP. Es wird jedoch, anstatt einer IP Adresse und eines Ethernet Ports, eine Baudrate und ein serieller Port benötigt. Die Baudrate muss bei Server und Client übereinstimmen. Der serielle Port ist abhängig von der verwendeten Schnittstelle am Raspberry Pi.
Dem seriellen Client in Quelltext A.2 im Anhang wird ebenfalls eine Baudrate übergeben und ein serieller Port festgelegt. Zunächst wird der generelle Verbindungsaufbau getes-

tet. Ist dieser gelungen, dann wird mittels des *heartbeats* überprüft, ob die Verbindung aufrecht erhalten wird und ansonsten eine Fehlermeldung angezeigt.
Die Skripte auf den Raspberry Pi's sind damit vollständig und können als Kommunikationstestobjekt im Feld verwendet werden. Bis zum Abschluss dieser Masterthesis war ein Test der LWL im betrachteten Netz jedoch nicht möglich.

3.2. Simulationsumgebung

Der Aufbau des Servers und Clients ist in der Simulationsumgebung Jarvis ähnlich wie im Kommunikationstest beschrieben, wird allerdings noch erweitert. Server und Client sollen so eingebunden werden, dass sie im *main_loop_code*[1] der virtuellen SPS verwendet werden können, ohne sie wiederholt vollständig anlegen zu müssen.

Client und Server in *PLC Library*

Aus software-architektonischer Sicht ist es sinnvoll, eine Server und Client Klasse in der *PLC library* anzulegen. Quelltext 3.3 zeigt die Server Klasse.

Quelltext 3.3: Server in der *PLC Library*

```
70  class PyModbusServer:
71
72      FC_WRITE_MULTIPLE_REGISTERS = 16
73      FC_WRITE_COILS = 15
74      FC_READ_COILS = 1
75      FC_READ_HOLDING_REGISTERS = 3
76      heartbeat_instances = set()
77
78      def __init__(self, port=5020, name="Server", heartbeat=False, heartbeat_address=0, heartbeat_dt=0.5, n_registers
            =100, n_coils=100):
79          self.slave_id = 0X01
80
81          store = ModbusSlaveContext(hr=ModbusSequentialDataBlock(0, [0]*n_registers),
82                                     co=ModbusSequentialDataBlock(0, [False]*n_coils),
83                                     di=ModbusSequentialDataBlock(0, [False]*n_coils),
84                                     ir=ModbusSequentialDataBlock(0, [False]*n_coils))
85          self.context = ModbusServerContext(slaves=store, single=True)
86          identity = ModbusDeviceIdentification()
87          self.heartbeat_address = heartbeat_address
88          self.heartbeat = heartbeat
89          self.heartbeat_dt = heartbeat_dt
90          self.name = name
91          if self.heartbeat:
92              self.heartbeat_instances.add(self)
93
94          self.lost_heartbeat = False
95
96          def modbus_server_thread():
97              StartTcpServer(self.context, identity=identity, address=("", port))
98
99          thread = Thread(target=modbus_server_thread)
100         thread.start()
101
```

[1]Die SPS Programmierung findet im *main_loop_code* statt. Sie wird in jedem Zeitschritt ausgeführt und es können beispielsweise Eingangsklemmen gelesen und Ausgangsklemmen geschrieben sowie Regelungen umgesetzt werden.

```
102     @classmethod
103     def get_instances(cls):
104         return cls.heartbeat_instances
105 
106     def __str__(self):
107         return self.name
108 
109     def set_multiple_registers(self, address, values):
110         self.context[self.slave_id].setValues(fx=self.FC_WRITE_MULTIPLE_REGISTERS, address=address, values=values)
111 
112     def set_coils(self, address, values):
113         self.context[self.slave_id].setValues(fx=self.FC_WRITE_COILS, address=address, values=values)
114 
115     def get_holding_registers(self, address, count=1):
116         return self.context[self.slave_id].getValues(fx=self.FC_READ_HOLDING_REGISTERS, address=address, count=count
            )
117 
118     def get_coils(self, address):
119         return self.context[self.slave_id].getValues(fx=self.FC_READ_COILS, address=address)
120 
121     def get_holding_registers_double(self, address):
122         rq = self.get_holding_registers(address, count=2)
123         raw = struct.pack('>HH', rq[0], rq[1])
124         register_value = struct.unpack('>f', raw)
125         return register_value
126 
127     def set_multiple_registers_double(self, address, values):
128         raw = struct.pack('>f', values)
129         value_write = struct.unpack('>HH', raw)
130 self.set_multiple_registers(address, [value_write[0]])
131         self.set_multiple_registers(address + 1, [value_write[1]])
```

Der erste Teil gleicht dem des Kommunikationstest, allerdings wird der Server um einige Funktionen und den *heartbeat* ergänzt. Der Aufbau des *heartbeats* wird im weiteren Verlauf beschrieben. Die Server Klasse hat, neben einigen den *heartbeat* betreffenden Attributen, die Attribute *port*, *n_floats* und *n_coils*. Es werden Ports ab 5020 verwendet und dann hochgezählt. Normalerweise wird als Standardport für Modbus TCP Anwendungen der Port 502 verwendet. Wenn eine Kommunikation mit einer realen SPS aufgebaut werden soll, wird dieser auch weiterhin verwendet. Innerhalb der Simulation kommen jedoch höhere Ports zum Einsatz, da niedrigere Ports eventuell nicht für alle Nutzer*innen freigegeben sind. Die Attribute *n_registers* und *n_coils* geben an, wie viele Register des Typs *holding registers* und *coils* initial angelegt werden. Sie müssen nicht alle genutzt werden, sondern geben lediglich die maximale Anzahl der verwendbaren Register an.
Die Funktionen *set_multiple_registers* und *set_coils* dienen dazu, Register auf dem eigenen Server zu setzen. Mittels *get_holding_registers* und *get_coils* können Register des eigenen Servers abgerufen werden. Die Funktionen *set_holding_registers_double* dient dazu, einen Wert auf dem eigenen Server in zwei Register zu schreiben. Dies ist notwendig, wenn zum Beispiel *floats* mit einigen Nachkommastellen geschrieben werden sollen. Diese können zwar in Integer umgewandelt und in einem Register gespeichert werden, jedoch gehen dann Informationen verloren. Mit Hilfe von *set_multiple_registers_double* wird der Wert in zwei Register geschrieben, wodurch mehr Informationen übermittelt werden können. Um auf dem eigenen Server zwei Register zu lesen und aus diesem wieder einen Wert zu machen, wird die Funktion *get_holding_registers_double* verwendet. Die Funktion liest die Bits zweier Register aus, setzt diese wieder zusammen und gibt sie als Zahl aus. Alle Funktionen verwenden die

gängigen *function codes*.
Quelltext 3.4 zeigt die Client Klasse. Der grundlegende Aufbau wurde ebenfalls durch einen *heartbeat* Zusatz sowie durch eine Funktion zum Verbindungstest ergänzt. Außerdem dienen zwei Funktionen dem Lesen und Schreiben von *floats* in zwei Registern, beispielsweise auf realen SPS.

Quelltext 3.4: Client in der *PLC Library*

```
132 class PyModbusClient(ModbusClient):
133     heartbeat_instances = set()
134 
135     def __init__(self, host="localhost", port=5020, name="Client", heartbeat=False, heartbeat_address=10,
            heartbeat_dt=0.5):
136         super().__init__(host, port)
137         self.test_connection()
138         self.name = name
139         self.heartbeat = heartbeat
140         self.heartbeat_address = heartbeat_address
141         self.heartbeat_dt = heartbeat_dt
142         self.lost_heartbeat = False
143         if self.heartbeat:
144             self.heartbeat_instances.add(self)
145 
146     @classmethod
147     def get_instances(cls):
148         return cls.heartbeat_instances
149 
150     def __str__(self):
151         return self.name
152 
153     def test_connection(self):
154         while True:
155             succeeded = self.connect()
156             if succeeded:
157                 print("Connected successfully with IP {} on port {}.".format(self.host, self.port))
158                 break
159 
160     def read_holding_registers_flag(self, address):
161         rq = self.read_holding_registers(address, 2)
162         raw = struct.pack('>HH', rq.registers[0], rq.registers[1])
163         register_value = struct.unpack('>f', raw)[0]
164         return register_value
165 
166     def write_holding_registers_flag(self, address, register_value):
167         raw = struct.pack('>f', register_value)
168         value_write = struct.unpack('>HH', raw)
169         self.write_register(address, value_write[0])
170         self.write_register(address + 1, value_write[1])
```

Die Funktion zum Testen der Verbindung ist notwendig, da der Client einen Fehler meldet und die Simulation beendet, wenn kein Server erreichbar ist. Das bedeutet, dass der Server immer vor dem Client gestartet werden müsste, was in der Simulation schwierig zu kontrollieren ist. Um dieses Problem zu beheben, wird in der *init* die Funktion *test_connection* aufgerufen. Diese versucht solange eine Verbindung zu dem Server aufzubauen, bis diese besteht. Währenddessen wird der entstehende Fehler abgefangen. Erst wenn die Verbindung erfolgreich war, wird die Schleife verlassen und der Verbindungsaufbau als erfolgreich gemeldet.
Wenn in einer realen SPS Daten in Registern gespeichert werden, dann kommt es beispielsweise in der Programmierumgebung CoDeSys oftmals vor, dass diese unter einem Datentyp (z.B. *DWord*) gespeichert werden, der zwei Register belegt. Sollen diese Daten nun ausgelesen werden, dann reicht eine einfach Pymodbus Funktion dafür nicht aus und es geht ein Teil der Information verloren. Mit der Funktion

read_holding_registers_flag können deswegen zwei Register auf einem anderen Server gelesen und anschließend zusammengesetzt werden. Ebenso muss beim Schreiben dieser Datentypen in *write_holding_registers_flag* vorgegangen werden. Der Wert wird in zwei Teile zerlegt und diese in zwei Register eines anderen Servers geschrieben.

Umsetzung des *heartbeats* in der *PLC Library*

Der *heartbeat* wurde aus software-architektonischen Gründen im *process code* der SPS Monolithen Klasse angelegt. Der Code wird dadurch im Hintergrund in jedem Zeitschritt ausgeführt. Abbildung 3.1 gibt einen Überblick darüber, wie der *heartbeat* angelegt ist. Im Quelltext 3.5 sind die Details zu sehen. Für das Verständnis des *heartbeats* nicht relevante Teile der Klasse sind zur Übersichtlichkeit ausgeblendet und durch “[...]" ersetzt worden.

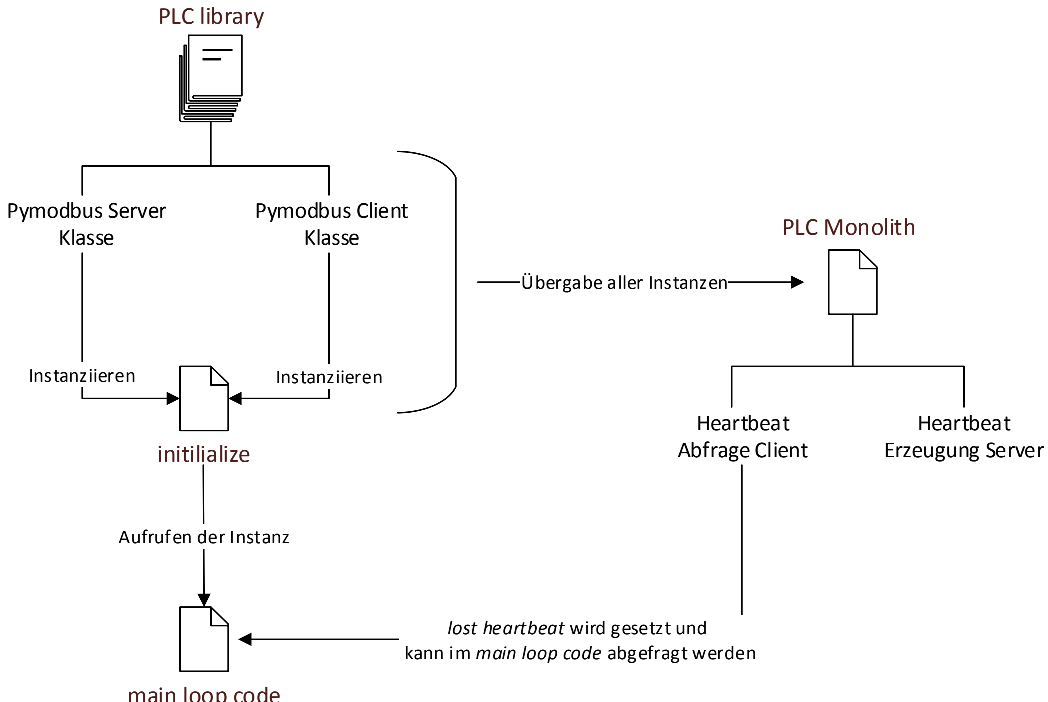

Abbildung 3.1.: Überblick über die Einbindung des *heartbeats*

Die Server und Client Klassen aus der *PLC Library* speichern mit Hilfe einer Klassenme-

thode alle erzeugten Instanzen, auf die dann im Monolithen zugegriffen werden kann. Als Attribute der Klassen muss die Anwender*in angeben, ob es einen *heartbeat* geben soll und wenn ja, welches Register mit welcher Frequenz zwischen *True* und *False* wechseln soll, bzw. erwartet wird.
In der *initialize* des Monolithen werden Listen mit den Attributen aller Klasseninstanzen erzeugt. Dies ermöglicht im *process code* des Monolithen einen spezifischen *heartbeat* (unterschiedliche Register und Frequenzen) für jeden Server und ebenso eine spezifische Abfrage der Clients. Wenn der Client in zwei Abfragen hintereinander den gleichen Wert liest, dann wird *lost_heartbeat* dieser Instanz auf *True* gesetzt. Die Variable *lost_heartbeat* kann innerhalb des *main_loop_codes* abgefragt werden.

Quelltext 3.5: Heartbeat im *process code*

```
172 class PLC(MonolithProcessor):
173     """
174     The PLC Monolith
175     """
176 [...]
177
178     def __init__(self, name, monolith_spec, simulation: misc.Simulation, ip):
179
180         self.plcPyInitCode = None
181         self.plcPyProcessCode = None
182
183         self.timestamp_wait_server = {}
184
185         self.timestamp_wait_client = {}
186
187         self.heartbeat_dt_server = {}
188
189         self.heartbeat_dt_client = {}
190
191         self.heartbeat_server = {}
192
193         self.heartbeat_client = {}
194
195         self.beat_old_client = {}
196
197         self.beat_new_client = {}
198
199         super().__init__(name=name, monolith_spec=monolith_spec, simulation=simulation, ip=ip)
200
201     def initialize(self):
202         [...]
203
204         for server in plc_lib.PyModbusServer.get_instances():
205             self.timestamp_wait_server[server.name] = datetime.strptime("1700", "%Y")
206
207         for client in plc_lib.PyModbusClient.get_instances():
208             self.timestamp_wait_client[client.name] = datetime.strptime("1700", "%Y")
209
210         for server in plc_lib.PyModbusServer.get_instances():
211             self.heartbeat_dt_server[server.name] = server.heartbeat_dt
212
213         for client in plc_lib.PyModbusClient.get_instances():
214             self.heartbeat_dt_client[client.name] = client.heartbeat_dt
215
216         for server in plc_lib.PyModbusServer.get_instances():
217             self.heartbeat_server[server.name] = False
218
219         for client in plc_lib.PyModbusClient.get_instances():
220             self.heartbeat_client[client.name] = False
221
222         for client in plc_lib.PyModbusClient.get_instances():
223             self.beat_old_client[client.name] = False
224
225         for client in plc_lib.PyModbusClient.get_instances():
226             self.beat_new_client[client.name] = False
227
228         for client in plc_lib.PyModbusClient.get_instances():
229             self.beat_new_client[client.name] = client.read_coils(client.heartbeat_address)
230
231     def process(self):
232         """
```

```
233         In each time step the transpiled plc process code (python) is executed
234         """
235         # Heartbeat on Server
236 
237         for server in plc_lib.PyModbusServer.get_instances():
238             if self.timestamp > self.timestamp_wait_server[server.name]:
239                 self.heartbeat_server[server.name] = not self.heartbeat_server[server.name]
240                 server.set_coils(address=server.heartbeat_address, values=[self.heartbeat_server[server.name]])
241                 self.timestamp_wait_server[server.name] = self.timestamp + timedelta(minutes=self.
                        heartbeat_dt_server[server.name])
242 
243         # Heartbeat Check from Client
244 
245         for client in plc_lib.PyModbusClient.get_instances():
246             if self.timestamp_wait_client[client.name] == datetime.strptime("1700", "%Y"):
247                 self.timestamp_wait_client[client.name] = self.timestamp + timedelta(minutes=(self.
                        heartbeat_dt_client[client.name]/2))
248             elif self.timestamp > self.timestamp_wait_client[client.name]:
249                 self.beat_old_client[client.name] = self.beat_new_client[client.name]
250                 self.beat_new_client[client.name] = client.read_coils(client.heartbeat_address)
251                 self.timestamp_wait_client[client.name] = self.timestamp + timedelta(minutes=self.
                        heartbeat_dt_client[client.name])
252 
253                 if self.beat_old_client[client.name].bits[0] == self.beat_new_client[client.name].bits[0]:
254                     client.lost_heartbeat = True
255                 else:
256                     client.lost_heartbeat = False
```

Übersicht über alle Funktionen zum Lesen und Schreiben

Tabelle 3.1 gibt abschließend einen Überblick über alle Funktionen, die zum Schreiben und Lesen von Daten über Modbus TCP innerhalb von Jarvis verwendet werden können. "value[0]" als Rückgabewert steht dabei für den ersten Eintrag in einer Liste. Pymodbus verwendetet diese Form standardmäßig, wenn eine Ausgabe von mehr als einem Wert möglich ist. Die Funktionen *get_coils* und *get_holding_registers* basieren direkt auf der Pymodbus Funktion *getValues* und verwenden diese Ausgabeform deswegen ebenfalls. Außerdem findet sich in Anhang A.5 und A.6 ein Beispiel, wie die Kommunikation zwischen zwei Soft-SPS umgesetzt wird.

Tabelle 3.1.: Übersicht über die Funktionen zum Schreiben und Lesen von Daten in Jarvis

Funktion	**Rückgabe- / Eingabewert**	**Aufgabe**
read_coils(self, address)	value.bits[0]	Coils von anderem Server lesen
read_holding_registers(self, address)	value.registers[0]	Register von anderem Server lesen
read_holding_registers_flag(self, address)	value	aus zwei Registern auf anderem Server *float* lesen
get_coils(self, address)	value[0]	Coils von eigenem Server lesen
get_holding_registers(self, address)	value[0]	Register von eigenem Server lesen
get_holding_registers_double(self, address, value)	value	aus zwei Registern auf eigenem Server *float* lesen
write_coils(self, address, values)	[value1,...]	Coils auf anderem Server schreiben
write_registers(self, address, values)	[value1,...]	Register auf anderem Server schreiben
write_holding_registers_flag(self, address, value)	value	*float* auf zwei Register auf anderem Server setzen
set_multiple_registers(self, address, values)	[value1,...]	Coils auf eigenem Server setzen
set_coils(self, address, values)	[value1,...]	Register auf eigenem Server setzen
set_multiple_registers_double(self, address, value)	value	*float* auf zwei Register auf eigenem Server setzen

4. Modellierung

In diesem Kapitel wird die Entwicklung und Umsetzung des Modells beschrieben, auf dessen Grundlage die Regelszenarien simuliert werden sollen. Abbildung 4.1 zeigt das hydraulische Schema des Gesamtmodells. Die Erstellung des Gesamtmodells erfolgt in drei Schritten. Das schrittweise Vorgehen beruht auf der Tatsache, dass kleine abgeschlossene Segmente des Gesamtmodells leichter geregelt und auf Funktionsfähigkeit geprüft werden können. Sind alle Segmente lauffähig, können sie zusammengeführt werden. Zunächst wird ein Erzeugermodell entworfen. Es enthält eine Energiezentrale mit verschiedenen Erzeugern.
Im zweiten Schritt wird ein Last- und Netzmodell erstellt, welches stellvertretend für eine Wärmeabnehmer*in zwei Lasten, einen Speicher und dessen Regelung enthält. Auf die detaillierte Regelung und Hydraulik der einzelnen WÜST wird hierbei verzichtet. Diese wurde in [Moritz] umgesetzt und getestet. Das hier verwendete Subsystem soll dazu dienen, die Kommunikation zwischen den SPS aufbauen und testen zu können, ohne dabei detaillierte Abläufe innerhalb der Übergabestation berücksichtigen zu müssen. Im dritten Schritt wird das Last- und Netzmodell mit dem Erzeugermodell zusammengeführt.
Es bedarf zu Beginn der Modellerstellung einer Übersicht über die enthaltenen Bestandteile des Systems. Basiert das Modell auf einem realen technischen System, dann steht im besten Fall ein Rohr- und Instrumentenfließschema (R&I) zur Verfügung. Sollte kein R&I existieren oder ist dieses unvollständig, dann muss es neu erstellt oder überarbeitet werden, bevor mit der eigentlichen Modellierung angefangen werden kann. Da nur der Bunker im Ansatz auf einer geplanten realen Anlage basiert, müssen die R&I größtenteils neu erstellt werden.
Des Weiteren sind zu Beginn der Masterarbeit nur Simulationen mit sogenannten Massenstrommodellen lauffähig. Daraus folgt, dass alle realen Komponenten, für deren Berechnung Druck essentiell ist, nicht innerhalb der Simulation genutzt werden können. Ein Standard R&I als Grundlage für die Modellerstellung kann daher zum aktuellen Zeitpunkt noch nicht in Jarvis umgesetzt werden, da nicht alle Komponenten in einer drucklosen Simulation zur Verfügung stehen. Beispielsweise werden elektrische Motorstellventile durch Umwälzpumpen ersetzt. Hieraus ergibt sich das Bedürfnis nach einem angepassten Fließbild. Dieses enthält nur Komponenten, die in Jarvis zur Verfügung stehen und angewendet werden sollen. Deswegen basieren die in den folgenden Unterkapiteln auf-

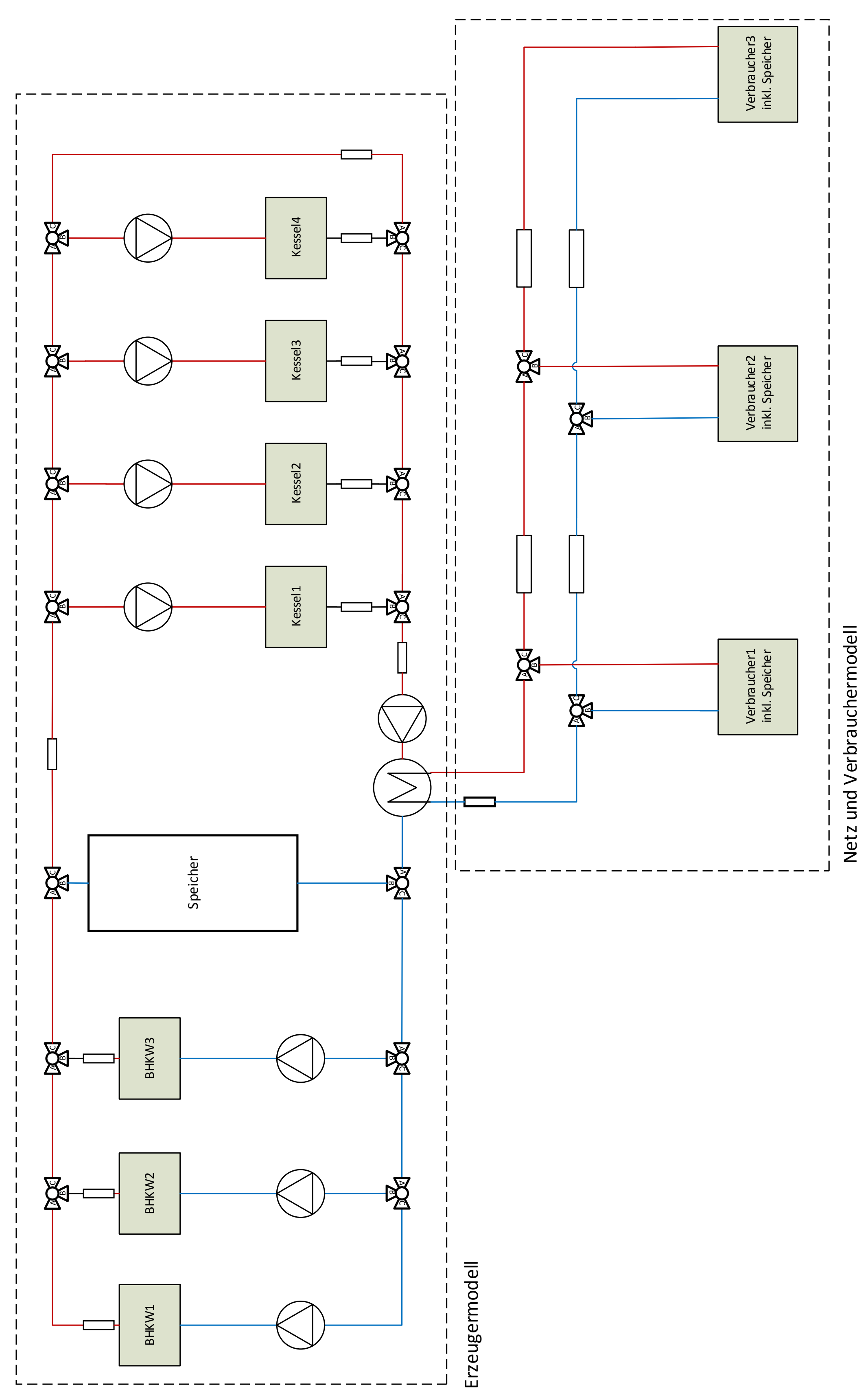

Abbildung 4.1.: Hydraulisches Schema des Gesamtmodells

geführten Schemata zwar auf dem Aufbau von R&I, sind jedoch den Bedürfnissen der Modellbildung angepasst. Tabelle 4.1 gibt eine Kurzbeschreibung der bestehenden verwendeten Komponenten in Jarvis. Rohre, T-Stücke und Temperatursensoren werden nicht näher erläutert.

Tabelle 4.1.: Beschreibung der verwendeten Komponenten

Komponenten in Fließbild	**Beschreibung der Komponenten in Jarvis**
CHP	wärmegeführtes BHKW mit analogem Steuersignal (4-20 mA), elektrische Leistung wird hier nicht berücksichtigt
Boiler	siehe oben, BHKW Komponente wird hier als Erdgaskessel genutzt, da dieses leicht geregelt werden kann
Pump	geregelte Pumpe, analoges Steuersignal (4-20 mA) wird in 0-100 % Pumpenleistung umgewandelt
Storage	Speicher mit je einem Stutzen oben und unten und einer einstellbaren Anzahl an Temperaturschichten, Schichttemperaturen als Ausgangssignal abrufbar
Load	Lastreihen werden eingelesen und diese abgefahren, bei fester Rücklauftemperatur wird der resultierende Massenstrom berechnet
Heat exchanger	Gegenstrom Wärmetauscher

In Anhang A.1 finden sich alle Parameter der PID-Regler sowie Einschaltverzögerungen und Mindestlaufzeiten nach Komponenten und Modellen geordnet.

4.1. Energiebunker als Erzeugermodell

Um den wärmenetzdienlichen Betrieb von WÜST erforschen zu können, ist die Einbeziehung und Betrachtung unterschiedlicher Wärmeerzeuger sinnvoll. Deswegen wird die Wärmeerzeugung im Modell durch eine Energiezentrale in einem Bunker dargestellt. Es handelt sich dabei um eine Energiezentrale in Hamburg Altona, die zurzeit durch den Verein "KulturEnergieBunkerAltonaProjekt" geplant wird. Zunächst wird die Hydraulik und Parametrierung des Modells erläutert und danach auf die Regelungssystematik eingegangen.

4.1.1. Hydraulik und Parametrierung

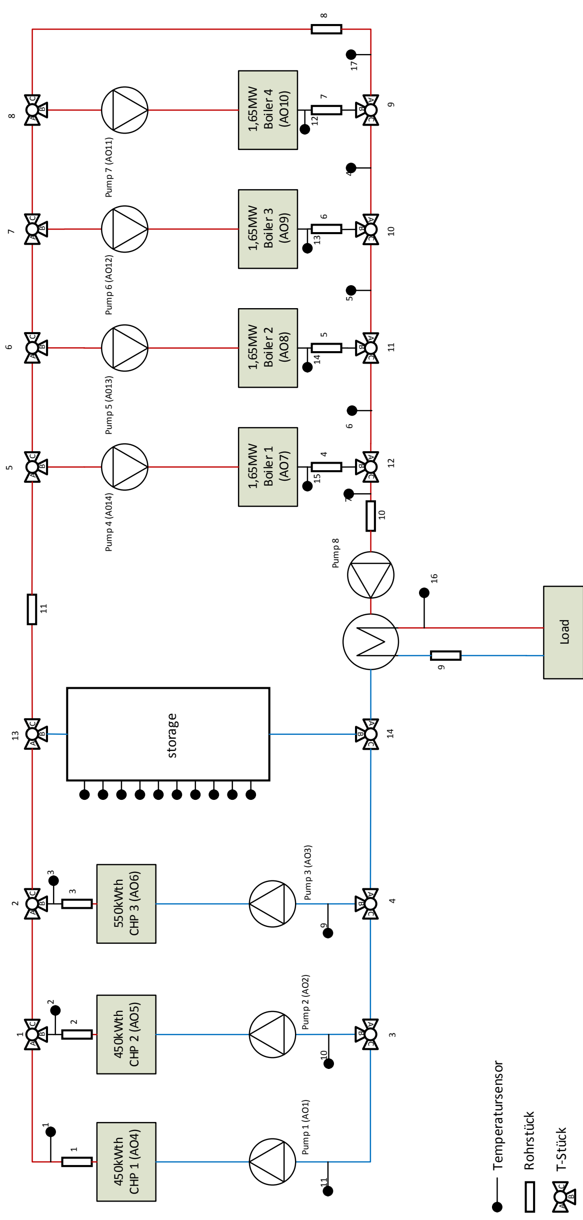

Abbildung 4.2.: Fließschema der Energiezentrale [3]

Das Fließschema in Abbildung 4.2 zeigt das Fließschema der Energiezentrale, welches auf Grundlage einer technischen Machbarkeitsstudie der Averdung Ingenieursgesellschaft (verweis machbarkeitsstudie) entstanden ist. Das Modell enthält drei Blockheizkraftwerke (BHKW) und vier erdgasbefeuerte Spitzenlastkessel. Zwei der BHKW werden mit Holzgas befeuert, das von einem Holzvergaser stammen soll, der mit regionaler Biomasse gespeist wird. Das dritte BHKW soll nach dem Bedarf am Strommarkt betrieben werden, da es den Bilanzkreis eines Energieversorgers ausgleichen soll, wenn dieser von seinen prognostizierten Windenergiemengen abweicht. [3]
Zur Umsetzung dieses Modells müssen die in Tabelle 4.1 aufgeführten Jarvis Komponenten um ein stromgeführtes BHKW ergänzt und die Speicherkomponente um die Berechnung des Füllstandes, dem State of Charge, (SOC) erweitert werden.

Komponentenerstellung und Anpassung

Die neue Komponente (textitCHPPlant_power_driven_from_timeseries) kann von dem bereits bestehenden BHKW in Jarvis abgeleitet werden. Das BHKW besitzt eine *initialize*, in der eine Initialisierung stattfindet, eine *temperature* und eine *mass flow* Methode zur Berechnung der Temperaturen und Massenströme. Der Aufbau des BHKW ist in der dazu gehörigen *YAML* Datei festgelegt. Die *initialize* und die *mass flow* Methode müssen nicht angepasst werden, die *temperature* Methode und die *YAML* Datei dagegen schon. Die Komponente des stromgeführte BHKW muss in der *YAML* Datei als Parameter eine Zeitreihe der elektrischen Leistungen erhalten und zusätzlich einen Parameter für die Quelle der Zeitreihe, damit festgelegt ist in welchem Format die *csv* Datei für die Zeitreihe eingelesen wird. Die Veränderungen im *YAML* können Quelltext A.4 und Quelltext A.3 im Anhang entnommen werden. Quelltext 4.1 zeigt die ursprüngliche *temperature* Methode des wärmegeführten BHKW, Quelltext 4.2 den angepassten Code des stromgeführten BHKW.

Quelltext 4.1: *temperature* Methode des bestehenden wärmegeführten BHKW

```
1  # Get parameters and inputs, set local constants
2  HEAT_FLOW_MAX = self.get_parameter(name="HEAT_FLOW_MAX")
3  mass_flow_quotients = self.get_input(name="mass_flow_quotient")
4  v_temperature_in = self.get_input(name="temp_return")
5  control_value = self.get_input(name="control_value")[0]
6  # CHP_COEFFICIENT = self.get_parameter(name="CHP_COEFFICIENT")
7
8  # Calculate outgoing temperature
9  heat_flow = HEAT_FLOW_MAX*(control_value - 0.004) / 0.016
10 d_heat = heat_flow * mass_flow_quotients[0][1]
11 mean_temperature = lib.get_mean_temperature(mass_flow_quotients=mass_flow_quotients, v_temperature_in=
         v_temperature_in, d_heat=d_heat)
12 v_temperature_out = lib.get_temperature_array(mass_flow_quotients=mass_flow_quotients, mean_temperature=
         mean_temperature)
13 # power = d_heat * CHP_COEFFICIENT
14
15 # Push outputs
16 self.push_output(name="temp_flow", value=v_temperature_out)
17 self.push_output(name="heat_flow", value=(heat_flow,))
```

Quelltext 4.2: *temperature* Methode des stromgeführten BHKW

```
# Get parameters and inputs, set local constants
series_name = self.get_parameter("POWER_SERIES")
series_source = self.get_parameter("POWER_SOURCE")
mass_flow_quotients = self.get_input(name="mass_flow_quotient")
v_temperature_in = self.get_input(name="temp_return")
power_series = self.get_input_from_series(name=series_name, source=series_source)
CHP_COEFFICIENT = self.get_parameter(name="CHP_COEFFICIENT")
HEAT_FLOW_MAX = self.get_parameter(name="HEAT_FLOW_MAX")
control_switch = self.get_input(name="control_value")[0]

# calculating outgoing temperature
if control_switch == 0.02:
    heat_flow = power_series / CHP_COEFFICIENT
else:
    heat_flow = 0

d_heat = min(heat_flow * mass_flow_quotients[0][1], HEAT_FLOW_MAX * mass_flow_quotients[0][1])
mean_temperature = lib.get_mean_temperature(mass_flow_quotients=mass_flow_quotients, v_temperature_in=
      v_temperature_in, d_heat=d_heat)
v_temperature_out = lib.get_temperature_array(mass_flow_quotients=mass_flow_quotients, mean_temperature=
      mean_temperature)

# Push outputs
self.push_output(name="temp_flow", value=v_temperature_out)
self.push_output(name="heat_flow", value=(heat_flow,))
```

Ergänzt wurde vor allem der Aufruf der Parameter *POWER SERIES* und *POWER SOURCE*, über die eine Datenreihe der elektrischen Leistung eingelesen wird. Der Wärmestrom *heat flow* wird hieraus mit Hilfe der Stromkennzahl (*CHP COEFFICIENT*) berechnet. Sollte der berechnete Wärmestrom den maximalen Wärmestrom des BHKW überschreiten, dann wird der maximale Wärmestrom verwendet. Das analoge Signal, das vorher zum Regeln des BHKW geregelt wurde, wird nun lediglich zum Ausschalten (4 mA) und Anschalten (20 mA) des BHKW verwendet.

Zur Berechnung des State of Charge des Speichers in Prozent wird Formel 4.1 verwendet und diese in die *temperature* Methode der Speicherkomponente eingefügt. Als T_{min} wird die Mindesttemperatur für den Vorlauf von 65 $^\circ C$ verwendet, da sich der Speicher nicht unter diesen Wert abkühlen darf. Als T_{max} wird die Vorlauftemperatur der BHKW von 90 $^\circ C$ verwendet.

$$SOC = \frac{\bar{T} - T_{min}}{T_{max} - T_{min}} * 100\% \tag{4.1}$$

Der Wert wird an einen Output der Komponente gegeben, damit dieser in Grafana dargestellt und zusätzlich in die Datenbank geschrieben wird.

Parametrierung

Um die generischen Jarvis Komponenten in spezifische Komponenten für diesen Anwendungsfall umzuwandeln, müssen diese parametriert werden. Die Leistungen der

Erzeuger können der Machbarkeitsstudie entnommen werden.(machbarkeitsstudie) Anders als in der Machbarkeitsstudie dargestellt, werden in diesem Modell jedoch Temperaturen zwischen 90 $°C$ im Vorlauf und minimal 50 $°C$ im Rücklauf verwendet. Dadurch lässt sich das Modell auf die Randbedingungen und Übergabestationen im betrachteten Netz beziehen.
Um die Pumpen und Rohre auszulegen, muss $\dot{m}_{max}$ basierend auf der Formel 2.4 berechnet werden. Außerdem muss $k \cdot A$ des Wärmeübertragers mit Hilfe von Formel 2.3 ausgelegt werden.

$$\dot{m}_{max} = \frac{\dot{Q}}{\Delta\vartheta \cdot c_p}$$

$$k \cdot A = \frac{\dot{Q}}{\frac{\Delta\vartheta_{max} - \Delta\vartheta_{min}}{\ln(\frac{\Delta\vartheta_{max}}{\Delta\vartheta_{min}})}}$$

Tabelle 4.6 stellt die Annahmen und Ergebnisse der Berechnung dar.

Tabelle 4.2.: Parameter Bestimmung des Bunkers

	Annahme	**Berechnung**	**Parameter im Modell**
Pump1&2	$\Delta\vartheta = 15K$	$\dot{m} = \frac{450kW}{4{,}2\frac{kJ}{kg \cdot K} \cdot 15K}$	$7,5\frac{kg}{s}$
Pump3	$\Delta\vartheta = 15K$	$\dot{m} = \frac{550kW}{4{,}2\frac{kJ}{kg \cdot K} \cdot 15K}$	$8\frac{kg}{s}$
Pump4-7	$\Delta\vartheta = 20K$	$\dot{m} = \frac{1650kW}{4{,}2\frac{kJ}{kg \cdot K} \cdot 20K}$	$20,5\frac{kg}{s}$
Pump8	$\dot{m}_{max} = \sum_{i=4}^{7} \dot{m}_i$	$\dot{m} = 4 \cdot 20,5\frac{kg}{s}$	$82\frac{kg}{s}$
Load	$\Delta\vartheta = 20K$ $\dot{Q}_{max} = 5000kW$	$\dot{m} = \frac{5000kW}{4{,}2\frac{kJ}{kg \cdot K} \cdot 20K}$	$60\frac{kg}{s}$
Wärmeübertrager	$\Delta\vartheta_{min} = 10K$ $\Delta\vartheta_{max} = 5K$ $\dot{Q}_{max} = 5000kW$	$k \cdot A = \frac{5000kW}{\frac{(5-10)K}{\ln(\frac{5K}{10K})}}$	$700\frac{kW}{K}$

Die maximalen Massenströme der Pumpen 1-7 und errechnen sich aus den maximalen Leistungen der Erzeuger und den angenommenen minimalen Temperaturdifferenzen bei dieser Leistung. Der maximale Massenstrom der Pumpe vor der Last ergibt sich aus der maximalen Last und dem minimal angenommen $\Delta\vartheta$. Für Pumpe 8 ergibt sich der

maximale Massenstrom aus der Summe der maximalen Massenströme der Pumpen 4-7. Für die Berechnung der Größe des Wärmeübertragers wird die maximale Last und eine angenommene minimale und maximale Temperaturspreizung des Wärmeübertragers verwendet.
Tabelle 4.3 stellt die anschließende Auslegung der Rohre dar. Der optimale Rohrinnendurchmesser kann im Diagramm in Anhang A.1 gegen den maximalen Volumenstrom abgelesen und nach Norm DIN EN ISO 6708 in Nennweiten umgerechnet werden, die im Modell als Parameter in mm angegeben werden. Die hydraulische Auslegung des Modells ist damit abgeschlossen.

Tabelle 4.3.: Auslegung der Rohre im Energiebunker

	Volumenstrom / $\frac{m^3}{h}$	**Rohrinnendurchmesser** / **DN**	**Parameter im Modell / m**
Pipe1&2	26	65	0,07
Pipe3	32	80	0,08
Pipe4-7	71	100	0,11
Pipe9	263	125	0,13

4.1.2. Regelung

Die Regelung der Energiezentrale soll in der Art gestaltet werden, dass der Anteil der erdgasbefeuerten Wärmeproduktion minimiert wird. Das stromgeführte BHKW bildet hierbei eine Ausnahme, da es zum Ausgleich des Strombilanzkreises eine wichtige Rolle spielt. Daraus ergibt sich eine Reihenfolge, in der die Erzeuger dazu geschaltet werden. Es gilt dabei die Vorlauftemperatur auf der Sekundärseite des Wärmetauschers jederzeit konstant zu halten, um eine durchgehende Versorgung der Endkund*innen gewährleisten zu können. Die größte Priorität unter den Erzeugern hat das stromgeführte BHKW, welches in Abhängigkeit von der erzeugten Windenergie geregelt wird. Es wird nur im Fall von Rücklauftemperaturen oberhalb von 65 $°C$ abgeschaltet.
Zunächst werden die Konzepte der Regelungen der Erdgaskessel und der BHKW kurz beschrieben. Danach folgt eine Darstellung der schrittweisen Einstellung der Gesamtregelung und darauf aufbauend die detaillierte Beschreibung der finalen Regelung anhand von Fließbildern.

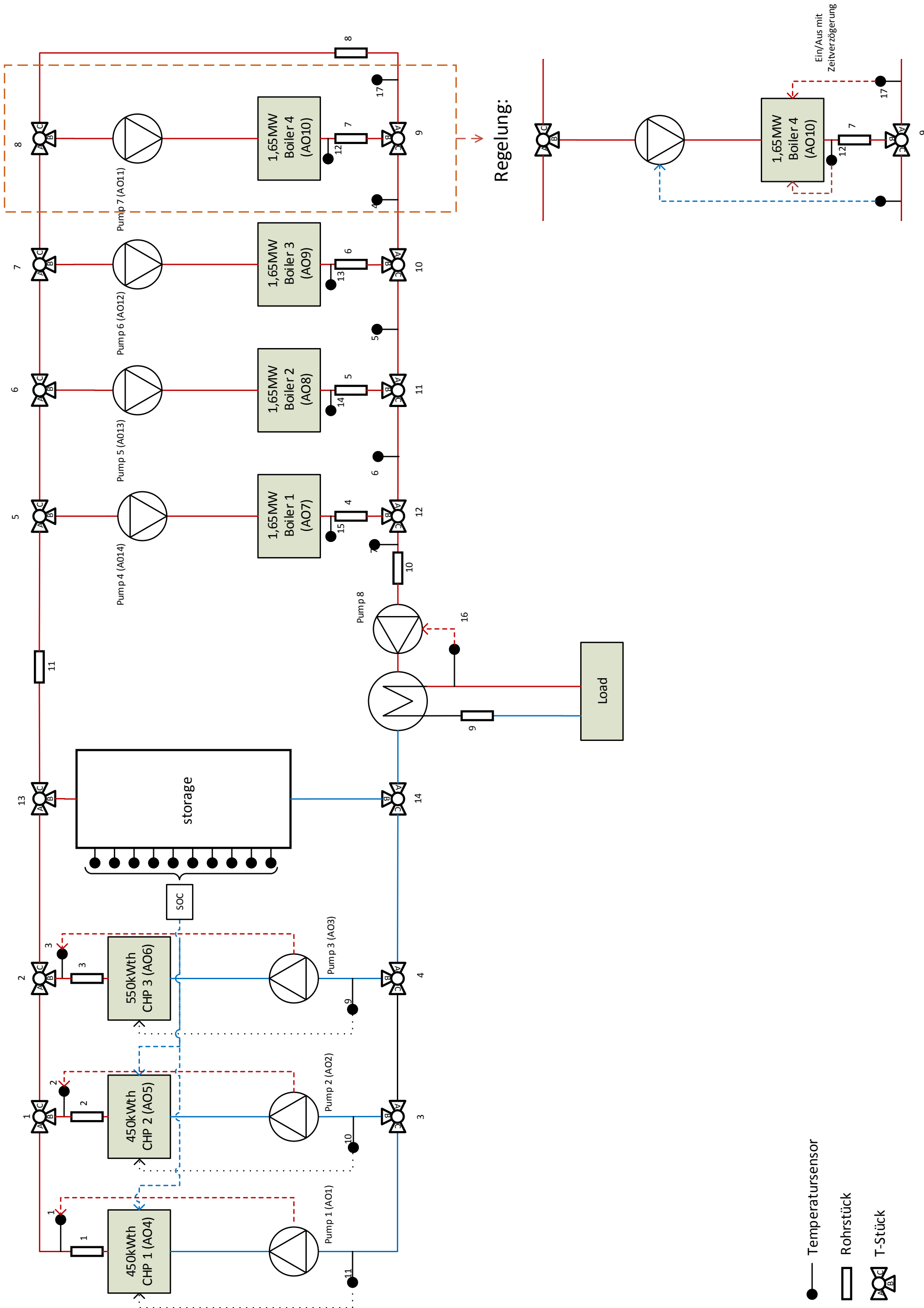

Abbildung 4.3.: Schematische Darstellung der Regelungen im Bunker

Regelungskonzept

Abbildung 4.3 zeigt das R&I des Regelungskonzepts. Der grundsätzliche Aufbau der Regelung soll bei allen vier Kesseln identisch sein. Die Vorlauftemperatur des Kessel soll über die Leistung des Kessels geregelt werden. Die Mischtemperatur am Ausgang C des T-Stücks [1] ergibt sich durch eine Leistungsregelung der Pumpe.
Des Weiteren soll es einen Schalter geben, der in Abhängigkeit von der Temperatur am Eingang A des T-Stücks den Kessel mit einer Zeitverzögerung anschaltet. Dies soll erfolgen, wenn die Temperatur über einen bestimmten Zeitraum unter eine vorgegebene Temperatur sinkt. Die Kessel unterscheiden sich in der Dauer der Einschaltverzögerung und den Sollwertvorgaben. *Boiler4* soll die geringste Einschaltverzögerung und die höchste minimale Temperaturvorgabe haben, sodass er als erstes eingeschaltet wird. Die Dauer der Einschaltverzögerung nimmt von *Boiler4* bis *Boiler1* zu und gegenläufig dazu sinkt die Vorgabe für die minimale Temperatur am Ausgang A des T-Stücks. Die Regelung von *CHP1* und *CHP2* soll gleich aufgebaut sein und sich nur durch eine Vorgabe des SOC Bereichs unterscheiden, in dem sie geregelt werden. Dahingegen unterscheidet sich die Regelung des *CHP3* etwas davon. Die Leistung der BHKW soll auf den SOC des Speichers geregelt werden. Die Vorlauftemperatur ergibt sich über eine Leistungsregelung der Pumpe. Außerdem soll es eine Abschaltung bei Rücklauftemperaturen oberhalb von 65 $°C$ geben. *CHP3* wird wie oben beschrieben in Abhängigkeit von der erzeugten Windenergie geregelt. Die Vorlauftemperaturregelung soll jedoch ebenfalls über die Pumpe erfolgen und es ist eine Abschaltung in Abhängigkeit von der Rücklauftemperatur vorgesehen. Übrig bleibt noch die Regelung der Vorlauftemperatur der Last, die durch *Pump8* erfolgt. Alle beschriebenen Regelungen sollen auf PID-Reglern basieren.

Umsetzung der Regelung

Die Einstellung der Regelung erfolgt in den in Tabelle 4.4 gezeigten Schritten. Ist die Regelung in einem Schritt erfolgreich, soll der nächste Schritt durchgeführt und ein weiterer Regelkreis ergänzt werden, bis schließlich die gesamte Regelung der Energiezentrale bestehen soll. In den Spalten der Tabelle stehen die Nummern der Temperatursensor, deren Temperatur von der Komponente in der ersten Spalte geregelt wird. Ist ein konstanter Wert eingestellt, dann ist im Falle der Pumpen der relative Massenstrom in Prozent angeben. Bei den Erzeugern werden konstante Werte als Leistung in Kilowatt angegeben. Größtenteils genügt bei den PID-Reglern der P-Anteil für einen stabilen Betrieb. *Tv* wird dann auf unendlich gesetzt, *Tn* auf null. Bei diesem Vorgehen stellt sich

[1] Gemeint ist jeweils das korrespondierende T-Stück des Kessels zur Vorlauftemperaturregulierung. Also zum Beispiel *boiler4* und *t_piece9*.

allerdings in Schritt 8 kein stabiler Zustand des Systems ein, bei dem alle Solltemperaturen erreicht werden können. Es kommt zu Wechselwirkungen zwischen den Pumpen 6,7 und 8, die sich nicht beheben lassen. Die Einstellschritte werden deswegen an diesem Punkt abgebrochen und das Regelungskonzept muss überdacht und die Regelung angepasst werden.

Vereinfachte Kesselregelung

Um die Wechselwirkungen zwischen den Pumpen zu umgehen, wird die Reglung der Erdgaskessel vereinfacht und die Pumpe zunächst konstant betrieben, während der Kessel die Mischtemperatur regelt. Abbildung 4.4 stellt die angepasste Regelung dar.

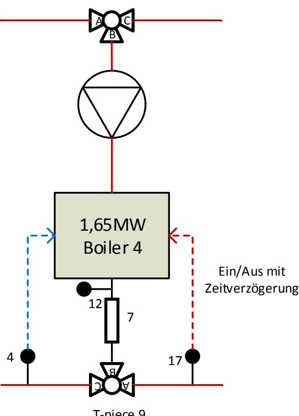

Abbildung 4.4.: Angepasste Kesselregelung

Durch die angepasste Regelung müssen die Einstellschritte erneut durchlaufen werden (Tabelle 4.5).

Tabelle 4.4.: Einstellschritte der Regelung der Energiezentrale

	Schritt 1	Schritt 2	Schritt 3	Schritt 4	Schritt 5	Schritt 6	Schritt 7	Schritt 8
Pump1	aus	aus	aus	aus	aus	aus	aus	aus
Pump2	aus	aus	aus	aus	aus	aus	aus	aus
Pump3	aus	aus	aus	aus	aus	aus	aus	aus
Pump4	aus	aus	aus	aus	aus	aus	aus	aus
Pump5	aus	aus	aus	aus	aus	aus	aus	aus
Pump6	aus	aus	aus	aus	aus	aus	aus	Temp.5
Pump7	50%	Temp.4	50%	Temp.4	Temp.4/ Temp.12	Temp.4/ Temp.12	Temp.4/ Temp.12	Temp.4/ Temp.12
Pump8	50%	50%	Temp.16	Temp.16	50%	Temp.16	Temp.16	Temp.16
Boiler1	aus	aus	aus	aus	aus	aus	aus	aus
Boiler2	aus	aus	aus	aus	aus	aus	aus	aus
Boiler3	aus	aus	aus	aus	aus	aus	aus	Temp.13 + Einschalt-verzögerung
Boiler4	Temp.12	Temp.12	aus	Temp.12	Temp.12	Temp.12	Temp.12 + Einschalt-verzöge-rung	Temp.12 + Einschalt-verzögerung
CHP1	aus	aus	aus	aus	aus	aus	aus	aus
CHP2	aus	aus	aus	aus	aus	aus	aus	aus
CHP3	aus	aus	aus	aus	aus	aus	aus	aus
Load	1400 kW	1400kW	1400 kW	1400 kW	1400 kW	1400 kW	1400 kW	1400 kW, nach 10 Minuten 2500 kW

Tabelle 4.5.: Einstellschritte nach Anpassung des Regelkonzeptes

	Schritt 1	Schritt 2	Schritt 3	Schritt 4	Schritt 5	Schritt 6	Schritt 7	Schritt 8	Schritt 9
Pump1	aus	aus	aus	aus	aus	aus	aus	aus	Temp.1
Pump2	aus	aus	aus	aus	aus	aus	aus	Temp.2	Temp.2
Pump3	aus	aus	aus	aus	aus	aus	Temp.3	Temp.3	Temp.3
Pump4	aus	aus	aus	aus	aus	50%	50%	50%	50%
Pump5	aus	aus	aus	aus	50%	50%	50%	50%	50%
Pump6	aus	aus	aus	50%	50%	50%	50%	50%	50%
Pump7	50%	50%	50%	50%	50%	50%	50%	50%	50%
Pump8	50%	Temp.16	Temp.16	Temp.16	Temp.16	Temp.16	Temp.16	Temp.16	Temp.16
Boiler1	aus	aus	aus	aus	aus	Temp.7 + Ein.	Temp.7 + Ein.	Temp.7 + Ein.	Temp.7 + Ein.
Boiler2	aus	aus	aus	aus	Temp.6 + Ein.	Temp.6 + Ein.	Temp.6 + Ein.	Temp.6 + Ein.	Temp.6 + Ein.
Boiler3	aus	aus	aus	Temp.5 + Ein.	Temp.5 + Ein.	Temp.5 + Ein.	Temp.5 + Ein.	Temp.5 + Ein.	Temp.5 + Ein.
Boiler4	Temp.4	Temp.4	Temp.4 + Einschaltverzögerung	Temp.4 + Ein.	Temp.4 + Ein.	Temp.4 + Ein.	Temp.4 + Ein.	Temp.4 + Ein.	Temp.4 + Ein.
CHP1	aus	aus	aus	aus	aus	aus	aus	aus	5 % v SOC v 75 %
CHP2	aus	aus	aus	aus	aus	aus	aus	5 % SOC v 75 %	10 % v SOC v 80 %
CHP3	aus	aus	aus	aus	aus	aus	200 kWel + Rücklaufabschaltung	200 kWel + Rücklaufabschaltung	200 kWel + Rücklaufabschaltung
Load	1400 kW	1400 kW	1400 kW	2000 kW	4000 kW	5000 kW	5000 kW	5000 kW	5000 kW

Die SOC Regelung von *CHP1* und *CHP2* führt bei den jetzigen Einstellungen jedoch dazu, dass der Speicher immer weiter entladen wird, bis der minimale SOC unterschritten wird und die BHKW abschalten. Die Erdgaskessel schalten erst dazu, wenn die Temperatur an Temperatursensor 17 unterschritten wird. Dies führt dazu, dass der Speicher entladen wird und die die BHKW ausgeschaltet werden, während die Erdgaskessel hochfahren. Dieses unerwünschte Verhalten kann verhindert werden, indem die minimale Grenze des SOC bei beiden BHKW auf 0 % gesetzt wird.

Problem - Einbruch der Vorlauftemperatur

Des Weiteren tritt im Grenzfall der maximalen Leistungsabnahme von 5000 kW ein unerwünschtes Regelverhalten auf. Abbildung 4.6 zeigt den Verlauf der Vorlauftemperatur der Last. Die Temperatur fällt plötzlich um fast 20 K ab. Dies liegt daran, dass die Vorlauftemperatur auf der Bunkerseite zu niedrig ist, was in Abbildung 4.5 zu sehen ist. Die Stufen im langsamen Anstieg der Temperatur sind der Zeitverzögerung der Kessel geschuldet.

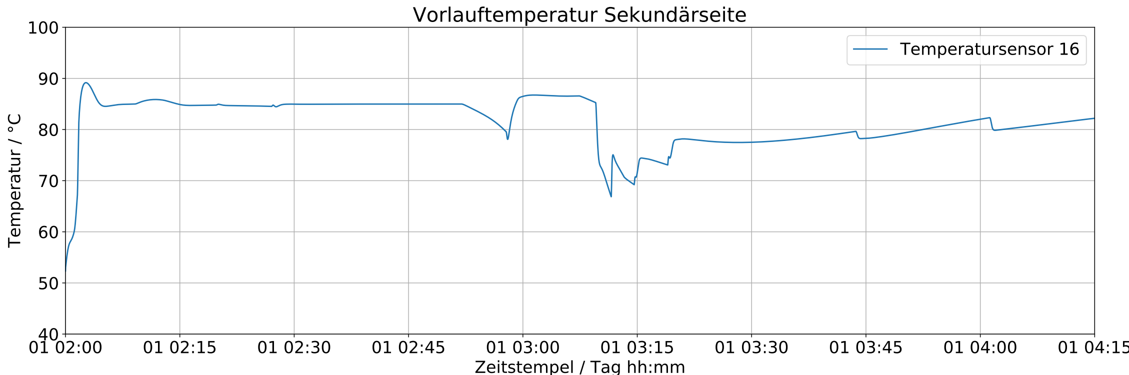

Abbildung 4.5.: Einbruch der Vorlauftemperatur der Last

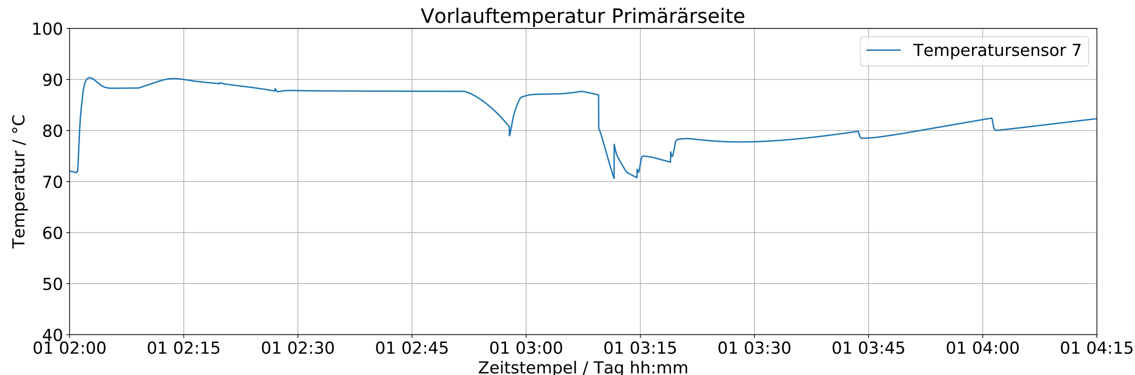

Abbildung 4.6.: Einbruch der Vorlauftemperatur auf der Primärseite des Wärmetauschers

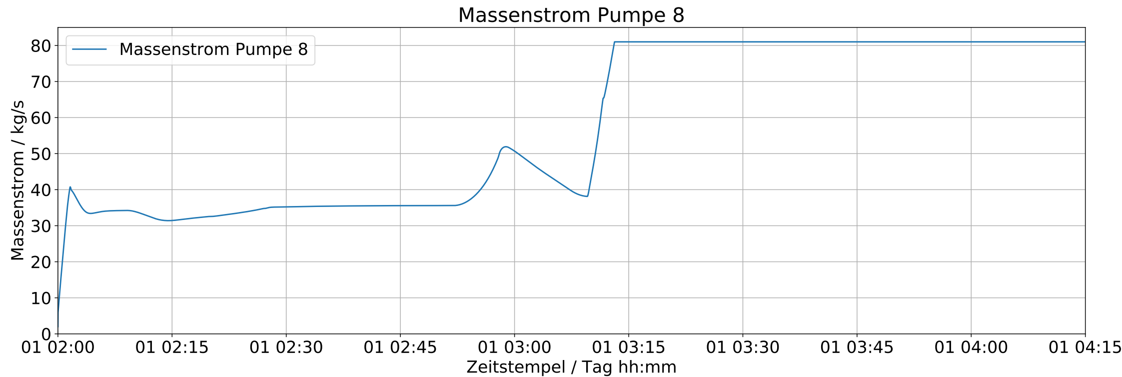

Abbildung 4.7.: Sprunghaftes Ansteigen des Massenstroms von Pumpe 8

Abbildung 4.7 zeigt, dass daraufhin der Massenstrom an Pumpe 8 sprunghaft ansteigt, da die Regelung durch erhöhten Durchfluss eine höhere Temperatur erreichen will. Dies ist jedoch nicht möglich, da die primärseitige Vorlauftemperatur zu niedrig ist. Die Last sorgt durchgehend für eine konstante Leistungsabnahme und konstante Rücklauftemperatur auf der Sekundärseite des Wärmeübertragers. Dadurch und durch den großen Unterschied zwischen dem Massenstrom auf der Primär- und Sekundärseite des Wärmetauschers steigt die Rücklauftemperatur auf der Primärseite. Abbildungen 4.8 und 4.9 zeigen, dass dies wiederum dafür sorgt, dass *CHP1* und *CHP2* abschalten, wodurch der Effekt der niedrigen Vorlauftemperaturen nur verstärkt wird und mehr Erdgaskessel dazu geschaltet werden, um dies auszugleichen.

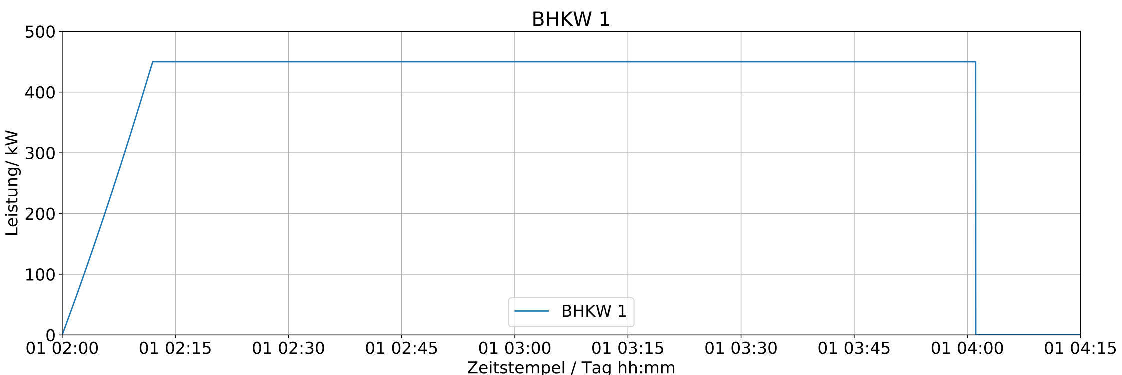

Abbildung 4.8.: Abschaltung BHKW1 aufgrund von zu hohen Rücklauftemperaturen

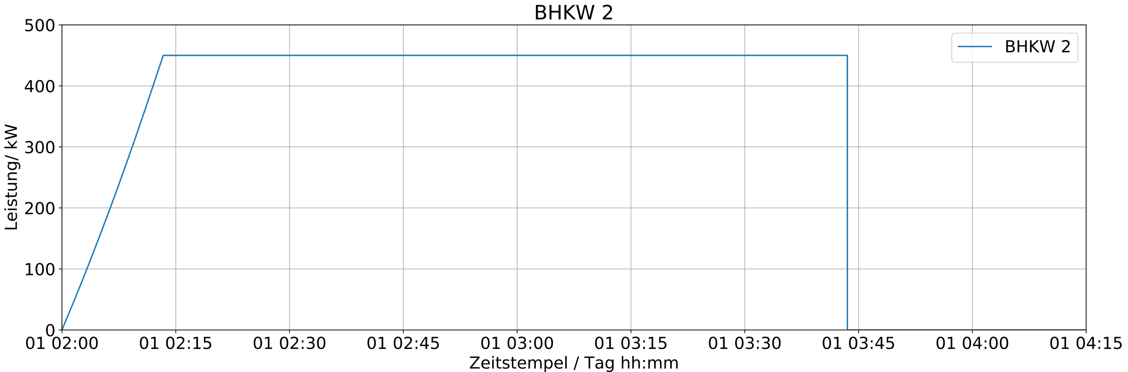

Abbildung 4.9.: Abschaltung BHKW2 aufgrund von zu hohen Rücklauftemperaturen

Das Problem basiert grundsätzlich auf folgenden Gegebenheiten. Der Wärmestrom auf der Primärseite muss dem auf der Sekundärseite des Wärmetauscher entsprechen, woraus sich durch das Verhältnis der Massenströme ein Verhältnis der Temperaturdifferenzen zueinander ergibt, wie die folgenden Formeln zeigen.

$$\dot{Q} = \dot{m} \cdot \Delta\vartheta \cdot c_p \tag{4.2}$$

$$\dot{m}_1 \cdot \Delta\vartheta_1 \cdot c_p = \dot{m}_2 \cdot \Delta\vartheta_2 \cdot c_p \tag{4.3}$$

$$\Delta\vartheta_1 = \frac{\dot{m}_2 \cdot \Delta\vartheta_2}{\dot{m}_1} \tag{4.4}$$

Wird der Massenstrom $\dot{m}_1$ auf der Primärseite größer und zeitgleich sinkt $\Delta\vartheta_2$, dann verringert sich auch das $\Delta\vartheta_1$ primärseitig. Das bedeutet, dass die Rücklauftemperatur steigt.
Um dieses Problem zu beheben, müssen zwei Bereiche bearbeitet werden. Zum einen sollte betrachtet werden, warum die Vorlauftemperatur primärseitig einbricht und wie dies durch schnelleres Gegensteuern der Erdgaskessel behoben werden kann. Zum anderen muss eine Lösung gefunden werden, um zu verhindern, dass eine sinkende primärseitige Vorlauftemperatur zu einer starken Erhöhung der Rücklauftemperaturen und Abschaltung der BHKW führt.

Lösung - Einbruch der Vorlauftemperatur

Hierzu gibt es drei Lösungsansätze, die kurz erläutert werden.

- Annahmen zur Berechnung des Wärmeübertragers und der Pumpen überprüfen
- Reduzierung der Pumpleistung bei primärseitigen Vorlauftemperaturen unter dem Sollwert
- Sollwertbegrenzung

Unter anderem durch den großen Unterschied zwischen den Massenströmen auf der Primär- und der Sekundärseite wird die Rücklauftemperatur angehoben. Um diese anzugleichen soll für den ersten Lösungsweg betrachtet werden, ob die maximalen Massenströme der Pumpe 4-7 und somit auch der Pumpe 8 notwendig sind. Falls sie grundsätzlich beibehalten werden müssen, soll über den zweiten Lösungsweg eine Begrenzung der Pumpleistung statt finden, wenn auf der Primärseite des Wärmetauschers Temperaturen unterhalb des Sollwertes für die Sekundärseite anliegen.
Als dritte Option kann eine Sollwertbegrenzung verwendet werden. Das bedeutet, dass die Solltemperatur auf der Sekundärseite sich nach der Temperatur auf der Primärseite richtet und beispielsweise immer zwei K über dieser liegt. Dies könnte allerdings durch den Modellaufbau dazu führen, dass eine Abwärtsspirale entsteht und die Vorlauftemperatur immer weiter sinkt. Deswegen wird diese Option als letzte Möglichkeit betrachtet, falls in den ersten beiden Fällen keine Verbesserung erzielt wird.
Es werden die Annahmen in Tabelle 4.6 betrachtet. Für Pumpe 4-7 wurde angenommen, dass bei maximaler Leistung ein $\Delta\vartheta$ von 20 K anliegt. Basierend auf dieser Annahme ergibt sich der maximale Massenstrom. Es wurde davon ausgegangen, dass die Temperatur durch die BHKW und den Speicher bereits um 15 K angehoben wurde. Das $\Delta\vartheta$ an der Last beträgt maximal 35 K. Es erscheint durchaus legitim davon auszugehen, dass die Rücklauftemperatur des Kessels nur um wenige $^{\circ}C$ angehoben werden konnte und das $\Delta\vartheta$ der Erdgaskessel weiter an 35 K anzunähern. Deswegen soll nun

von einem $\Delta\vartheta$ von 30 K ausgegangen werden. Tabelle (ref) zeigt die Ergebnisse der darauf folgenden Berechnungen.

Tabelle 4.6.: Angepasste Parametrierung der Bunkerkomponeten

Pump4-7	$\Delta\vartheta = 30K$	$\dot{m} = \frac{1650kW}{4{,}2\frac{KJ}{kg \cdot K} \cdot 30K}$	$13\frac{kg}{s}$
Pump8	$\dot{m}_{max} = \sum_{i=4}^{7} \dot{m}_i$	$\dot{m} = 4 \cdot 13\frac{kg}{s}$	$52\frac{kg}{s}$

Pumpe 8 wird grundsätzlich verkleinert und daraufhin überprüft, ob das Problem behoben werden konnte. Dabei zeigt sich, dass zunächst ein weiteres Problem auftritt.

Problem - Massenstromumkehr

Es kommt zu einer Massenstromumkehr, sodass das Wasser von den Kesseln ausgehend nicht Richtung Pumpe 8 fließt, sondern kurzzeitig über Temperatursensor 17 zurückfließt und diesen erwärmt. Abbildung 4.10 zeigt die Temperatur am Temperatursensor 17. Im Vergleich dazu ist die Temperatur an Sensor 18 in Abbildung 4.11 dargestellt. An Sensor 17 ist eine eindeutige Temperaturspitze zu erkennen. Da diese an Temperatursensor 18 nicht auftritt, muss die plötzlich erhöhte Temperatur aus der Richtung der Erdgaskessel kommen.

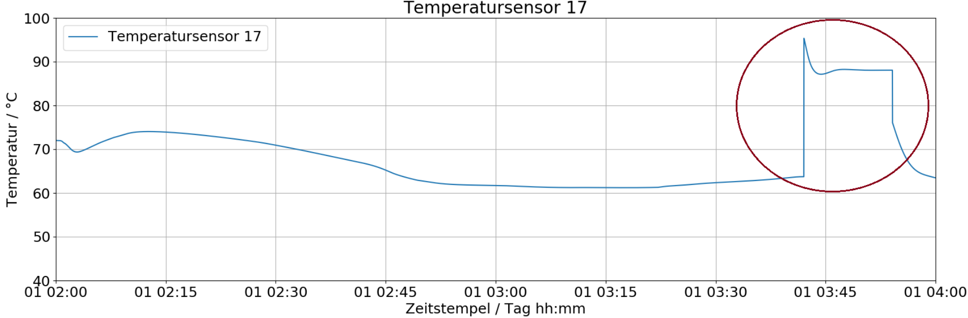

Abbildung 4.10.: Temperaturspitze an Temperatursensor 17

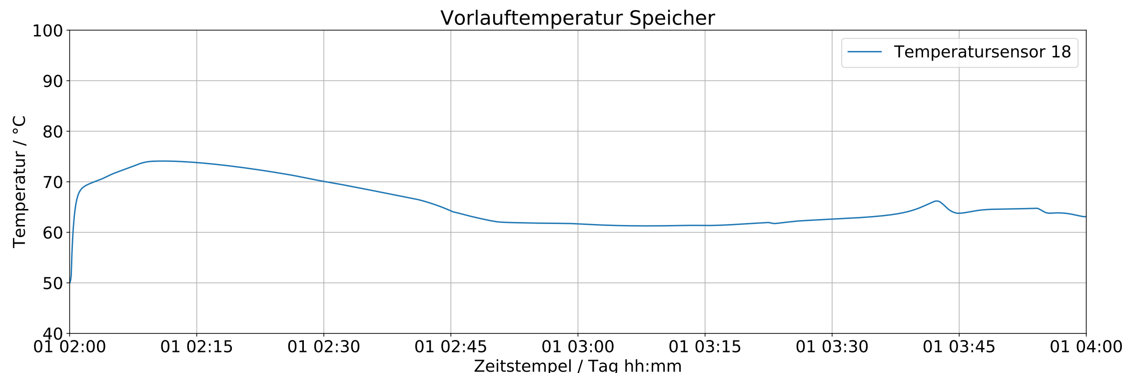

Abbildung 4.11.: Vorlauftemperatur Speicher an Temperatursensor 18

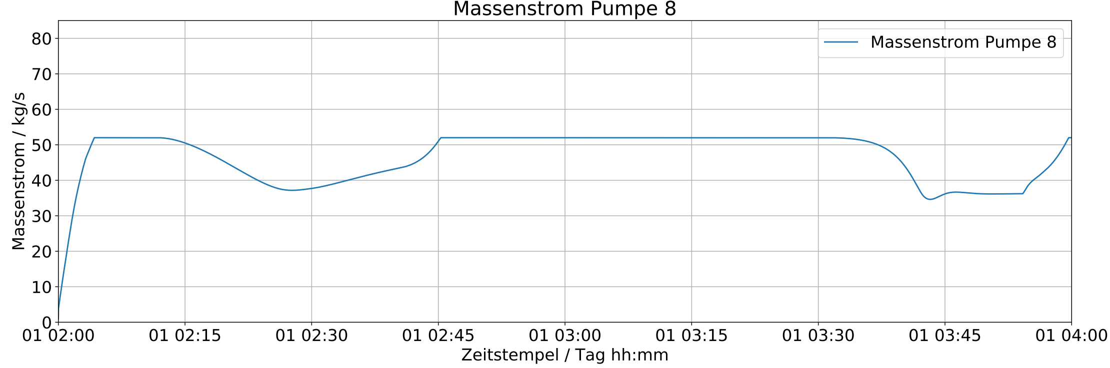

Abbildung 4.12.: Massenstrom Pumpe 8

Dies hat zur Folge, dass die Kessel ihre Leistungen reduzieren, beziehungsweise ganz abschalten. Diese Massenstromumkehr hängt damit zusammen, dass Pumpe 4-7 mit konstanter Leistung laufen. Wenn alle Kessel laufen, sind auch alle Pumpen angeschaltet und haben einen gesamten Massenstrom von 40 $\frac{kg}{s}$. Wird dann die Drehzahl von Pumpe 8 (Abbildung 4.12) reduziert und der sich ergebenden Massenstrom ist kleiner als die Summe der Massenströme der Pumpen, dann dreht sich der Massenstrom um.

Lösung - Massenstromumkehr

Die ursprüngliche Regelungsvereinfachung und das Feststellen der Pumpen kann nicht aufrecht erhalten werden. Um weiterhin sich beeinflussende Regelkreise zu vermeiden,

wird die Leistung der Pumpe an die des Kessels gekoppelt. So sind die Massenströme variabel, haben aber keinen starken Einfluss auf die Regelung der Kessel. Abbildung 4.13 und 4.14 zeigen, dass diese Maßnahme erfolgreich ist und die Massenstromumkehr effektiv verhindert wird.

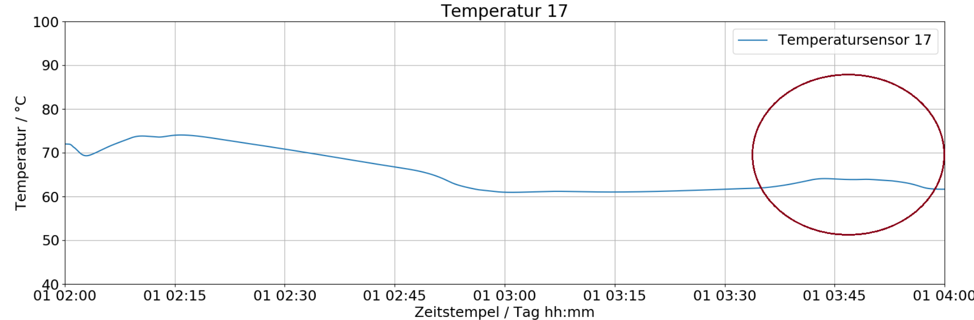

Abbildung 4.13.: Temperatur an Temperatursensor 17 bei variablem Massenstrom der Pumpe

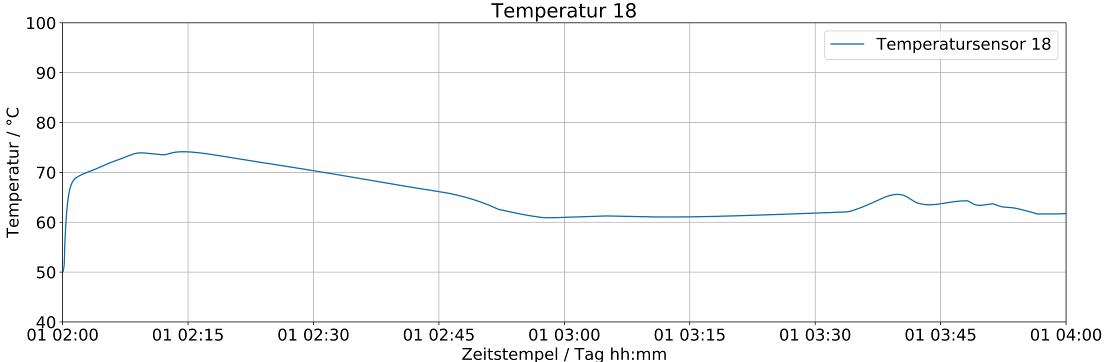

Abbildung 4.14.: Temperatur an Temperatursensor 18 bei variablem Massenstrom der Pumpe

Finale Regelung

Nun bleibt zu überprüfen, ob die Verkleinerung der Pumpe zu einem stabileren Regelverhalten beigetragen hat. Zusätzlich kommt die Synchronisation der Pumpen mit der BHKW Regelung der Gesamtregelung zu Gute. Abbildung 4.15 zeigt, dass die Maßnahmen erfolgreich waren. Im Grenzfall mit einer maximalen Last von 5000 kW

sind die Temperaturen nicht jederzeit konstant, aber ausreichend stabil. Abbildung 4.16 und 4.17 und zeigen, dass die Regelung in niedrigeren Lastbereichen durchgehend verlässlich läuft.

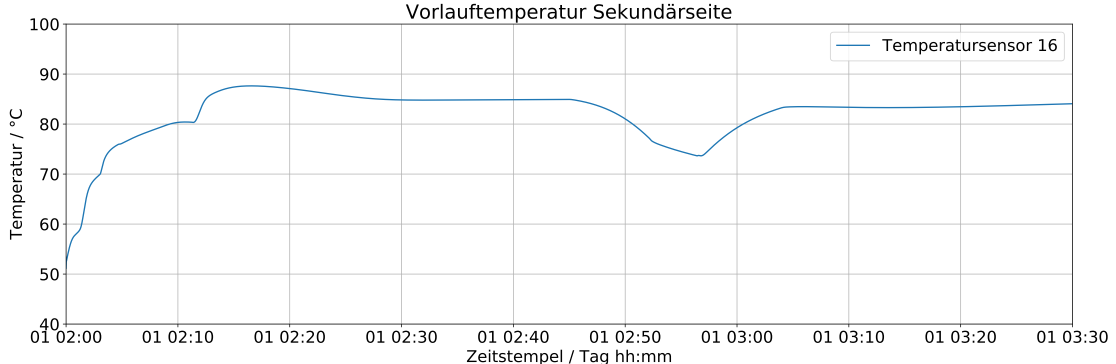

Abbildung 4.15.: Vorlauftemperatur Primärseite nach Verkleinern der Pumpe bei 5000 *kW*

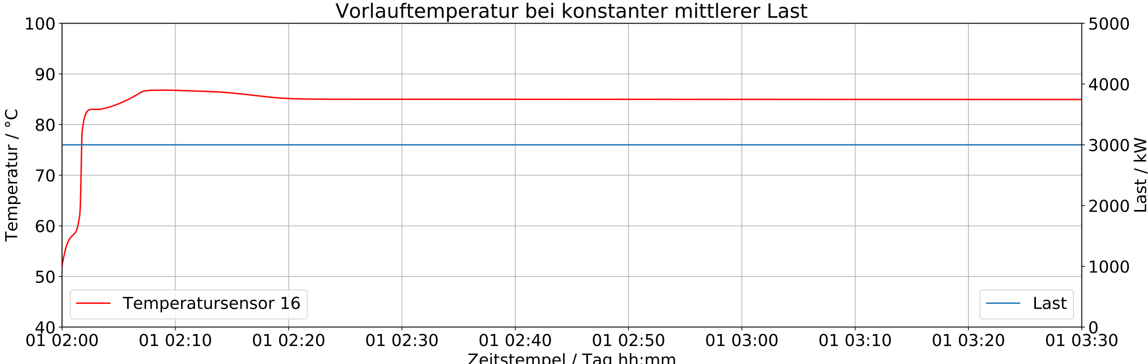

Abbildung 4.16.: Vorlauftemperatur Primärseite nach Verkleinern der Pumpe bei konstanter Last

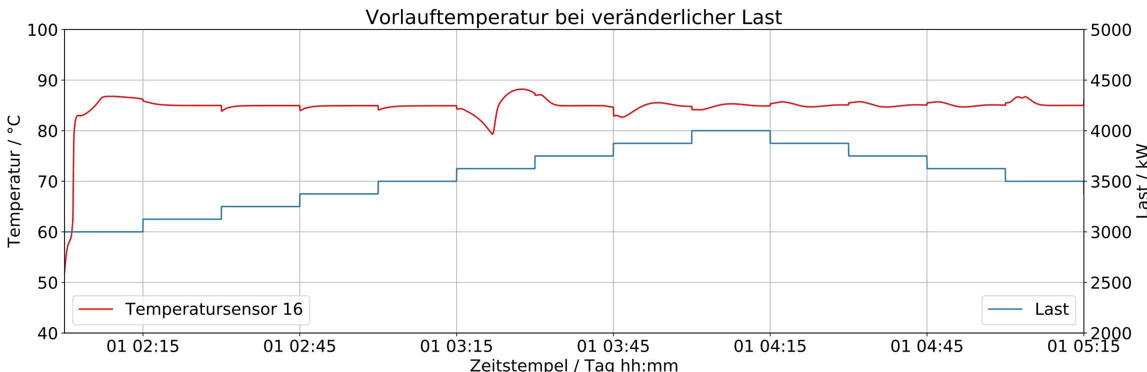

Abbildung 4.17.: Vorlauftemperatur Primärseite nach Verkleinern der Pumpe bei veränderlicher Last

Die finale Regelung der Erdgaskessel und der BHKW werden abschließend mit Hilfe von Flussdiagrammen dargestellt und beschrieben.
Das Flussdiagramm in Abbildung 4.18 zeigt die Regelung von *Boiler3*. Sie steht exemplarisch für alle Kesselregelungen. Sinkt die Temperatur an Eingang A des T-Stücks des Boiler unter eine vorgegebene Temperatur (hier 88 $^{\circ}C$) und ist zusätzlich der Kessel rechts bereits bei mindestens 95 % seiner Leistung, dann soll der Kessel mit einer Zeitverzögerung starten. Hierfür wird in einer Abfrage überprüft, ob der sogenannter *timestampWait* des Kessel der initialen Zeit (01.01.1700) entspricht. Ist dies der Fall, dann bedeutet dies, dass der Kessel nun angeschaltet werden würde. Um dies zu verzögern, wird die *timestampWait* auf die aktuelle Zeit zuzüglich 20 Minuten gestellt. Somit wird dafür gesorgt, dass die erste Abfrage solange mit *false* beantwortet wird, bis die Wartezeit abgelaufen ist. Wird die Mindesttemperatur am T-Stück überschritten oder sinkt die Leistung des vorgeschalteten Kessels, dann soll der Kessel trotzdem noch eine bestimmte Zeit weiterlaufen. Dies wird über *timestampWaitOn* realisiert. Dieser Wert wird, solange die Temperatur unterschritten wird, in jedem Zeitschritt auf die aktuelle Zeit zuzüglich 20 Minuten gesetzt. Wird die Mindesttemperatur überschritten, dann wird über eine weitere Abfrage geprüft, ob die 20 Minuten bereits abgelaufen sind. Solange dies nicht zutrifft wird der Kessel weiterhin geregelt. Ist die Zeit abgelaufen, dann wird der Kessel ausgeschaltet und und der *timestampWait* zurück auf die initiale Zeit gesetzt.
Die Regelung von BHKW1 und BHKW1 ist identisch und wird mit dem Flussbild in Abbildung 4.19 beschrieben. Zunächst gibt es erneut eine Einschaltverzögerung, die dafür sorgt, dass das BHKW erst nach 15 Minuten angeht, wenn es auf Grund von zu hohen Rücklauftemperaturen abgeschaltet wurde. Danach kommt die Abfrage, ob dies weiterhin eingehalten wird und die Rücklauftemperaturen noch unter 65 $^{\circ}C$ liegen.

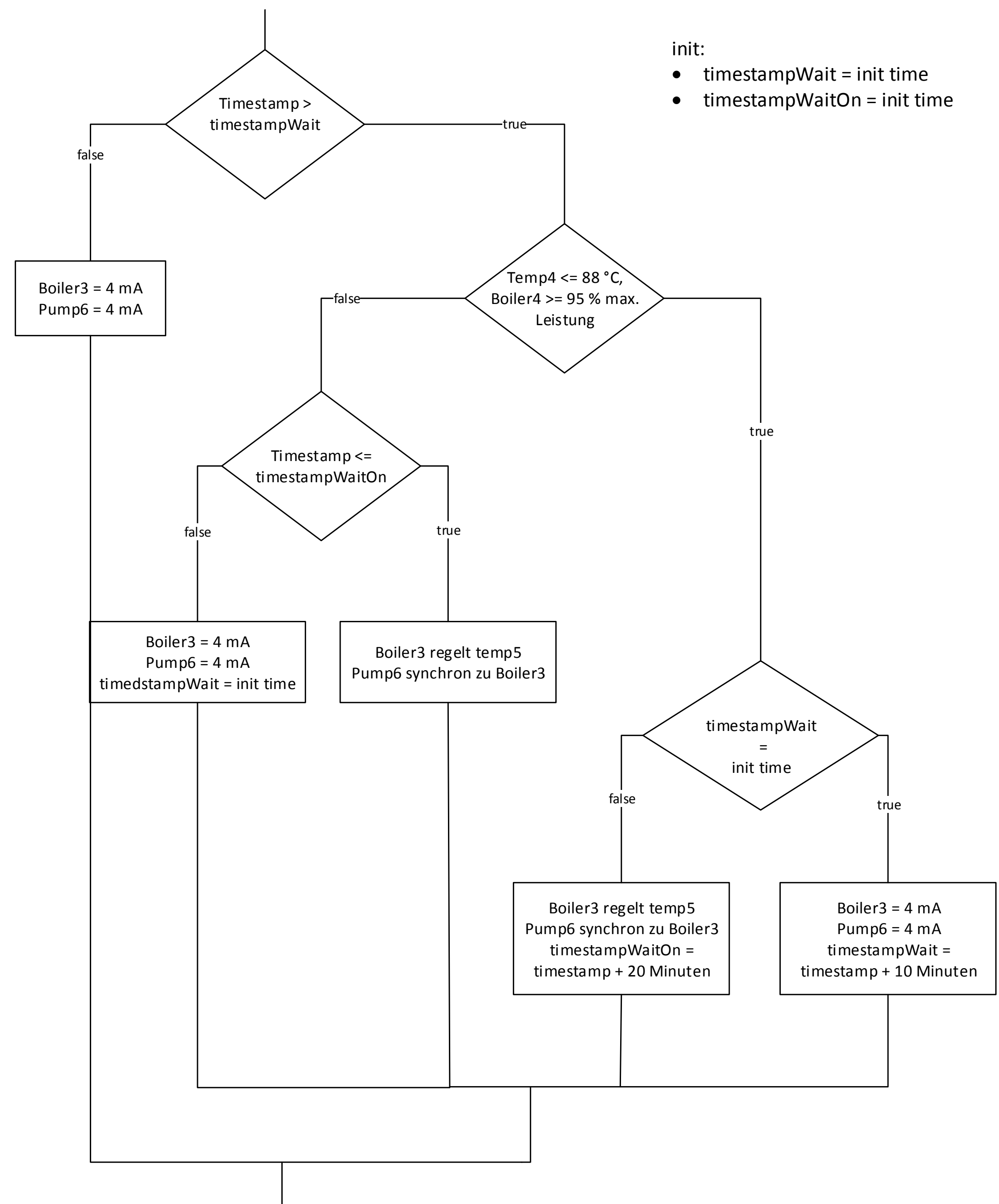

Abbildung 4.18.: Regelung eines Erdgaskessels

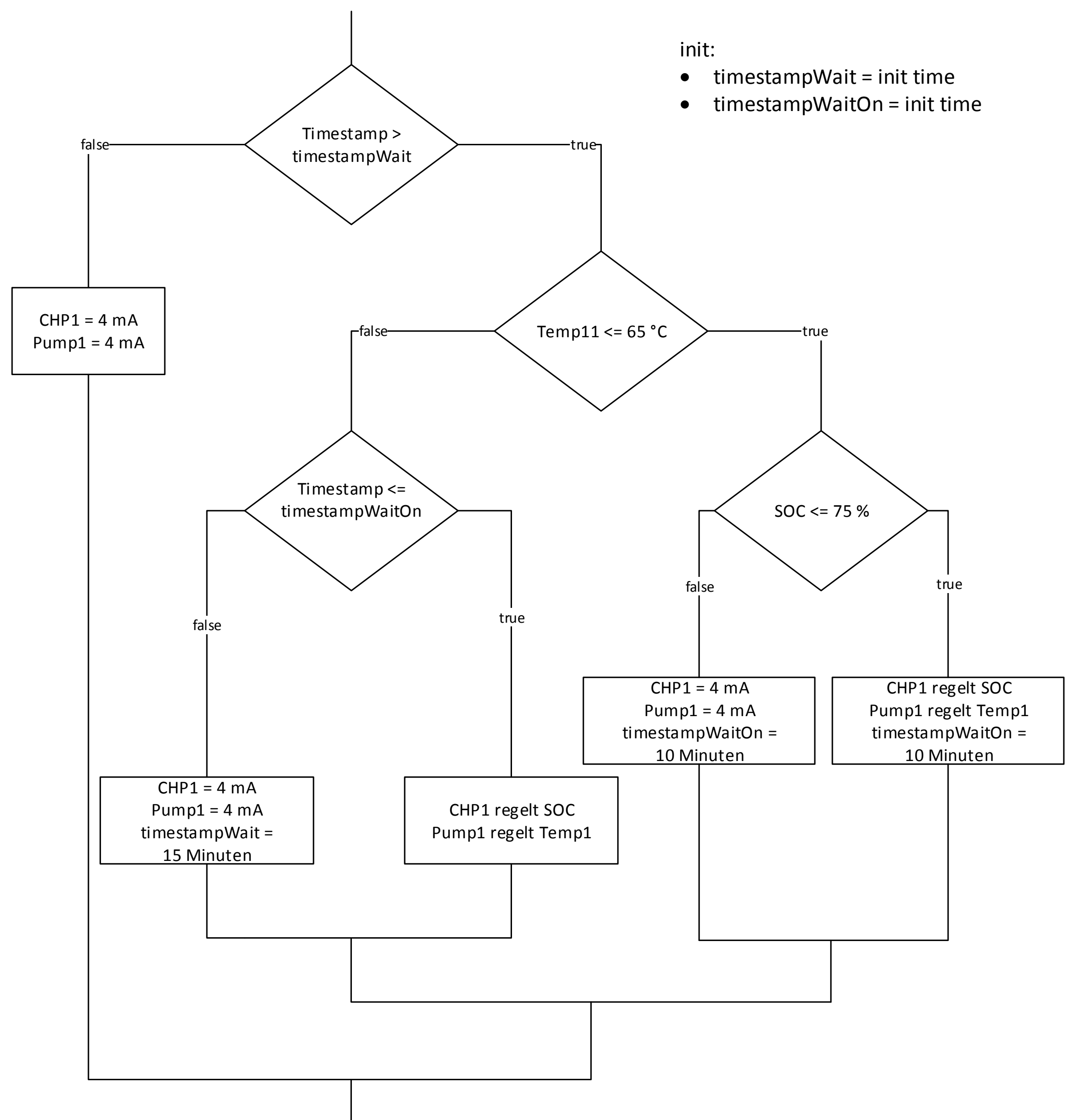

Abbildung 4.19.: Regelung BHKW1

Ist dies der Fall und ist zusätzlich der SOC kleiner oder gleich 75 %, dann regelt das BHKW den SOC und die Pumpe regelt die Vorlauftemperatur. Außerdem wird eine Mindestlaufzeit von 10 Minuten gesetzt. Liegt der SOC über 75 %, dann schaltet die Anlage ab. Wenn die Temperatur über 65 $^{\circ}C$ liegt, regelt das BHKW weiter bis die 10 Minuten

Mindestlaufzeit abgelaufen sind. Ist die Zeit abgelaufen und die Temperatur liegt weiterhin über 65 $^{\circ}C$, dann wird das BHKW abgeschaltet.
Abbildung 4.20 zeigt die Regelung des BHKW3. Die Regelung ist prinzipiell gleich aufgebaut, nur dass das BHKW über Zeitreihen aus der Datenbank geregelt wird und sich nicht an dem SOC des Speichers orientiert.

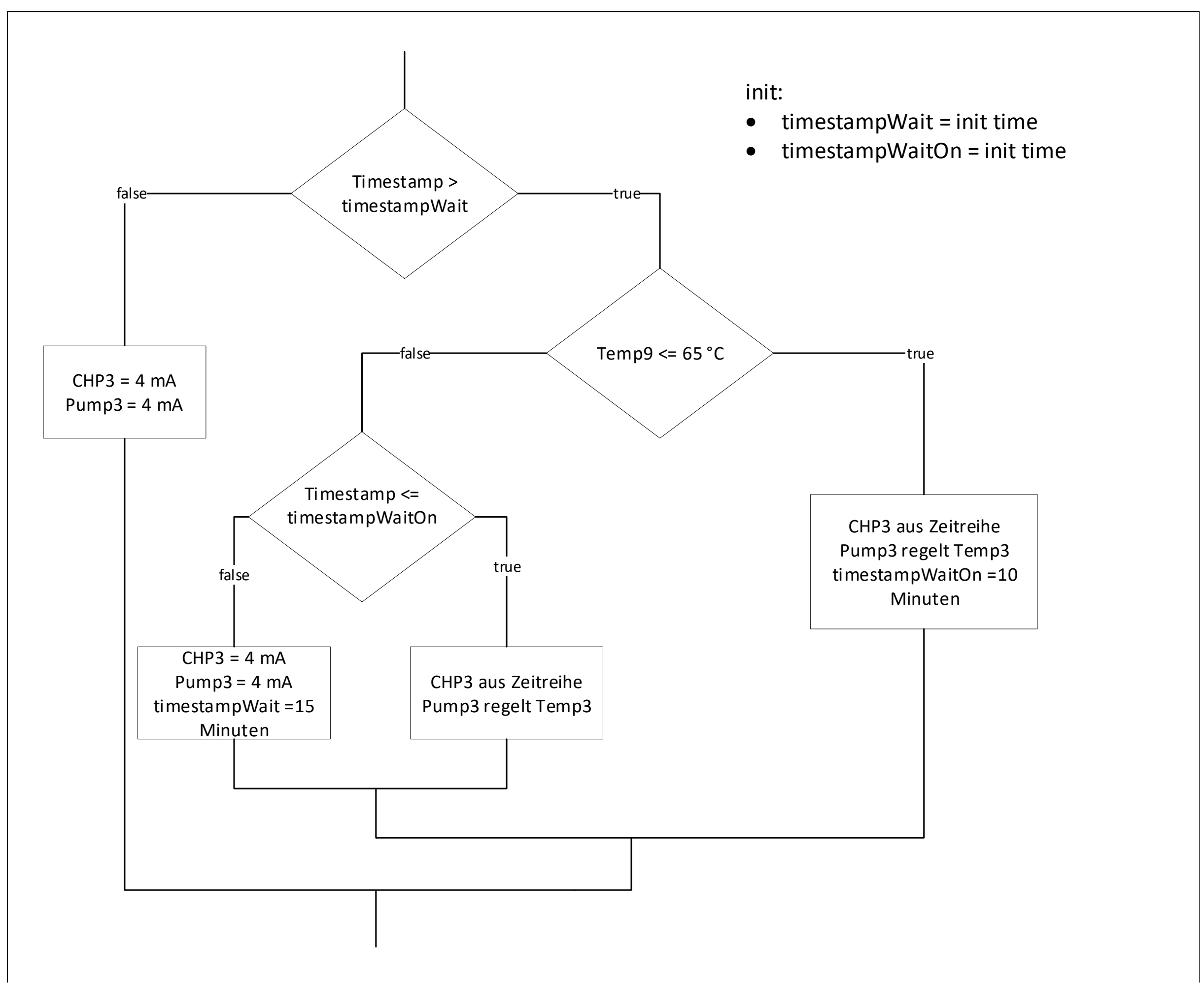

Abbildung 4.20.: Regelung BHKW3

4.2. Vereinfachtes Last- und Netzmodell

Das vereinfachte Last- und Netzmodell dient dazu, die hydraulische und kommunikationsseitige Anbindung der Verbraucher, inklusive der iWÜST, zu modellieren. Es sollen drei Lasten enthalten, die aus einem Heizkreis und einer Trinkwarmwassererwärmung bestehen und mit einem konstanten Erzeuger verbunden sind. Außerdem sollen sowohl virtuelle SPS für jede iWÜST, als auch für die W-SPS angelegt und deren Kommunikation untereinander aufgebaut werden.

4.2.1. Hydraulik und Parametrierung

Abbildung 4.21 zeigt das R&I des Subsystems. Bei dem Erzeuger handelt es sich um eine Komponente mit einer regelbaren Leistung von maximal 5000 kW. iWÜST1 und iWÜST2 haben beide eine identische Last von maximal 250 kW, sowohl für die Heizkreise als auch die TWW-Bereitung. Die dritte iWÜST hat mit maximal 500 kW die doppelte Größe. Die TWW-Bereitungen enthalten zusätzlich je einen Speicher.
Die Last ist eine Komponente mit konstanter Rücklauftemperatur bei variabler Leistung. Der Massenstrom errechnet sich in jedem Zeitschritt aus dem $\Delta\vartheta$ und der Leistung. Als Lastkurve wird bei allen Lasten eine Sinuskurve verwendet, da im Projekt noch keine separaten Lastkurven für TWW und Heizung verfügbar sind. Ein entsprechender Lastgenerator ist in Arbeit, konnte für diese Arbeit jedoch nicht berücksichtigt werden. Bei den Lasten der iWÜST 1 und 2 wurde der momentane maximale Wert der Last auf 200 kW und der minimale Wert auf 50 kW festgelegt. Bei iWÜST3 liegen die Werte zwischen 100 und 400 kW. Die Regelung findet dadurch in einem mittleren Lastbereich statt und verläuft dadurch garantiert stabil.
Der maximale Massenstrom der Last wird entsprechend der maximalen Leistung und einem angenommenen $\Delta\vartheta$ berechnet, wobei von der maximal möglichen Leistung und nicht der momentanen maximalen Last ausgegangen wird. Die Pumpe zur Regelung des Speicherfüllstandes erhält die gleiche Größe wie die Lastpumpe, sodass das Be- und Entladen des Speichers mit der gleichen Geschwindigkeit erfolgen kann. Die Größe der Speicher ist darauf aufbauend so ausgelegt, dass das Volumen innerhalb von 20 Minuten ausgetauscht werden kann. Tabelle 4.7 zeigt die notwendigen Annahmen und hydraulischen Berechnungen aller Komponenten. Die Pumpe der Last ist minimal größer gewählt als die der Speicher, da ansonsten bei voller Leistung der Pumpen innerhalb des Speichers ein Massenstrom von 0 $\frac{kg}{s}$ auftreten kann und dies zu einer Fehlermeldung und einem Abbruch der Simulation führt.

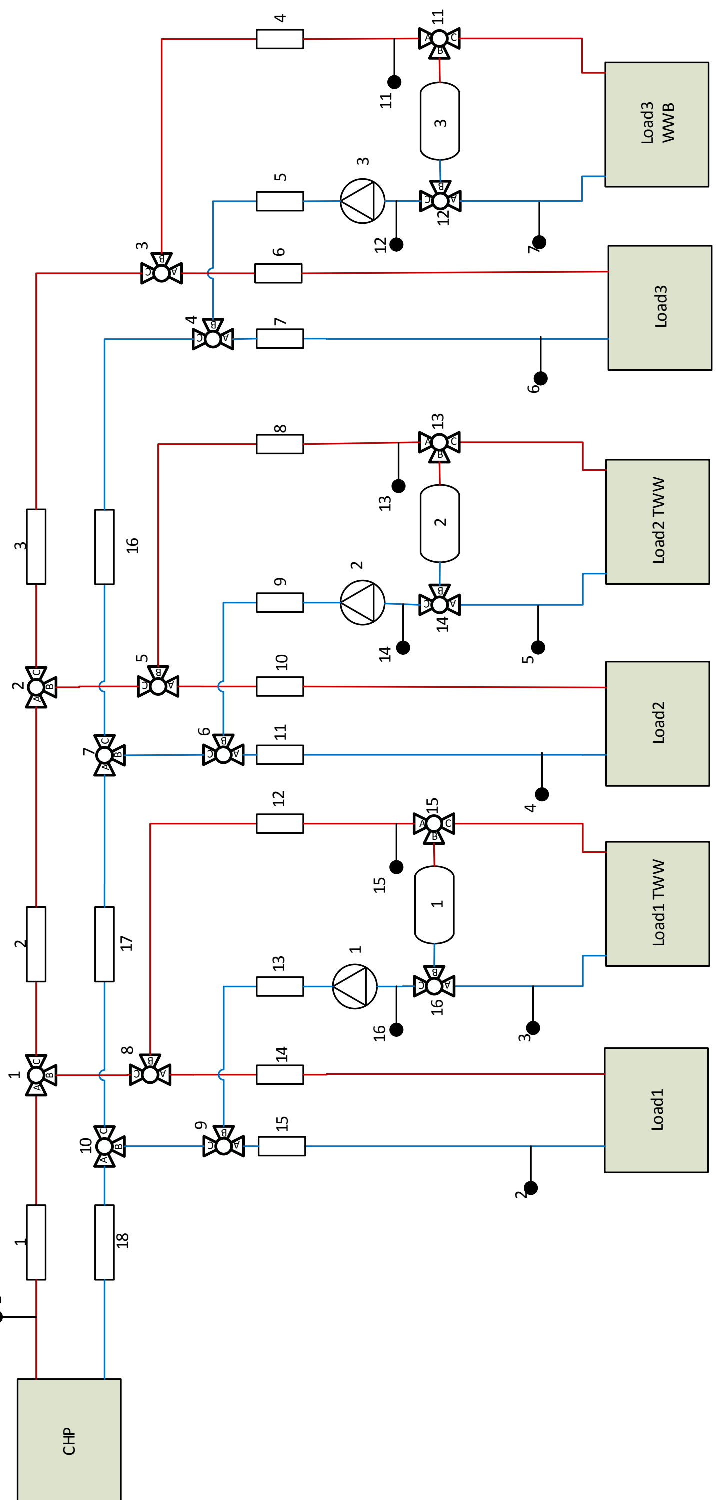

Abbildung 4.21.: R&I Schema des Last- und Netzmodells

Tabelle 4.7.: Hydraulische Berechnungen aller Komponenten

	Annahme	**Leistung/ kW**	**Parameter im Modell / m**
Pipe 8-15	$\Delta\vartheta = 32K$	500	0,04
Pipe 4-7	$\Delta\vartheta = 32K$	1000	0,04
Pipe 1&18	$\Delta\vartheta = 32K$	4000	0,08
Pipe 2&17	$\Delta\vartheta = 32K$	3000	0,07
Pipe 3&16	$\Delta\vartheta = 32K$	2000	0,04
	Annahme	**Leistung/ kW**	**Parameter im Modell / kg/s**
Pump1&2	$\Delta\vartheta = 32K$	250	2,00
Load1&2	$\Delta\vartheta = 32K$	250	2,05
Pump3	$\Delta\vartheta = 32K$	500	4,00
Load3	$\Delta\vartheta = 32K$	500	4,05
	Volumen / m^3	**Höhe / m**	**Radius / m**
Speicher3	4,42	2,2	0,8
Speicher1&2	2,24	1,7	0,65

4.2.2. Regelung

Im Lastmodell sind nur zwei Regelungsarten notwendig. Auf der einen Seite sollen die SOC der Speicher mit Hilfe der Pumpen 1-3 geregelt werden und auf der anderen Seite ist eine Regelung des Erzeugers notwendig, der für eine konstante Vorlauftemperatur an Temperatursensor 1 sorgen soll. Für die Einstellung der Vorlauftemperatur auf 90 $°C$ wird die Leistung des Erzeugers geregelt. Abbildung 4.22 zeigt das Ergebnis.

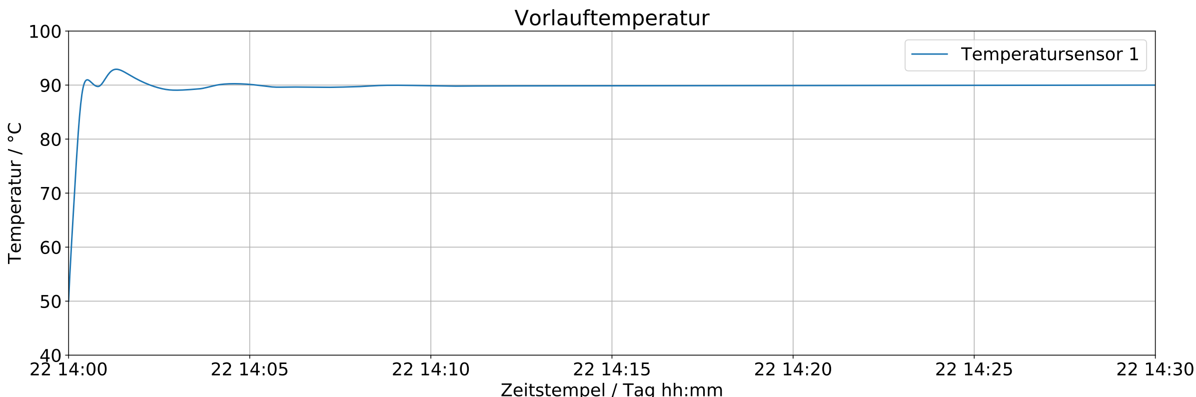

Abbildung 4.22.: Temperaturregelung am Erzeugervorlauf (Temperatursensor 1)

Für die Regelung des SOC mittels der Pumpen 1-3 muss zunächst ein T_{min} und T_{max} der Speicher bestimmt werden, um mit deren Hilfe den aktuellen SOC errechnen zu können. Die SOC Berechnung basiert auf Formel 4.1. T_{min} und T_{max} sollen für das Lastmodell möglichst genau bestimmt werden, da die Regelung des SOC der Speicher für den netzdienlichen Betrieb eine wichtige Rolle spielen. Hierfür wird eine Simulation mit anfangs vollbeladenen Speichern gestartet. Das bedeutet, dass alle Schichten des Speichers eine Temperatur von 85 $°C$ haben, da die Vorlauftemperatur am Verbraucher und somit im Speicher etwas niedriger ist als die Vorlauftemperatur direkt hinter dem Erzeuger. Die Speicher werden entladen bis der Vorlauf der Last nicht mehr über der Mindesttemperatur von 65 $°C$ gehalten werden kann.

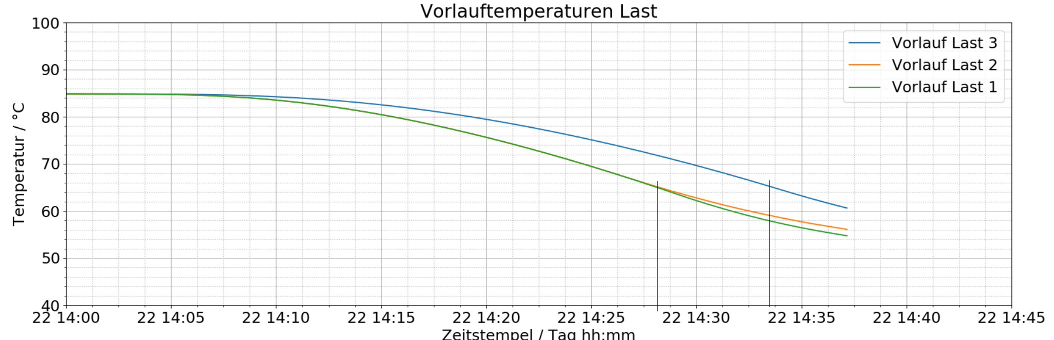

Abbildung 4.23.: Vorlauftemperaturen der Lasten

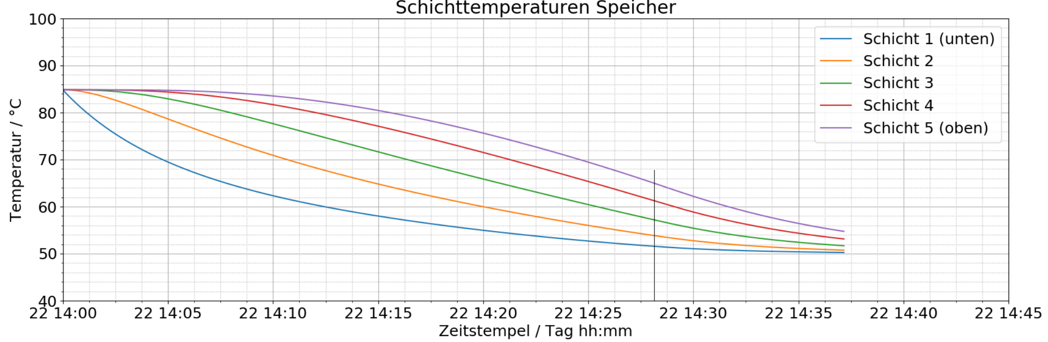

Abbildung 4.24.: Schichttemperaturverlauf Speicher 1

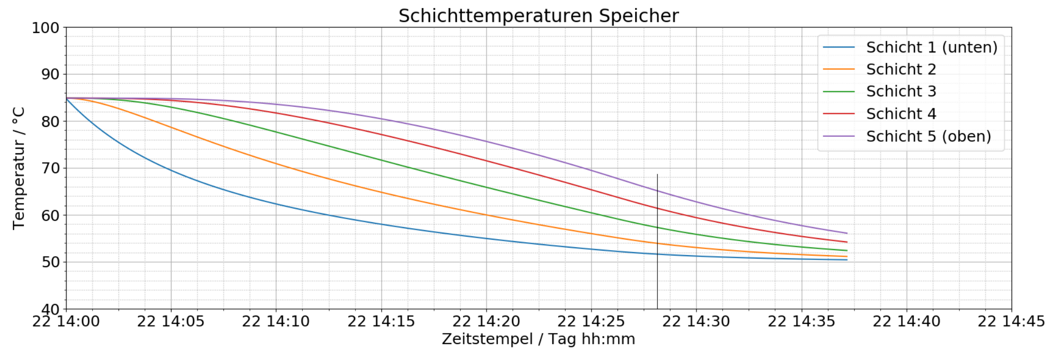

Abbildung 4.25.: Schichttemperaturverlauf Speicher 2

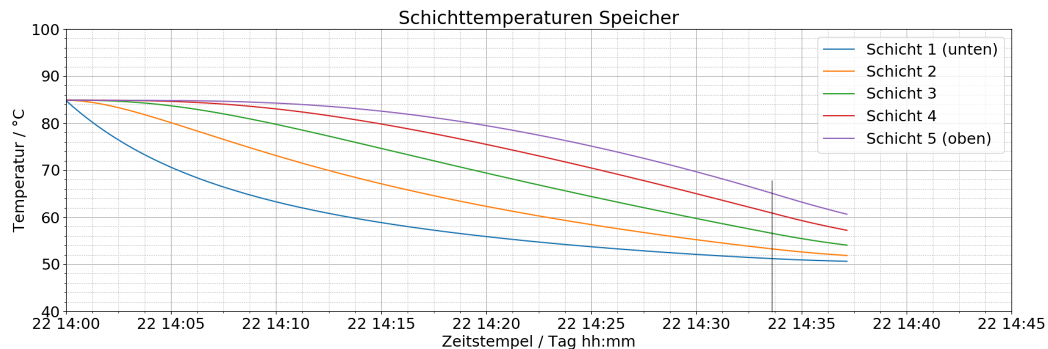

Abbildung 4.26.: Schichttemperaturverlauf Speicher 3

Abbildung 4.23 zeigt den Verlauf der Vorlauftemperaturen und die Zeitpunkte, zu denen die Temperaturen unterhalb des Mindestwertes liegen. An diesem Punkt wird die Mischtemperatur der Speicherschichten bestimmt, die folglich als T_{min} angenommen wird. Abbildung 4.24, 4.25 und 4.26 zeigen die Verläufe der Schichttemperaturen. Tabelle 4.8 stellt die Ergebnisse der gemittelten Temperaturen dar. Um T_{max} zu bestimmen, wird andersherum vorgegangen und die Simulation mit einem leeren Speicher, also allen Schichten auf der Temperatur des Rücklaufes gestartet. Der Speicher wird geladen bis die Rücklauftemperatur oberhalb des Maximalwertes von 65 $°C$ liegt. Dies ist in Abbildung 4.27 zu sehen. An diesem Punkt wird die Mischtemperatur des Speichers bestimmt und diese als T_{max} festgelegt. Abbildung 4.28, 4.29 und 4.30 zeigen die Verläufe der Schichttemperaturen.

Nachdem T_{min} und T_{max} bestimmt wurden, kann nun die Speicherregelung umgesetzt werden. Der Massenstrom der Pumpen wird variiert, um einen SOC einzustellen. Abbil-

Tabelle 4.8.: Ermittelte T_{max} und T_{min} für Speicher 1-3

	$\mathbf{T}_{\text{max}}$	$\mathbf{T}_{\text{min}}$
Speicher1	330,95 K	339,75 K
Speicher2	330,85 K	340,95 K
Speicher3	330,55 K	338,15 K

dung 4.31, 4.32 und 4.33 zeigen das Ergebnis bei einem Sollwert von 50 %. Damit sind die notwendigen Regelungen umgesetzt. Die Parameter der PID-Regler finden sich in Anhang A.1.

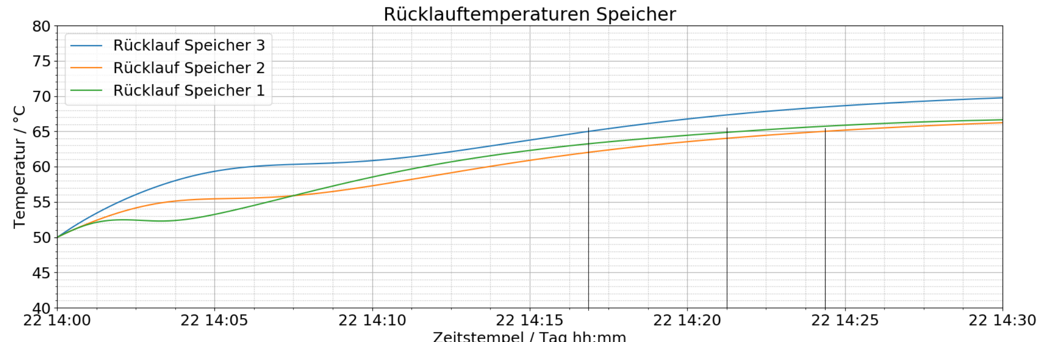

Abbildung 4.27.: Rücklauftemperaturen der Speicher

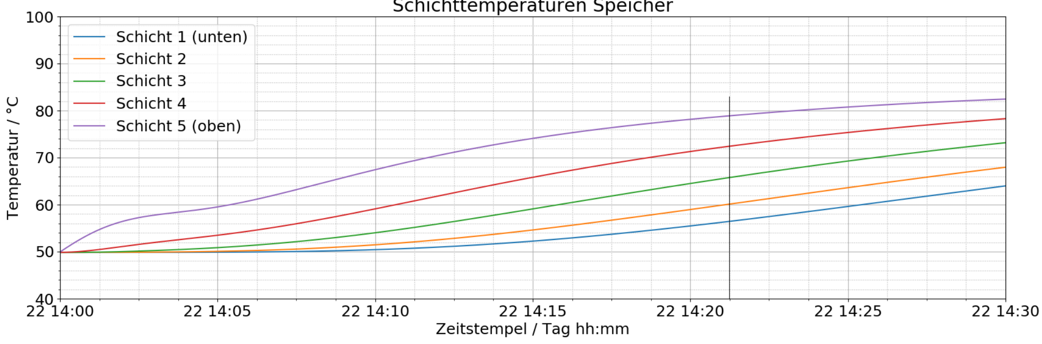

Abbildung 4.28.: Schichttemperaturverlauf Speicher 1

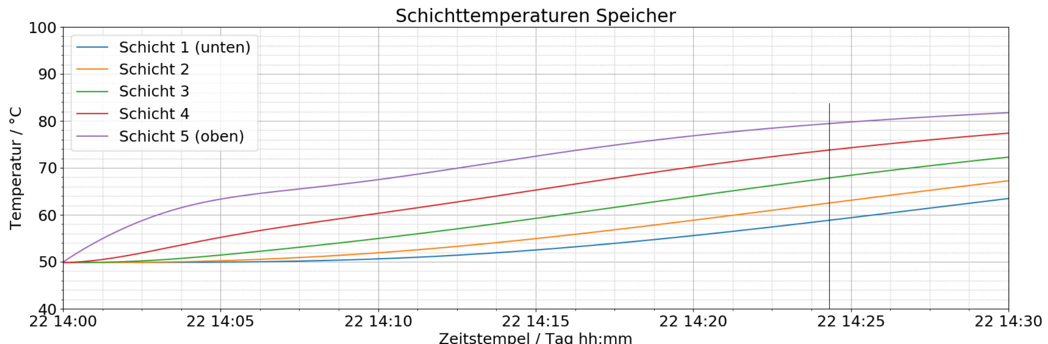

Abbildung 4.29.: Schichttemperaturverlauf Speicher 2

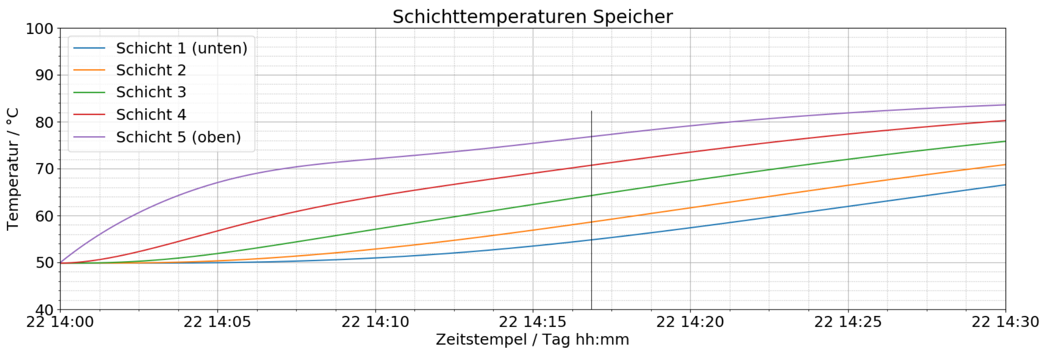

Abbildung 4.30.: Schichttemperaturverlauf Speicher 3

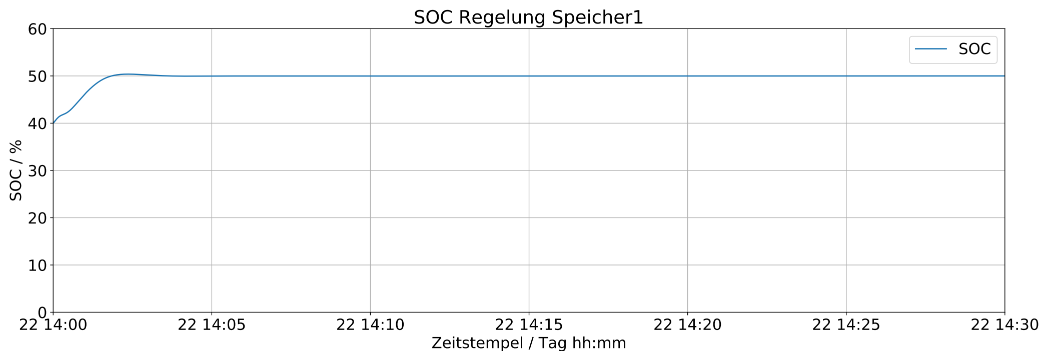

Abbildung 4.31.: SOC Regelung des Speicher 1

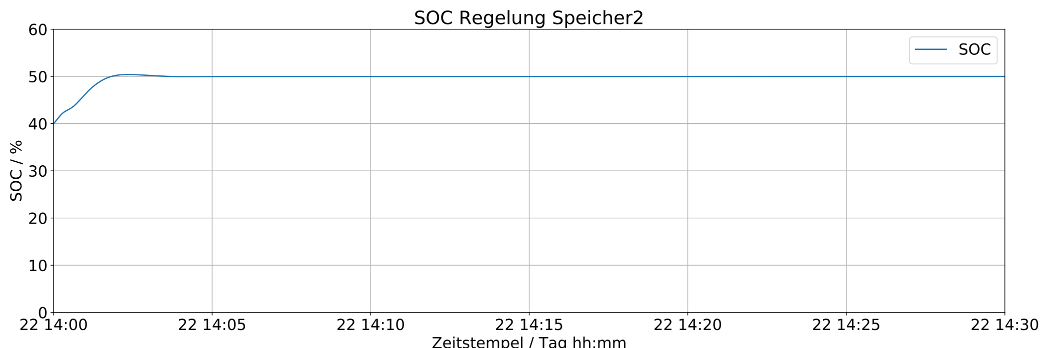

Abbildung 4.32.: SOC Regelung des Speicher 2

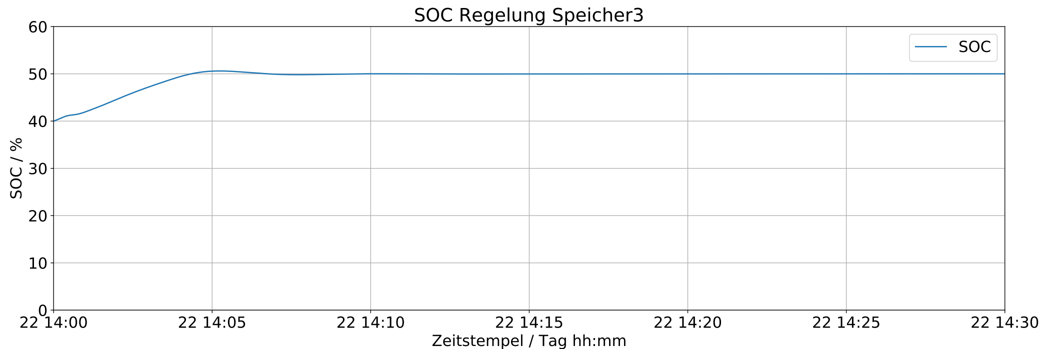

Abbildung 4.33.: SOC Regelung des Speicher 3

4.2.3. Kommunikation

Die virtuellen SPS sollen untereinander über Modbus TCP verbunden werden, wozu die Pymodbus Implementierung in Jarvis verwendet wird. Abbildung 4.34 stellt den Aufbau dar. Jede iWÜST erhält eine eigene virtuelle SPS, die mit der übergeordneten W-SPS verbunden wird. Dafür enthalten alle iWÜST einen Server sowie einen Client. Die W-SPS erhält einen Server, auf den alle iWÜST zugreifen können und für jede iWÜST einen eigenen Client. Dies ist notwendig, da die verwendeten Server zwar von mehreren Clients gleichzeitig angefragt werden können, Clients jedoch immer nur mit einem Server kommunizieren können. Die Clients der iWÜST erreichen den Server der W-SPS über den Standard Port 5020. Zur Kommunikation mit dem Server der iWÜST1 wird der

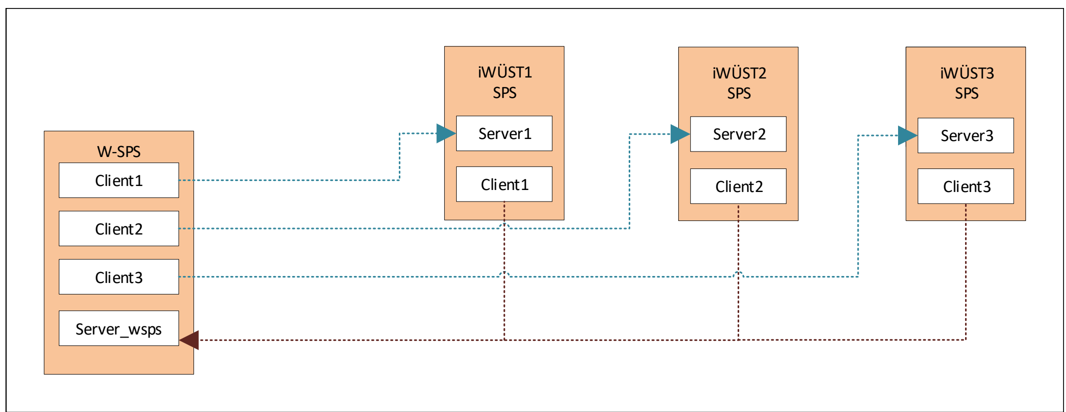

Abbildung 4.34.: Kommunikationsnetz des Verbraucher- und Lastmodells

Port 5021, für iWÜST2 Port 5022 und für iWÜST3 Port 5023 verwendet. Außerdem erhalten alle Server und Clients einen *heartbeat* mit einem "Herzschlag" alle 15 Minuten. Quelltext 4.3 zeigt beispielhaft die Instanziierung der Server und Clients der iWÜST1 und Quelltext 4.4 die der W-SPS.

Quelltext 4.3: *initialize* der W-SPS

```
1  self.temp_controller = plc_lib.PID()
2  self.drive_controller = plc_lib.AnalogDriveController()
3  self.temp_controller1 = plc_lib.PID()
4  self.drive_controller1 = plc_lib.AnalogDriveController()
5  self.server_wsps = plc_lib.PyModbusServer(port=config.MODBUS_PORT_SOFT_PLC,
6                                             heartbeat=True, heartbeat_address=10, heartbeat_dt=1, name="Server_WSPS")
7  self.client_cpu1 = plc_lib.PyModbusClient("localhost", port=config.MODBUS_PORT_SOFT_PLC + 1, name="iWuest1",
8                                             heartbeat=True, heartbeat_address=13, heartbeat_dt=15)
9  self.client_cpu2 = plc_lib.PyModbusClient("localhost", port=config.MODBUS_PORT_SOFT_PLC + 2, name="iWuest2",
10                                            heartbeat=True, heartbeat_address=13, heartbeat_dt=15)
11 self.client_cpu3 = plc_lib.PyModbusClient("localhost", port=config.MODBUS_PORT_SOFT_PLC + 3, name="iWuest3",
12                                            heartbeat=True, heartbeat_address=13, heartbeat_dt=15)
```

Quelltext 4.4: *initialize* der iWÜST1

```
1  # CPU1
2  self.temp_controller = plc_lib.PID()
3  self.drive_controller = plc_lib.AnalogDriveController()
4  self.server1 = plc_lib.PyModbusServer(port=config.MODBUS_PORT_SOFT_PLC + 1, heartbeat=True, heartbeat_address=13,
       heartbeat_dt=15, name="Server1")
5  self.client1 = plc_lib.PyModbusClient("localhost", port=config.MODBUS_PORT_SOFT_PLC, heartbeat=True,
       heartbeat_address=10, heartbeat_dt=1, name="Client1")
```

Die Simulation wird gestartet und zunächst überprüft, ob der Verbindungsaufbau zwischen den Servern und Clients funktioniert und der *heartbeat* vorhanden ist oder ob dessen Verlust gemeldet wird. Abbildung 4.35 zeigt, dass alle Verbindungen bestehen und keine Meldungen über einen Kommunikationsausfall auftreten. Damit ist die Kommunikationsbasis gelegt und im Gesamtmodell kann diese Struktur erweitert werden.

```
Starting elements of 'WSPS: first_project: Variant 1: Scenario 1 from us5517' ..
.
Connected successfully with IP localhost on port 5020.
Connected successfully with IP localhost on port 5020.
Connected successfully with IP localhost on port 5021.
Connected successfully with IP localhost on port 5022.
Connected successfully with IP localhost on port 5020.
Connected successfully with IP localhost on port 5023.
```

Abbildung 4.35.: Verbindungsaufbau zwischen den Servern und Clients zu Beginn der Simulation

4.3. Zusammenführung der Modelle

In diesem Abschnitt wird die Zusammenführung des Erzeuger-, Last- und Netzmodelles beschrieben. Hierfür ist eine hydraulische Verbindung sowie eine Anpassung der Regelung und eine kommunikationsseitige Verbindung der Modelle notwendig.
In Abbildung 4.1 ist die Verbindungen zwischen den Teilmodellen zu sehen. Das Energieerzeugermodell wird als Subsystem in das Last- und Netzmodell integriert. Die Last des Erzeugermodells wird entfernt, ebenso der Erzeuger des Lastmodells. Stattdessen werden die beiden Teilmodelle an dieser Stelle gekoppelt. Die beiden Subsysteme sind damit hydraulisch verknüpft und funktionsfähig.

Regelung

Es wird überprüft, ob die Regelung der Vorlauftemperatur hinter dem Wärmetauscher noch gute Ergebnisse liefert und die Erzeuger den Wärmebedarf der Verbraucher decken können. Außerdem werden die SOC Regelungen überprüft. Mit minimaler Anpassung der Parameter, welche in Anhang A.1 zu sehen sind, lassen sich gute Ergebnisse erzielen. Abbildung 4.36 zeigt die Vorlauftemperatur hinter dem Wärmetauscher. Abbildung 4.37, 4.38 und 4.39 zeigen die Ergebnisse der SOC Regelung der Speicher mit einem Sollwert von 50 %.

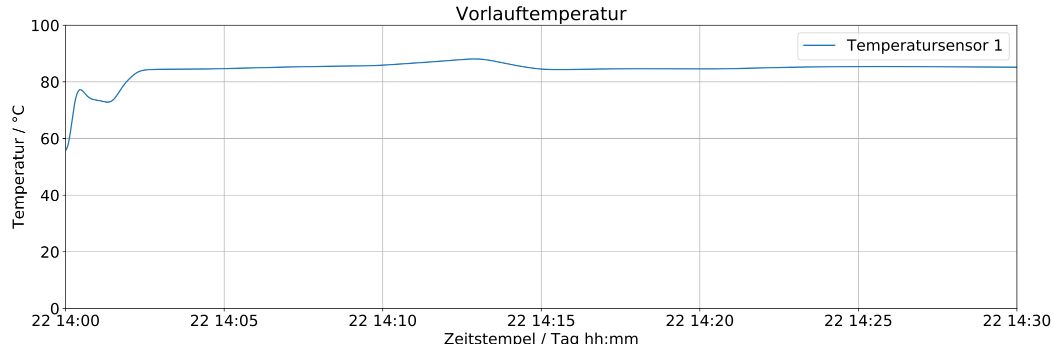

Abbildung 4.36.: Temperaturregelung am Vorlauf der Last

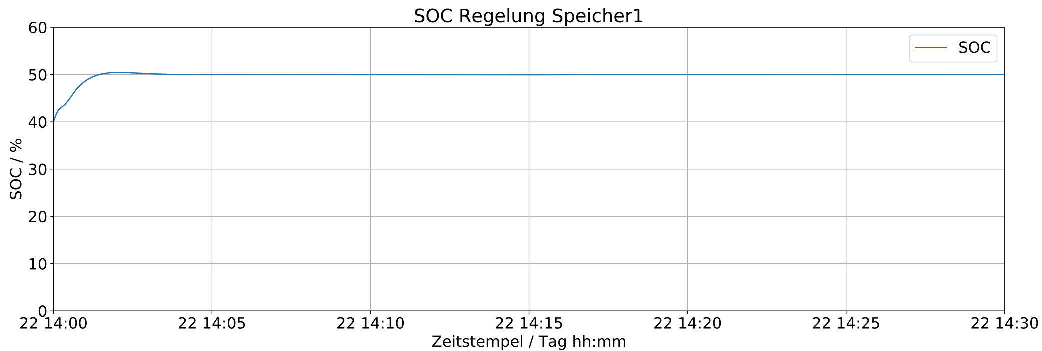

Abbildung 4.37.: SOC Regelung des Speicher 1

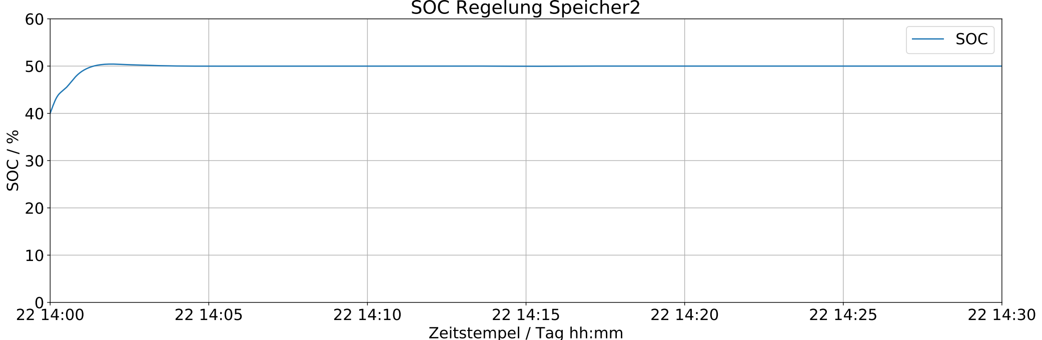

Abbildung 4.38.: SOC Regelung des Speicher 2

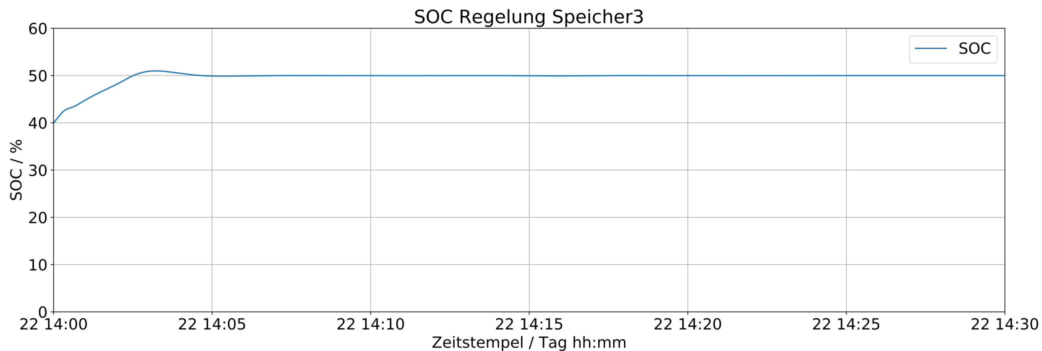

Abbildung 4.39.: SOC Regelung des Speicher 3

Kommunikation

Das Kommunikationsnetz des Last- und Netzmodells soll nun noch um die Z-SPS des Energieerzeugers ergänzt werden, um ein Netzwerk für das Gesamtmodell aufzubauen. Abbildung 4.40 zeigt den Aufbau.

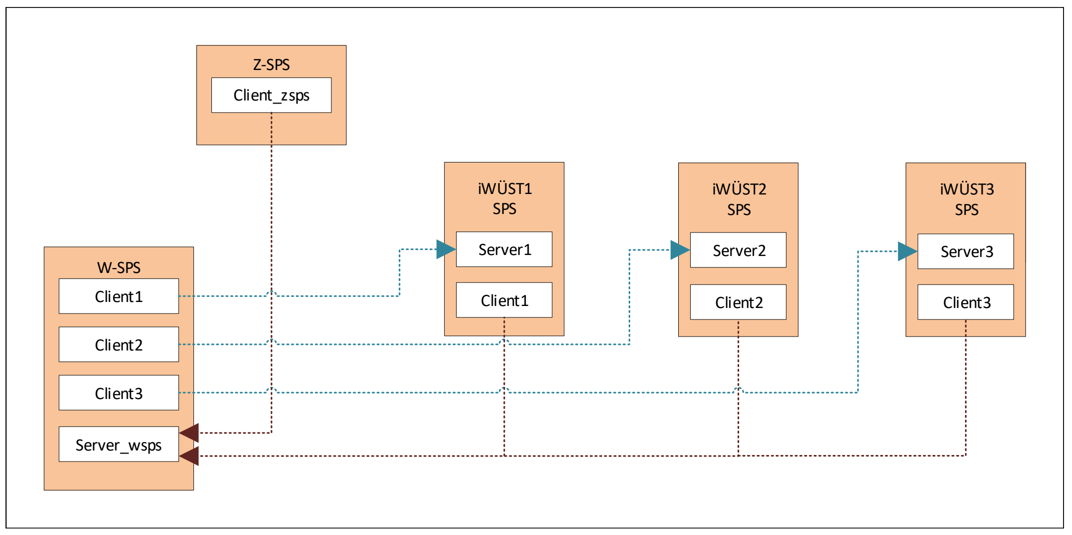

Abbildung 4.40.: Kommunikationsnetz des gesamten Modells

Die Z-SPS wird mit dem Server der W-SPS verbunden. Die Z-SPS enthält nur einen Client, da in diesem Modell nur Werte auf der W-SPS geschrieben werden sollen. Quelltext

4.5 zeigt die Instanziierung des Clients auf der Z-SPS. Damit ist das Kommunikations-netz für das Gesamtmodell aufgebaut und kann in der Umsetzung der Regelszenarien mit Datenpunkten gefüllt werden.

Quelltext 4.5: *initialize* der Z-SPS

```
1
2 self.client_zsps = plc_lib.PyModbusClient("localhost", port=config.MODBUS_PORT_SOFT_PLC, name="Z-SPS",
3                                            heartbeat=False)
```

5. Konzept der W-SPS Szenarien

In diesem Kapitel wird die Kozeptionierung der WSPS und ihrer Regelszenarien beschrieben. Die Regelszenarien sind mit den bereits bestehenden Szenarien der intelligenten Wärmeübergabestation verknüpft. Allerdings soll die W-SPS eigene Szenarien erhalten, deren Entwicklung in diesem Kapitel beschrieben werden soll.

5.1. Rahmenbedingungen

Für die Konzeptumsetzung im Modell ist es wichtig, zunächst die Rahmenbedingungen für diese festzulegen. Das Kommunikationsnetz weicht von der realen Planung ab, da in dem entwickelten Modell kein Leitsystem vorkommt. Das bedeutet, dass Signale, die von dem Leitsystem kommen sollten, entweder von der Z-SPS übertragen werden oder direkt auf der W-SPS eingegeben werden. Des Weiteren sind alle Komponenten in der Energiezentrale im Modell an die Z-SPS angeschlossen und haben keine eigene SPS, sodass nur eine SPS auf Seite der Energiezentrale existiert. In Abbildung 4.40 ist dieser Aufbau zu sehen. Für die W-SPS ergeben sich fünf relevant Szenarien:

- Inbetriebnahme (Szenario 1)
- Kommunikationsausfall (Szenario 3)
- Netzkollaps (Szenario 4)
- Funktionsprüfung (Szenario 5)
- Normalbetrieb (Szenario 10)

Die Nummerierung der Szenarien orientiert sich an der Nummerierung der iWÜST Szenarien. Auf der W-SPS sind verschiedene Möglichkeiten zur Umsetzung des Normalbetriebs möglich. Grundsätzlich befasst sich der Normalbetrieb auf der W-SPS damit, wie die Flexibilität der Speicher im Netz genutzt werden kann, um maximalen ökologischen Nutzen unter technischen und wirtschaftlichen Rahmenbedingungen zu ermöglichen. Das Szenario wird in zwei Bereiche unterteilt. Für 10a) wird der Sammelbegriff Leistungsoptimierung verwendet. Darunter kann neben dem Einhalten eines Lastprofils

auch eine Spitzenlastreduktion aus wirtschaftlichen Gründen oder eine Lastverschiebung wegen eines dynamischen Wärmepreises genannt werden. Bei 10b) geht es um technische Netzengpassbewältigung. 10b) wird in dieser Arbeit nicht bearbeitet, da für dieses Szenario komplexere Modelle inklusive einer Druckberechnung notwendig wären. 10a) wird auf die Einhaltung eines 15-Minuten-Intervall-Lastprofils[1] begrenzt. Deswegen wird im folgenden Szenario 10 mit “Einhalten eines Profils" betitelt.

5.2. Regelszenarien

Die Szenarien sind in Tabelle 5.1 zusammengefasst und werden im Folgenden erläutert.

Inbetriebnahme und Kommunikationsausfall

Das erste Szenario ist die Inbetriebnahme. Es wird beim Start der Anlage durchgeführt und die W-SPS über den Status auf der iWÜST informiert. Wenn diese vor Ort in Betrieb genommen wird, dann sendet sie die aktuelle Inbetriebnahme Testnummer an die W-SPS. Sobald die Inbetriebnahme abgeschlossen ist, bekommt die W-SPS ein Signal, ob die Inbetriebnahme erfolgreich war. Die W-SPS startet erst dann den Regelbetrieb mit einer iWÜST, wenn diese die Inbetriebnahme als erfolgreich gemeldet hat.
Um zu überprüfen, ob die Kommunikation zwischen W-SPS und iWÜST weiterhin besteht, schicken diese einen *heartbeat* hin und her. Der *heartbeat* wird vom jeweils anderen gelesen. Verändert sich innerhalb von 15 Minuten der Wert des Registers nicht, dann wird davon ausgegangen, dass die Kommunikation unterbrochen wurde. Es soll eine Nachricht auf der W-SPS erscheinen, in der steht, mit welcher SPS die Kommunikation ausgefallen ist. Wenn zu allen SPS keine Kommunikation mehr möglich ist, dann soll eine gesonderte Nachricht erscheinen, aus der hervor geht, dass jegliche Kommunikation ausgefallen ist und eventuell ein Problem mit der W-SPS besteht.

Netzkollaps

Das Netzkollaps Szenario wird ausgelöst durch ein Signal der Z-SPS. Das Szenario löst aus, wenn die Vorlauftemperatur für länger als 5 Minuten 5 K unter der Solltemperatur liegt und zusätzlich mindestens 90 % des maximalen Durchflusses vorhanden sind. Im Modell handelt es sich dabei um den Massenstrom der primärseitigen Pumpe im Vorlauf

[1] Das Intervall des Lastprofils orientiert sich an den Handelszeiträumen des elektrischen Marktes.

Tabelle 5.1.: Regelszenarien der W-SPS

Szenario	**Bedingung für Auslösung in W-SPS**	**Aktion auf W-SPS**	**Szenario in iWÜST**
1: Inbetriebnahme	Start der Anlage	Datenempfang von iWüst	Inbetriebnahme
3: Kommunikationsausfall	Register können nicht gelesen/geschrieben werden	Anzeige: Kommunikationsausfall mit SPS [SPS Nummer]. Wenn alle ausfallen, dann wird eine zusätzliche Nachricht ausgegeben.	Kommunikationsausfall
4: Netzkollaps	Signal von Z-SPS (Vorlauf über 5 Minuten 5 K unter Sollwert und Massenstrom am Wärmetauscher mindestens 90 % des Maximums)	Signalweitergabe; Freigabe, wenn Vorlauf für mindestens 10 Minuten stabil	Netzkollaps
5: Funktionsprüfung	Startsignal von W-SPS und Bereitschaft iWÜST	Prüfung wird nachts durchgeführt, nur eine Anlage zur Zeit, Einstellbar wie oft die Prüfung durchgeführt wird (ca. alle 2 Monate)	Funktionsprüfung
10a): Leistungsoptimierung:			
Einhalten eines Profils	andere Szenarien nicht aktiv SOC >= 100 %	W-SPS rechnet Energiemenge in SOC Differenz um a) alle iWÜST Speicher werden gleichmäßig genutzt b) örtlich nächster Speicher wird zuerst verwendet W-SPS gibt Signal an iWÜST weiter	Normalbetrieb Speicher überladen

des Wärmetauschers. Sobald das Szenario ausgelöst wurde, wird das Signal an alle iWÜST weitergegeben. Diese führen dann ebenfalls das Netzkollaps Szenario aus und reduzieren ihre Speicher auf den Mindestfüllstand, um das Netz zu entlasten. Sobald die Temperatur für mindestens 10 Minuten den Sollwert erreicht, wird eine Freigabe erteilt und das Szenario beendet.

Funktionsprüfung

Die Funktionsprüfung soll ausgelöst werden, wenn sowohl die iWÜST meldet, dass sie bereit ist für einen Funktionstest, also auch innerhalb der W-SPS der Funktionsprüfungsstart aktiviert wurde. Dieser wird aktiviert, wenn keine andere iWÜST zur gleichen Zeit eine Funktionsprüfung durchführt. Außerdem ist sie abhängig von der Uhrzeit und soll nur nachts in einem Intervall von etwa zwei Monaten ausgelöst werden. Die Intervalldauer soll flexibel von der Nutzer*in vorgegeben werden können. Abbildung 5.1 zeigt welche Daten für das Szenario zwischen W-SPS und iWÜST ausgetauscht werden.

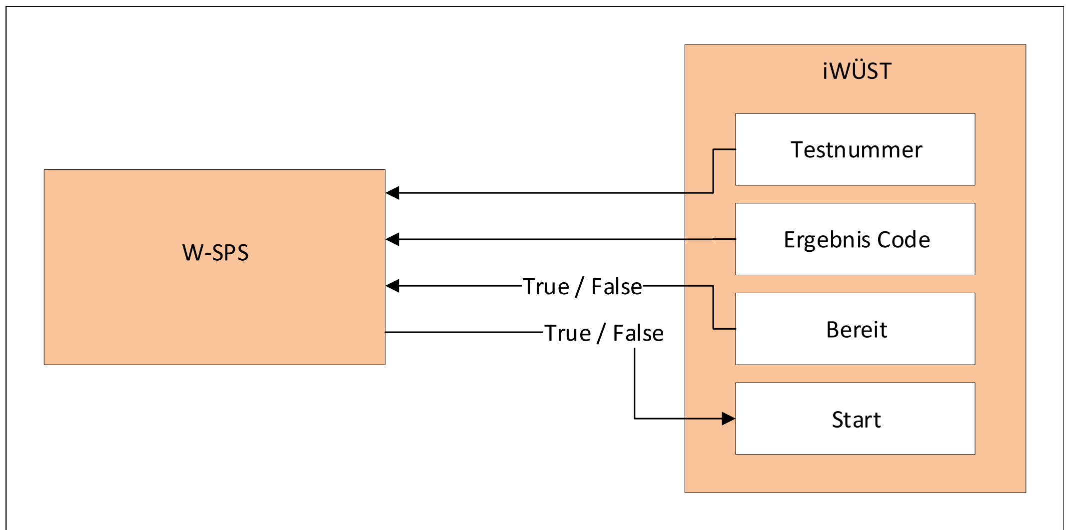

Abbildung 5.1.: Datenaustausch zwischen iWÜST und W-SPS für die Funktionsprüfung

Einhalten eines Profils

Das Szenario Einhalten eines Profils soll in zwei Varianten umgesetzt werden. In Variante a) werden alle Speicher gleichzeitig für die Flexibilität genutzt und die Energie in Abhängigkeit von der Speichergröße aufgeteilt. Für die Aufteilung der Energie ist dies die

einfachste Variante. In Variante b) wird zunächst der Speicher verwendet, der räumlich am nächsten an der Energiezentrale liegt. Im Modell ist dies iWÜST1. Je nach Energiemenge werden dann nacheinander Speicher 2 und 3 hinzugenommen. Durch Variante b) soll aufgrund der räumlichen Nähe die Flexibilität möglichst schnell zur Verfügung stehen. Der zeitliche Unterschied beim Laden der Speicher wird in Abbildung 5.2 verdeutlicht. Die Rohrleitungen zwischen dem ersten und dem zweiten Speicher haben eine Länge von 250 m.

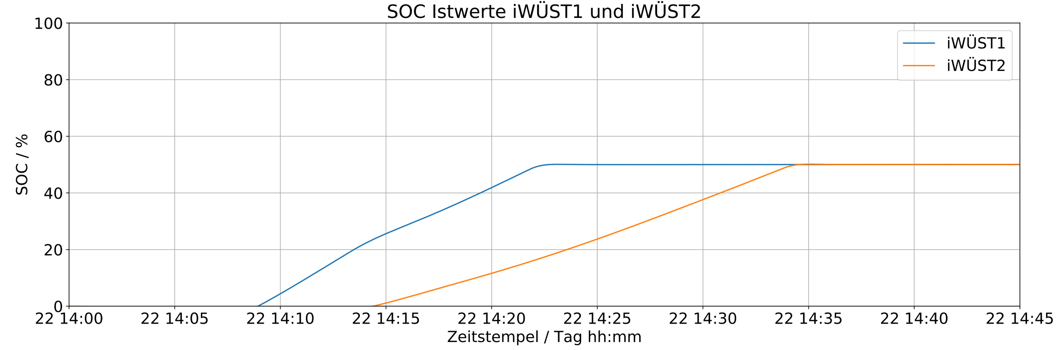

Abbildung 5.2.: Zeitlicher Unterschied beim Laden der Speicher 1 und 2

In beiden Fällen erhält die W-SPS eine Energiemenge als Viertelstundenwert. Die Speicher sollen in der Lage sein, diese Energiemenge durch Veränderung des SOC's aufzunehmen. Der SOC soll in beiden Varianten über die Viertelstunde gleichmäßig erhöht oder verringert werden. Dadurch ist die Steigung der Energiemenge, sprich die Leistung, über die Viertelstunde konstant.
Soll zusätzliche Energie im Netz gespeichert werden, wenn beispielsweise erneuerbare Erzeuger nicht abgeschaltet werden sollen, dann wird auf der iWÜST das Szenario "Speicher überladen" aktiviert. Hierfür wird der Sollwert für den Speicherfüllstand auf 100 % gesetzt und der Rücklauf erwärmt. Innerhalb der W-SPS handelt es sich um das gleiche Szenario, auch wenn die iWÜST zwischen Normalbetrieb und Speicher überladen unterscheidet.

6. Umsetzung und Simulation der W-SPS Szenarien

Dieses Kapitel beschreibt die Umsetzung des Konzeptes der Regelszenarien anhand von Ausschnitten aus der Programmierung. Die vollständige Umsetzung kann im elektronischen Anhang nachvollzogen werden. Anschließend werden die Ergebnisse dargestellt und bewertet.

6.1. Umsetzung

Für die Umsetzung der Szenarien wurde zunächst eine Datenpunktliste erstellt, damit die Register auf allen SPS festgelegt und die Kommunikation zwischen den SPS vollständig umgesetzt werden kann. Tabelle 6.1 zeigt die Auflistung der Datenpunkte. Neben der Nummer der Register und einer Beschreibung dieser, ist die Register- und Zugriffsart aufgeführt. Die Zugriffsart beschreibt, welche SPS Zugriff auf das Register hat und ob es sich um ein Lesen oder Schreiben des Registers handelt. Alle iWÜST haben den gleichen Registeraufbau. Manche Datenpunkte, wie beispielsweise der SOC Sollwert, sind auf zwei Registern aufgeteilt, damit auch die Nachkommastellen abgespeichert werden können.
In der realen Umsetzung soll die W-SPS weitere Datenpunkte der iWÜST lesen und an die Leitwarte weitergeben, beispielsweise alle Temperaturen, die gemessen werden, oder Stellungen der Ventile. Da das verwendete Modell jedoch vereinfacht ist und nicht über die hydraulischen Details einer iWÜST verfügt, können diese Werte auch nicht an die W-SPS übergeben werden.[1] Nach einer Erweiterung der Hydraulik, kann die Abfrage der Register allerdings problemlos ergänzt werden.

[1] Ein detailliertes Modell wurde zeitgleich mit dieser Arbeit im Rahmen der Masterthesis von Moritz Verbeck angefertigt und wird mit Abschluss dieser gekoppelt.

Tabelle 6.1.: Datenpunkte zur Kommunikation zwischen iWÜST, Z-SPS und W-SPS

W-SPS Register	**Datenpunkt**	**Registerart**	**Zugriffsart**
1	Inbetriebnahme Nr. iWÜST1	Holding Register	iWÜST - schreiben
2	Inbetriebnahme Nr. iWÜST2	Holding Register	iWÜST – schreiben
3	Inbetriebnahme Nr. iWÜST3	Holding Register	iWÜST – schreiben
1	Inbetriebnahme Status iWÜST1	coil	iWÜST - schreiben
2	*heartbeat*	coil	iWÜST – lesen
3	Netzkollaps	coil	Z-SPS - schreiben
4	Inbetriebnahme Status iWÜST2	coil	iWÜST - schreiben
5	Inbetriebnahme Status iWÜST3	coil	iWÜST - schreiben
iWÜST Register	**Datenpunkt**	**Registerart**	**Zugriffsart**
0	Szenario Nummer	Holding Register	W-SPS – lesen
2	Funktionstest Nummer	Holding Register	W-SPS – lesen
3	Funktionstest Ergebnis Code	Holding Register	W-SPS – lesen
6	max. Energie pro Viertelstunde	Holding Register	W-SPS – lesen
7	max. Energie pro Viertelstunde	Holding Register	W-SPS – lesen
10	SOC Sollwert	Holding Register	W-SPS - schreiben
11	SOC Sollwert	Holding Register	W-SPS - schreiben
12	SOC Istwert	Holding Register	W-SPS – lesen
13	SOC Istwert	Holding Register	W-SPS – lesen
14	max. Speicherkapazität	Holding Register	W-SPS – lesen
15	max. Speicherkapazität	Holding Register	W-SPS – lesen
1	Netzkollaps	coil	W-SPS – schreiben
2	Funktionstest Bereit	coil	W-SPS – lesen
4	Speicher überladen	coil	W-SPS – schreiben
6	Funktionstest Start	coil	W-SPS – schreiben
13	*heartbeat*	coil	W-SPS – lesen

Inbetriebnahme und Kommunikationsausfall

Das Inbetriebnahme Szenario wird in den *initialize codes* der iWÜST umgesetzt, sodass es am Anfang der Simulation einmalig ausgeführt wird. Quelltext 6.1 stellt die Umsetzung für iWÜST1 dar. Bis auf die Nummer des Registers, ist der Quelltext bei den anderen iWÜST identisch. Zuerst kann festlegt werden, ob die Inbetriebnahme erfolgreich sein soll. Anschließend wird der Wert in ein Register der W-SPS geschrieben. Wenn

die Inbetriebnahme erfolgreich sein soll, wird auch eine Testnummer auf die W-SPS geschrieben. Quelltext 6.2 zeigt das Gegenstück auf der W-SPS.

Quelltext 6.1: Umsetzung der Inbetriebnahme im *initialize code* der iWÜST

```
self.initial_operation = [True]
self.client1.write_coils(address=self.start_address_coils + 1, values=self.initial_operation)

if self.initial_operation[0]:
    self.scenario_nr = [1]
    self.server1.set_multiple_registers(address=self.start_address_floats, values=self.scenario_nr)
    self.initial_operation_nr = [15]
    self.client1.write_registers(address=self.start_address_floats + 1, values=self.initial_operation_nr)
```

Quelltext 6.2: Umsetzung der Inbetriebnahme im *initialize code* der W-SPS

```

while not self.initial_operation_1[0] and i < 100000:
    self.initial_operation_1 = self.server_wsps.get_coils(self.start_address_coils + 1)
    i = i + 1

if self.initial_operation_1[0]:
    scenario_nr_1 = self.client_cpu1.read_holding_registers(self.start_address_floats_plc1)
    print("Scenario number WUEST1: {}".format(scenario_nr_1.registers[0]))
    initial_operation_nr_1 = self.server_wsps.get_holding_registers(self.start_address_floats + 1)
    print("Initial operation test with WUEST1 - test nr.: {}".format(initial_operation_nr_1[0]))
    print("Initial operation with WUEST1 successfully terminated.")
else:
    print("Could not complete initial operation. WUEST1 is not ready for operation.")
```

Die W-SPS führt zunächst eine Schleife aus und verlässt diese erst, wenn die Inbetriebnahme erfolgreich war oder die Anzahl der Schritte *i* überschritten wird. Die Schleife ist notwendig, da der *initialize code* der W-SPS in der Simulation möglicherweise vor dem Code der iWÜST ausgeführt wird. Ist dies der Fall, dann ist der Wert in dem ausgelesenen Register *False* und die Inbetriebnahme gilt als gescheitert, woraufhin auch keine anderen Szenarien möglich sind. Die Anzahl *i* basiert auf Erfahrungswerten und ist so groß gewählt, dass die Register auf jeden Fall richtig ausgelesen werden. Dies kann als vorübergehende Lösung angesehen werden, da das Problem vermutlich durch Maßnahmen innerhalb der Simulationsplattform anders abgefangen werden kann. Wird der Inbetriebnahme Status als *True* erkannt, dann folgt eine Ausgabe mit der Testnummer und der Bestätigung, dass der Test bestanden wurde. Ob die Kommunikation bestehen bleibt, wird über den *heartbeat* überwacht.

Netzkollaps

Quelltext 6.3 zeigt die Umsetzung des Netzkollapses in der Z-SPS. Hier erfolgt die Abfrage der Vorlauftemperatur und des Massenstroms an der primärseitigen Pumpe. Die Z-SPS beschreibt daraufhin ein Register der W-SPS mit dem Netzkollaps Status. Quelltext 6.4 zeigt, dass die W-SPS ihr Register ausliest und den Status an die iWÜST weitergibt. Wird der Netzkollaps aktiviert, gibt sie außerdem eine Nachricht aus. Wie auch bei den folgenden Szenarien, ist eine weitere Bedingung für das Auslösen des Szenarios

auf der W-SPS, dass die Inbetriebnahme erfolgreich war. Quelltext 6.5 zeigt schlussendlich, dass die iWÜST im Falle des Netzkollapses den Sollwert für den Speicherfüllstand auf den vorgegebenen Minimalwert setzt.

Quelltext 6.3: Umsetzung des Netzkollapses im *main loop code* der Z-SPS

```
1
2  if temp_load < 80 + 273.15 and control_signal_pump8 >= 0.018:
3      if self.timestampWaitGridBreakdown == datetime.strptime("1700", "%Y"):
4          self.timestampWaitGridBreakdown = self.timestamp + timedelta(minutes=10)
5      elif self.timestamp > self.timestampWaitGridBreakdown:
6          self.gridBreakdown = [True]
7          self.client_zsps.write_coils(self.start_address_coils + 3, self.gridBreakdown)
8          self.timestampWaitGridBreakdownOn = self.timestamp + timedelta(minutes=5)
9  elif self.timestamp > self.timestampWaitGridBreakdownOn:
10     self.gridBreakdown = [False]
11     self.timestampWaitGridBreakdown = datetime.strptime("1700", "%Y")
12     self.client_zsps.write_coils(self.start_address_coils + 3, self.gridBreakdown)
```

Quelltext 6.4: Umsetzung des Netzkollapses im *main loop code* der W-SPS

```
1  grid_breakdown_control_system = self.server_wsps.get_coils(self.start_address_coils + 3)
2  if self.initial_operation_1[0]:
3
4      if grid_breakdown_control_system[0]:
5          self.grid_breakdown = [True]
6          self.client_cpu1.write_coils(self.start_address_coils_plc1 + 1, self.grid_breakdown)
7          scenario_nr_1 = self.client_cpu1.read_holding_registers(self.start_address_floats_plc1)
8          print("Scenario Number WUEST1: {}".format(scenario_nr_1.registers[0]))
9      else:
10         self.grid_breakdown = [False]
11         self.client_cpu1.write_coils(self.start_address_coils_plc1 + 1, self.grid_breakdown)
```

Quelltext 6.5: Umsetzung des Netzkollapses im *main loop code* der iWÜST

```
1
2      grid_breakdown = self.server1.get_coils(address=self.start_address_coils + 1)
3      if grid_breakdown[0]:
4          self.soc_set = self.soc_min
5          self.scenario_nr = [4]
6          self.server1.set_multiple_registers(address=self.start_address_floats, values=self.scenario_nr)
```

Funktionsprüfung

Die Umsetzung der Funktionsprüfung auf der W-SPS im *main loop code* wird in Quelltext 6.6 gezeigt. Zunächst wird abgefragt, ob eine festgelegte Zeit bereits abgelaufen ist. Die Zeit kann in der *initialize* eingegeben werden. Sie definiert, mit welchem zeitlichen Abstand die Funktionsprüfung durchgeführt werden soll. In der realen Umsetzung soll diese Zeit bei etwa zwei Monaten liegen. Da die Simulationen momentan allerdings in der Regel nur zwei bis drei Stunden umfassen, wird die Zeit für Testzwecke auf wenige Minuten gesetzt. Im nächsten Schritt wird überprüft, ob es sich um eine Zeit zwischen 22 Uhr Abend und 4 Uhr morgens handelt. Wenn dies der Fall ist und die iWÜST für eine Funktionsprüfung bereit ist, dann wird diese gestartet. Allerdings nur, wenn keine andere iWÜST gerade eine Funktionsprüfung durchführt. Wurde die Funktionsprüfung gestartet, liefert die W-SPS eine Aussage über die Testnummer und den Ausgang der Prüfung. Kann die Prüfung nicht gestartet werden, weil die iWÜST nicht bereit ist oder

gerade eine andere iWÜST eine Prüfung durchführt, dann meldet die iWÜST dies zurück. Auf der iWÜST werden bei Start der Prüfung zwei Register mit der Testnummer und dem Ergebnis Code beschrieben. Diese können vorher festgelegt werden.

Quelltext 6.6: Umsetzung der Funktionsprüfung im *main loop code* der iWÜST

```
1
2  # iWUEST2
3  if self.timestamp_wait_ft_2 == datetime.strptime("1700", "%Y"):
4    self.timestamp_wait_ft_2 = self.timestamp + timedelta(minutes=self.ft_dt_2)
5  elif self.timestamp > self.timestamp_wait_ft_2 and datetime.strptime("04:00:00", "%H:%M:%S") > datetime.time(self.
        timestamp) > datetime.strptime("22:00:00", "%H:%M:%S"):
6    if not self.ft_start_1[0] and not self.ft_start_3[0]:
7      self.ft_start_2 = [True]
8          self.client_cpu2.write_coils(self.start_address_coils_plc2 + 6, self.ft_start_2)
9    else:
10         self.ft_start_2 = [False]
11         self.client_cpu2.write_coils(self.start_address_coils_plc2 + 6, self.ft_start_2)
12     if ft_ready_plc2.bits[0] and self.ft_start_2[0]:
13             ft_result_code_2 = self.client_cpu2.read_holding_registers(self.start_address_floats_plc2 + 3)
14     if ft_result_code_2.registers[0] == 200:
15       scenario_nr_2 = self.client_cpu2.read_holding_registers(self.start_address_floats_plc2)
16             print("Scenario Number iWUEST2: {}".format(scenario_nr_2.registers[0]))
17             ft_test_nr_2 = self.client_cpu2.read_holding_registers(self.start_address_floats_plc2 + 2)
18             print("Functional test iWUEST2 started, Test - Nr.: {}".format(ft_test_nr_2.registers[0]))
19             print("Functional test iWUEST2 successfully terminated.")
20             self.ft_start_2 = [False]
21             self.client_cpu2.write_coils(self.start_address_coils_plc2 + 6, self.ft_start_2)
22             self.timestamp_wait_ft_2 = self.timestamp + timedelta(minutes=self.ft_dt_2)
23             self.client_cpu2.write_registers(self.start_address_floats_plc2 + 3, values=[0])
24     elif ft_result_code_2.registers[0] == 400:
25             print("Functional test iWUEST2 caused an error.")
26             self.timestamp_wait_ft_2 = self.timestamp + timedelta(minutes=self.ft_dt_2)
27             self.client_cpu2.write_registers(self.start_address_floats_plc2 + 3, values=[0])
28     else:
29              print("Waiting for functional test start iWUEST2.")
30   else:
31     print("Could not start functional test. Either iWUEST2 is not ready or another iWUEST is running a test already
           .")
```

Einhalten eines Profils

Abbildung 6.1 zeigt Schritt für Schritt, wie Variante a) umgesetzt wurde. Die gesamte Energie Q_{total} wird auf die drei Speicher aufgeteilt. Dafür wird zunächst vorausgesetzt, dass auf der W-SPS festgelegt ist, wie viele iWÜST mit welcher maximalen Speicherkapazität vorhanden sind. Es ist auch möglich, dies zu automatisieren, sodass die iWÜST bei ihrer Anmeldung ihre Speicherkapazität direkt mitteilen und die W-SPS die Aufteilung automatisch übernimmt. Die Aufteilung wurde zunächst händisch übernommen und im Anschluss eine Automatisierung ergänzt. Da Speicher 3 etwa die doppelte Größe hat, ist auch die Energiemenge doppelt so groß. Die maximale Kapazität Q_{max} des Speichers wird aus dem Register der iWÜST gelesen und verwendet, um die notwendige SOC Differenz SOC_{diff_total} für eine Viertelstunde zu berechnen. Die Berechnung von Q_{max} basiert auf Formel 6.1.

$$Q_{max} = m \cdot (T_{max} - T_{min}) \cdot c_p \tag{6.1}$$

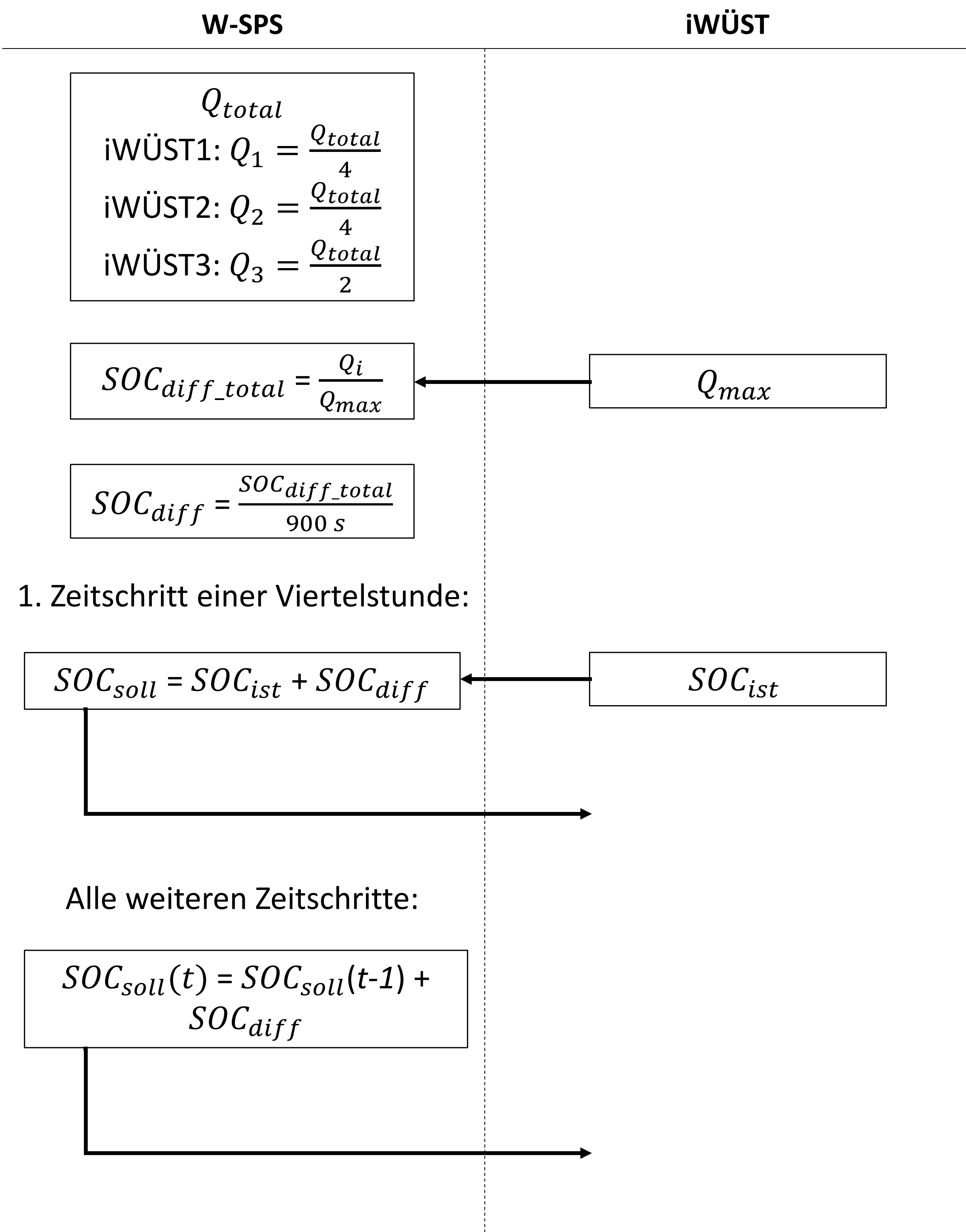

Abbildung 6.1.: Einhalten eine Profils unter gleichmäßiger Nutzung aller Speicher

Daraufhin kann auch die Änderung pro Sekunde SOC_{diff} berechnet werden. Im ersten Zeitschritt der Viertelstunde wird der Istwert des SOC von der iWÜST gelesen, der SOC Sollwert berechnet und dieser an die iWÜST übergeben. In allen weiteren Zeitschritten wird der Sollwerten basierend auf dem Wert im vorherigen Zeitschritt berechnet. Sollte eine Regelabweichung bestehen, kann somit verhindert werden, dass sich diese fortpflanzt. Dies wäre der Fall, wenn in jedem Zeitschritt der Istwert als Basis verwendet werden würde.
Für die Automatisierung der Energieaufteilung werden bei der Inbetriebnahme die maximalen Speicherkapazitäten an die W-SPS übergeben. Die W-SPS berechnet aus allen Kapazitäten der angemeldeten iWÜST einen Gesamtwert und schließlich den Anteil für jede iWÜST an dieser Gesamtmenge. Mittels des Anteils und der Flexibilitätsvorgabe für eine Viertelstunde können die Energiemengen für jede iWÜST automatisch berechnet werden.

Abbildung 6.2 zeigt das Vorgehen in Variante b). Die Berechnung der SOC Änderung ist prinzipiell die Gleiche, allerdings wird die Energiemenge anders aufgeteilt. Außerdem wird neben der maximalen Energiemenge pro Speicher noch berücksichtigt, wie viel Energie pro Viertelstunde bei maximalem Massenstrom übertragen werden kann. Formel 6.2 zeigt die Berechnung der maximalen Energie pro Viertelstunde Q_{max_900s}.

$$Q_{max_900s} = \dot{m}_{max} \cdot 0.25h \cdot (T_{max} - T_{min}) \cdot c_p \tag{6.2}$$

Zunächst wird die momentane maximal verfügbare Kapazität Q_{max_ist} der Speicher berechnet. Hierfür wird der im ersten Zeitschritt eingelesene SOC Istwert und die maximale Kapazität verwendet. Q_{max_ist} wird Q_{max_900s} gegenübergestellt und das Minimum verwendet. Ist die momentan verfügbare Kapazität größer, aber die Pumpleistung kann diese Energiemenge nicht übertragen, dann muss die maximale Energie pro Viertelstunde verwendet werden. Ist die maximal übertragbare Energie pro Viertelstunde größer als die verfügbare Kapazität, dann muss die Kapazität verwendet werden. Ist die ermittelte Energie größer als die geforderte Energie, dann wird lediglich Speicher 1 benötigt und der SOC Sollwert wie bereits beschrieben berechnet. Ist die geforderte Energie jedoch größer, dann erfolgt als Nächstes eine Abfrage, ob die Kapazität von Speicher 2 für die übrige Energie ausreicht. Ist dies der Fall, dann wird der SOC von Speicher 1 maximal erhöht und der SOC von Speicher 2 entsprechend der restlichen Energiemenge angepasst. Ist auch Speicher 2 nicht ausreichend, dann werden Speicher 1 und 2 maximal beladen und die restliche Energie von Speicher 3 übernommen. Sind bereits alle Speicher bei 100 %, dann wird das Szenario Speicherüberladen in der iWÜST ausgelöst und der Rücklauf aufgewärmt. Dies gilt auch für Version a).

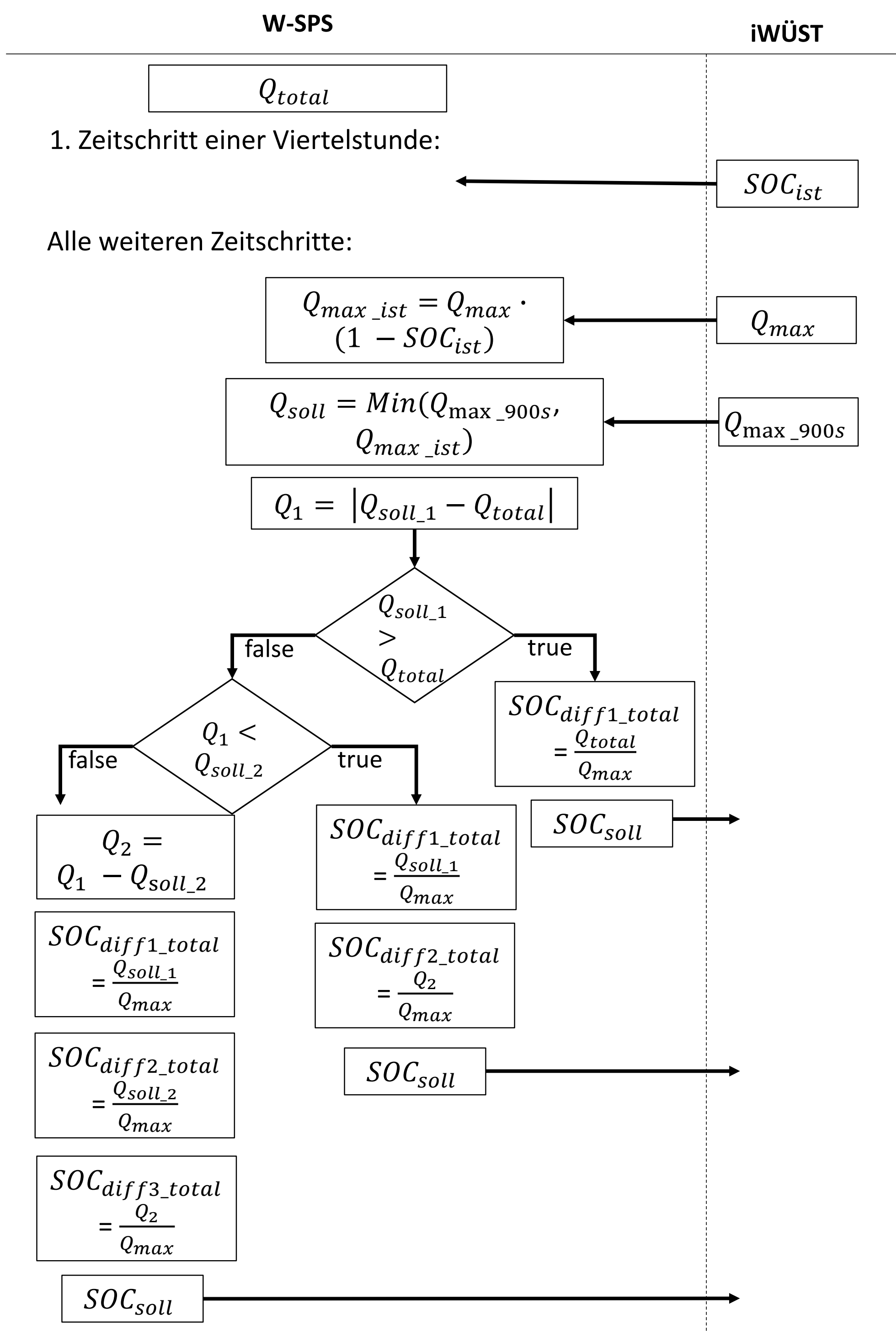

Abbildung 6.2.: Einhalten eines Profils unter ungleichmäßiger Nutzung der Speicher

6.2. Ergebnisse und Ergebnisbewertung

Dieses Unterkapitel stellt die Ergebnisse der Szenariensimulation dar. Zur Bewertung der Ergebnisse werden sie mit den ursprünglichen Anforderungen an die Regelszenarien verglichen.

Inbetriebnahme und Kommunikationsausfall

Das Inbetriebnahme Szenario soll zu Beginn überprüfen, ob alle Teile der iWÜST erfolgreich in Betrieb genommen wurden. Ist dies der Fall, wird dies an die W-SPS zurück gemeldet und der Regelbetrieb kann starten. Die Ausgabe in Abbildung 6.3 zeigt, dass das Inbetriebnahme Szenario erfolgreich ist. Der Inbetriebnahme Status der iWÜST wurde auf *True* gesetzt, was diese zusammen mit der Testnummer erfolgreich zurückmelden. Im weiteren Verlauf wird kein Kommunikationsfehler gemeldet, da alle *heartbeats* auf die gleiche Frequenz gesetzt wurden. Wird die Frequenz bei einem Teilnehmer verändert, dann wird die Meldung über den Verlust angezeigt. Die Szenarien Inbetriebnahme und Kommunikationsausfall konnten somit erfolgreich umgesetzt werden.

```
Terminal
Instantiated Component "temp_sensor9" (Temperature_Sensor) from 'WSPS: coupled_p
roject: Variant 1: Scenario 1 from us5517'.

Starting elements of 'WSPS: coupled_project: Variant 1: Scenario 1 from us5517'
...
Connected successfully with IP localhost on port 5020.
Connected successfully with IP localhost on port 5020.
Connected successfully with IP localhost on port 5021.
Connected successfully with IP localhost on port 5022.
Connected successfully with IP localhost on port 5020.
Connected successfully with IP localhost on port 5023.
Connected successfully with IP localhost on port 5020.
Scenario number WÜST1: 1
Initial operation test with WÜST1 - test nr.: 15
Initial operation with WÜST1 successfully terminated.
Scenario number WÜST2: 1
Initial operation test with WÜST2 - test nr.: 15
Initial operation with WÜST2 successfully terminated.
Scenario number WÜST3: 1
Initial operation test with WÜST3 - test nr.: 15
Initial operation with WÜST3 successfully terminated.
... done.
```

Abbildung 6.3.: Ausgabe des Inbetriebnahme Szenarios

Netzkollaps

Im Fall eines Netzkollapses soll gezeigt werden, dass das Netz durch Leeren der Speicher entlastet werden kann. Abbildung 6.4 zeigt den Verlauf der Vorlauftemperatur und Abbildung 6.5 den Massenstrom an der primärseitigen Pumpe. Abbildung 6.6 zeigt daraufhin den Wechsel vom Szenario Normalbetrieb Nummer 10 in das Szenario Netzkollaps Nummer 4. Der Wechsel geschieht, nachdem die Vorlauftemperatur mindestens 5 Minuten unterhalb von 80 $°C$ liegt und die Pumpe mindestens 46,8 $\frac{kg}{s}$, also 90 % des maximalen Massenstroms, erreicht hat.
Im Modell dauert es durch die Trägheit der Pumpenregelung relativ lange bis das Szenario ausgelöst wird. Da die Zeitdauer mit 5 Minuten recht klein gewählt ist, ist die Dauer bis zum Auslösen trotzdem akzeptabel. Der Zeitraum bis zum Auslösen des Szenarios sollte also auch in Abhängigkeit der Pumpenregelung gewählt werden. In Abbildung 6.7 ist zu sehen, dass die iWÜST durch das Leeren ihrer Speicher keine Last mehr darstellen.
In den Szenarien der iWÜST ist ein Mindestfüllstand von 10 % vorgesehen. Bei der Art der Bestimmung von T_{min} und T_{max} in dieser Arbeit ist dies jedoch nicht zwangsläufig notwendig, da T_{min} so gewählt wurde, dass bei einer Ladung von 0 % noch die minimale Vorlauftemperatur gehalten werden kann. Der Mindestfüllstand wurde dementsprechend auf 0 % gesetzt, wodurch eine Entlastung des Netzes solange garantiert wird, bis die minimale Vorlauftemperatur an der iWÜST nicht mehr gehalten werden kann.
Die Last steigt ab einem gewissen Punkt wieder an. Der Grund dafür ist in Abbildung 6.8 zu sehen. Der Speicherfüllstand wird immer weiter reduziert und liegt irgendwann unterhalb des Mindestfüllstandes, weswegen die Speicherpumpen in Abbildung 6.9 wieder hochfahren, um den Sollwert zu halten.

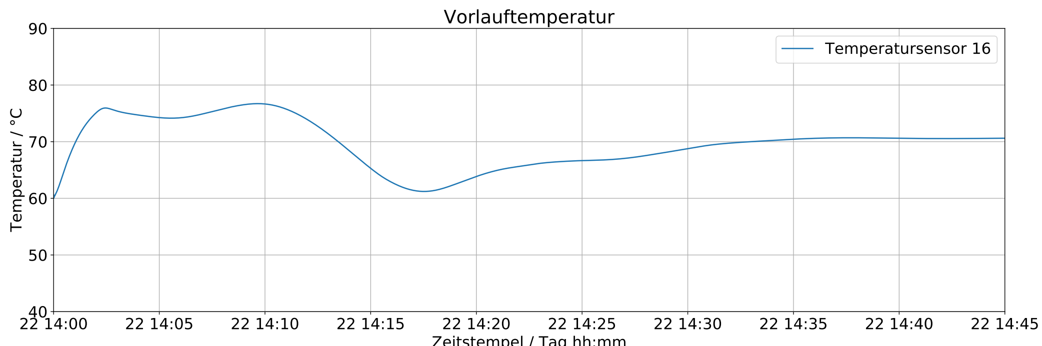

Abbildung 6.4.: Verlauf der Vorlauftemperatur im Szenario Netzkollaps

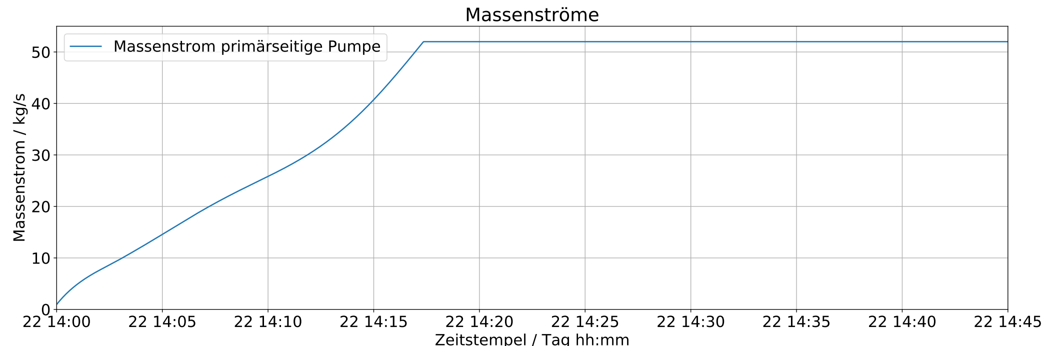

Abbildung 6.5.: Verlauf des Massenstroms der primärseitigen Pumpe im Szenario Netzkollaps

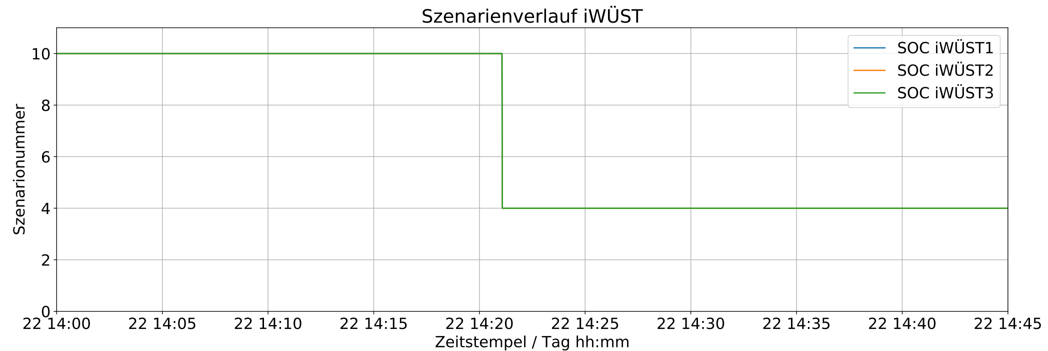

Abbildung 6.6.: Wechsel zwischen Normalbetrieb und Netzkollaps auf iWÜST3

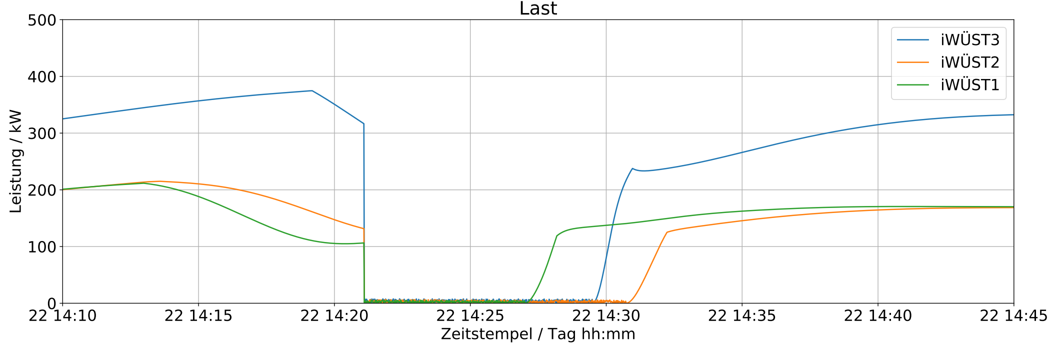

Abbildung 6.7.: Lastkurven der TWW an iWÜST 1-3 aus der Sicht des Netzes

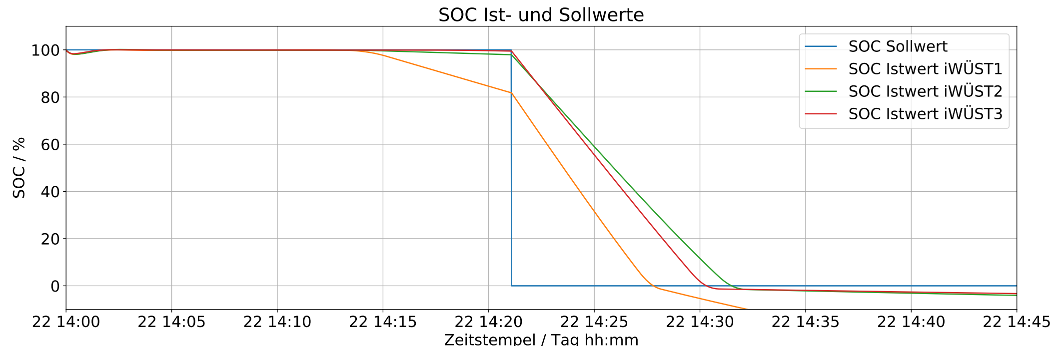

Abbildung 6.8.: SOC Verlauf der iWÜST während eines Netzkollapses

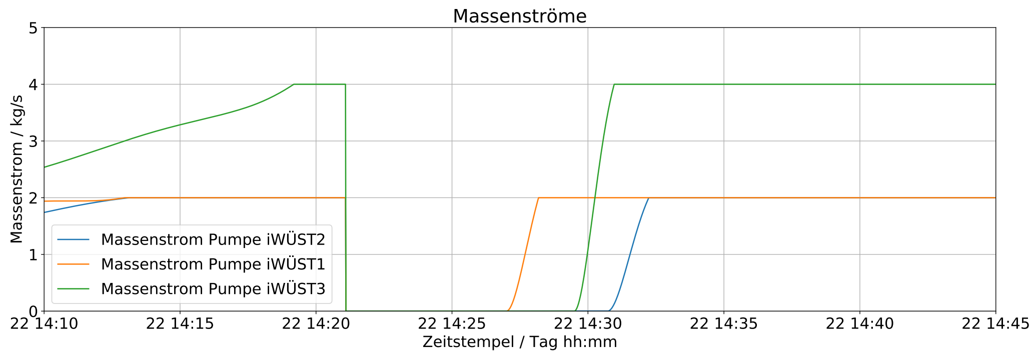

Abbildung 6.9.: Massenströme der Pumpen 1-3 während des Netzkollapses

Dieses Regelverhalten ist grundsätzlich erwünscht, jedoch mit der aktuellen Vorlauftemperatur nicht möglich, da diese bereits zu niedrig ist. Die Ergebnisse zeigen, dass das Szenario richtig funktioniert und das Netz dadurch entlastet werden kann, solange die Temperatur hoch genug ist, um einen Mindestfüllstand zu halten. Die Höhe des Mindestfüllstandes sollte generell betrachtet werden, da sie von den maximalen und minimalen Temperaturen abhängt, die zur SOC Berechnung verwendet werden. Auf der iWÜST könnte noch ergänzt werden, dass sie das Szenario erst wieder verlassen, wenn ihre Vorlauftemperatur ebenfalls über einen Zeitraum stabil ist.

Funktionsprüfung

Die Funktionsprüfung soll in einem bestimmten Intervall durchgeführt werden, außerdem nur Nachts und wenn nicht bereits andere iWÜST Funktionsprüfungen durchführen. Die Ausgaben während einer Funktionsprüfung sind in Abbildung 6.10 zu sehen. Zunächst wurde der Test ohne Eingrenzung der Uhrzeit durchgeführt. Alle iWÜST haben die gleiche zeitliche Vorgabe bekommen, wann die Prüfung durchgeführt werden soll. Bei iWÜST1 und iWÜST2 wurde der Ergebniscode eingegeben, der für einen erfolgreichen Test steht. Auf iWÜST3 dagegen wurde der Ergebnis Code für eine fehlerhafte Prüfung hinterlegt. In Abbildung 6.10 ist zu sehen, dass iWÜST1 mit der Funktionsprüfung startet und die anderen iWÜST währenddessen warten. iWÜST 1 und 2 bestehen den Test und melden dies zurück, wohingegen iWÜST3 einen Fehler meldet. Das Szenario verhält sich also gemäß den Anforderungen. Als Nächstes wird die Abhängigkeit von der Uhrzeit hinzugefügt, woraufhin die Ergebnisse in der Nacht die Gleichen sind, während die Prüfung tagsüber nicht gestartet wird. Das Szenario wurde somit erfolgreich umgesetzt.

```
Terminal
2018-07-22 14:00:29
2018-07-22 14:00:30
Waiting for functional test start iWÜSt1.
Could not start functional test. Either iWÜST2 is not ready or another iWÜST is
running a test already.
Could not start functional test. Either iWÜST3 is not ready or another iWÜST is
running a test already.
2018-07-22 14:00:31
Scenario Number iWÜST1: 5
Functional test iWÜST1 started, Test - Nr.: 1
Functional test iWÜST1 successfully terminated.
Waiting for functional test start iWÜST2.
Could not start functional test. Either iWÜST3 is not ready or another iWÜST is
running a test already.
2018-07-22 14:00:32
Scenario Number iWÜST2: 5
Functional test iWÜST2 started, Test - Nr.: 2
Functional test iWÜST2 successfully terminated.
Waiting for functional test start iWÜST3.
2018-07-22 14:00:33
Functional test iWÜST3 caused an error.
2018-07-22 14:00:34
2018-07-22 14:00:35
2018-07-22 14:00:36
```

Abbildung 6.10.: Ausgabe der iWÜST bei einer Funktionsprüfung

Einhalten eines Profils

Zuerst werden die Ergebnisse der Variante a) betrachtet, in der alle Speicher gleichmäßig be- und entladen werden. Bei dem aufgezeichneten Zeitraum handelt es sich um drei Viertelstunden. In der ersten Viertelstunde soll keine zusätzliche Energie aufgenommen werden und lediglich der SOC bei 50 % gehalten werden. In der zweiten Viertelstunde soll eine Energie von insgesamt 30 kWh aufgenommen werden. Das entspricht je 7,5 kWh für iWÜST1 und iWÜST2 und 15 kWh für iWÜST3. In der dritten Viertelstunde soll eine Energie von insgesamt 40 kWh an das Netz abgegeben werden, also 10 kWh von iWÜST1 und iWÜST2 und 20 kWh von iWÜST3. Abbildung 6.11 zeigt, dass der Istwert der SOC's der iWÜST1 nahezu identisch ist mit dem Sollwert. Dementsprechend zeigt sich auch in Abbildung 6.12, dass die Energieveränderung in Speicher 1 der geforderten Menge entspricht. Die rote Linie markiert den Sollwert für die zweite Viertelstunde und die grüne Linie den Sollwert für die dritte Viertelstunde. Beide Werte werden zum Ende der jeweiligen Viertelstunde erreicht. Das Gleiche gilt für iWÜST 1 und 2.

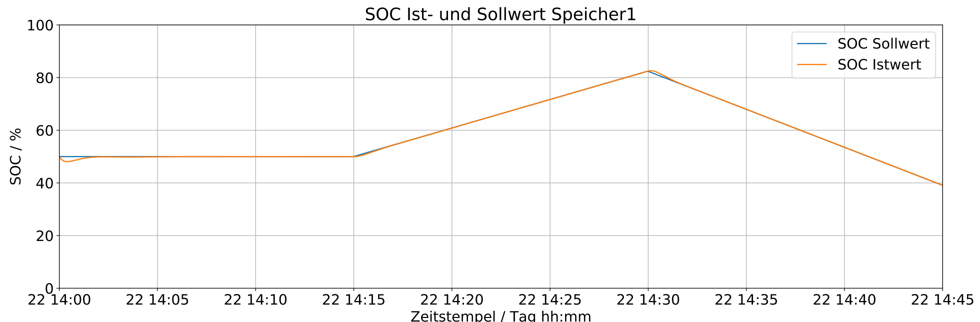

Abbildung 6.11.: SOC Verlauf auf iWÜST1

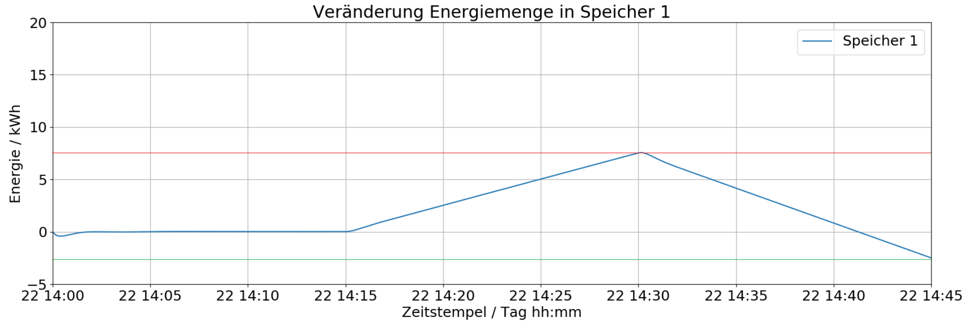

Abbildung 6.12.: Änderung der enthaltenen Energie im Speicher der iWÜST1

Allerdings fällt bei der Durchführung des Szenarios auf, dass der Temperaturverlauf der Schichtung des Speichers im Modell nicht den Erwartungen für einen realen Verlauf entspricht. Abbildung 6.13 zeigt die Schichtung im Modell und Abbildung 6.14 einen erwartbaren Verlauf. Die Anzahl der Schichten wird daraufhin von 5 auf 10 Schichten erhöht. Der neue Verlauf ist in Abbildung 6.15 zu sehen und entspricht teilweise mehr dem erwarteten Verlauf, ist allerdings nicht vollständig zufriedenstellend. Für die erfolgreiche Umsetzung des Szenarios ist dies nicht relevant, sollte jedoch in späteren Arbeiten Berücksichtigung finden.

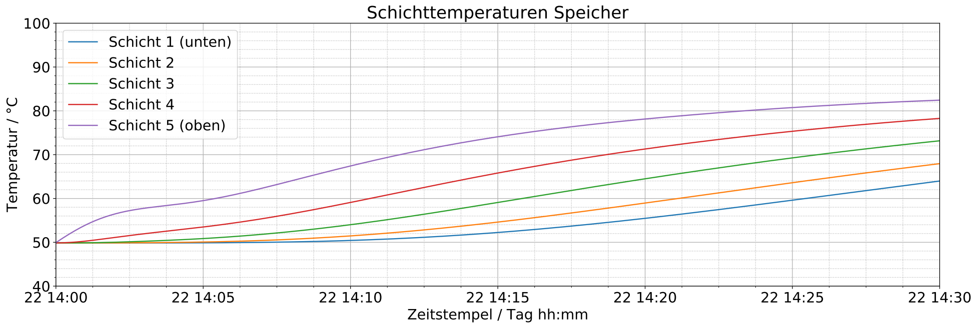

Abbildung 6.13.: Temperaturverlauf der 5 Schichten im Modell

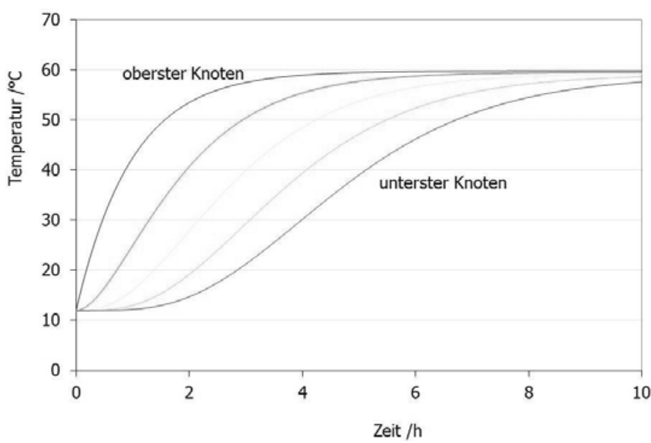

Abbildung 6.14.: Erwarteter Temperaturverlauf [7, S.123]

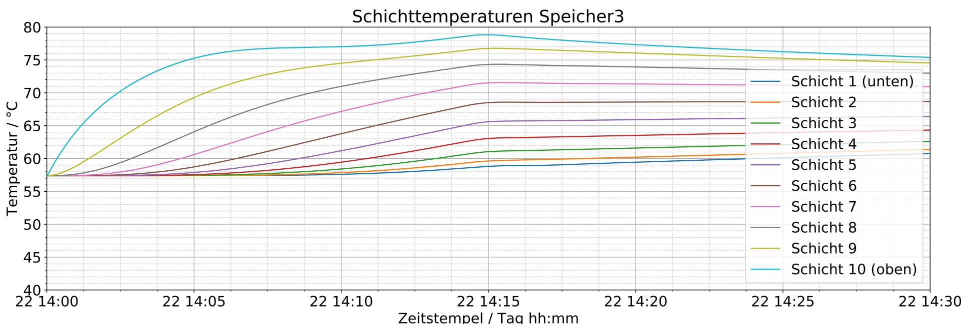

Abbildung 6.15.: Temperaturverlauf der 10 Schichten im Modell

Die zuvor händisch vorgenommene Verteilung der Energiemenge wurde automatisiert, sodass die maximale Speicherkapazität bei der Inbetriebnahme der iWÜST automatisch übergeben wird. Die W-SPS berechnet daraus die Gesamtkapazität aller angemeldeten iWÜST und die Anteile der Speicher. Abbildung 6.16 zeigt die Ausgabe dieser Automatisierung.

```
Terminal
Starting elements of 'WSPS: coupled_project: Variant 1: Scenario 1 from us5517' ...
Connected successfully with IP localhost on port 5020.
Connected successfully with IP localhost on port 5020.
Connected successfully with IP localhost on port 5020.
Connected successfully with IP localhost on port 5021.
Connected successfully with IP localhost on port 5022.
Connected successfully with IP localhost on port 5023.
Connected successfully with IP localhost on port 5020.
Scenario number WÜST1: 1
Initial operation test with WÜST1 - test nr.: 15
Initial operation with WÜST1 successfully terminated.
Scenario number WÜST2: 1
Initial operation test with WÜST2 - test nr.: 15
Initial operation with WÜST2 successfully terminated.
Scenario number WÜST3: 1
Initial operation test with WÜST3 - test nr.: 15
Initial operation with WÜST3 successfully terminated.
The share of the storage of iWÜST1 is 0.2606348262452423.
The share of the storage of iWÜST2 is 0.2991376814325978.
The share of the storage of iWÜST3 is 0.44022749232215996.
3 WUEST have registered at W-SPS. The total capacity of the storages is 88.64213562011719
kWh.
... done.
```

Abbildung 6.16.: Ausgabe nach Ergänzung der automatisierten Kapazitätsberechnung

Die Umsetzung der Variante a) war somit erfolgreich und durch eine SOC Erhöhung nach dem verwendeten Prinzip kann eine vorgegebene Flexibilität für eine Viertelstunde bereitgestellt werden.

Ist die aufzunehmende Energie so groß, dass alle Speicher maximal beladen werden, dann wird das Szenario *Speicher überladen* auf der iWÜST ausgelöst. Dies sieht vor, dass der Sollwert für den Speicherfüllstand auf 100 % gesetzt, der Speicher maximal beladen und der Rücklauf dadurch ebenfalls erhöht wird. Abbildung 6.17 zeigt beispielhaft den Verlauf der SOC'S auf den iWÜST1 und Abbildung 6.18 den Verlauf der Rücklauftemperaturen. Die Rücklauftemperaturen steigen langsam an.

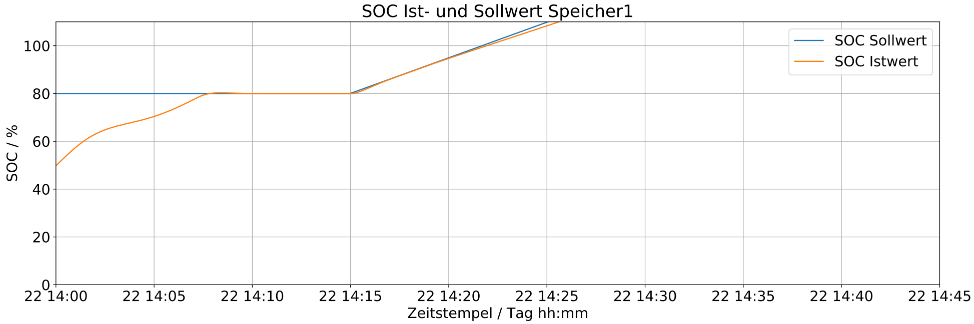

Abbildung 6.17.: Verlauf des SOC auf der iWÜST1

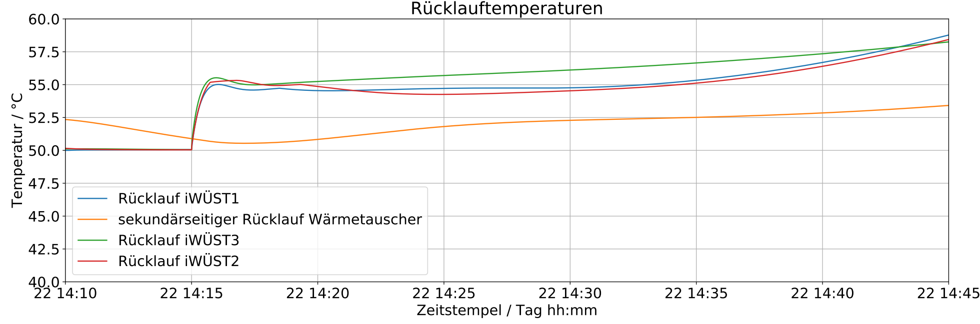

Abbildung 6.18.: Verlauf der Rücklauftemperaturen während der Speicherüberladung

Hier zeigt sich erneut, dass die verwendete Art der Bestimmung von T_{min} und T_{max} bei den Szenarien berücksichtigt werden muss. T_{max} ist so ausgelegt, dass der Rücklauf bei 100 % immer unter einer Temperatur von 65 $°C$ liegt. Der Rücklauf erhöht sich dadurch nur minimal, wenn der SOC auf 100 % geregelt wird. Um den Rücklauf weiter zu erwärmen, müssen SOC Wert von über 100 % eingestellt werden.

In Variante b) werden zwei Viertelstunden betrachtet. In der Ersten soll keine Veränderungen stattfinden und der SOC bei 50 % liegen. In der Zweiten sollen zuerst 15 kWh

und im Anschluss 30 kWh aufgenommen werden. Abbildung 6.19 und 6.20 zeigen, dass SOC Soll- und Istwerte der iWÜST übereinstimmen. Dies führt zu der Energieerhöhung in iWÜST1 und iWÜST2 (Abbildung 6.19 und 6.20). Die Energiemenge in Speicher 3 ist unverändert. In Summe ergibt sich eine Energieerhöhung um 15 kWh, was dem Sollwert entspricht.
Für eine Vorgabe von 30 kWh sind die SOC's im Anhang in Abbildung A.2, A.3 und A.4 zu sehen und die Energiemengen im Anhang in A.5, A.6 und A.7. Der Sollwert für den SOC der iWÜST2 wird jedoch zum Ende der Viertelstunde nicht mehr erreicht. Dies führt dazu, dass die Energiemenge in Summe etwas niedriger ist und nur 29,8 kWh anstatt der vorgegebenen 30 kWh beträgt.

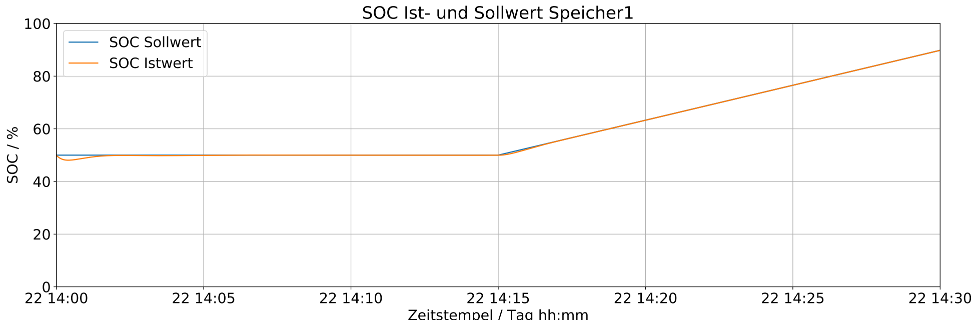

Abbildung 6.19.: SOC Verlauf der iWÜST1

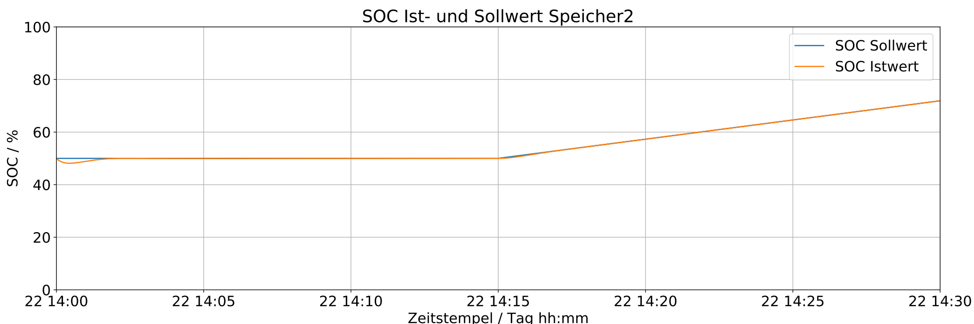

Abbildung 6.20.: SOC Verlauf der iWÜST2

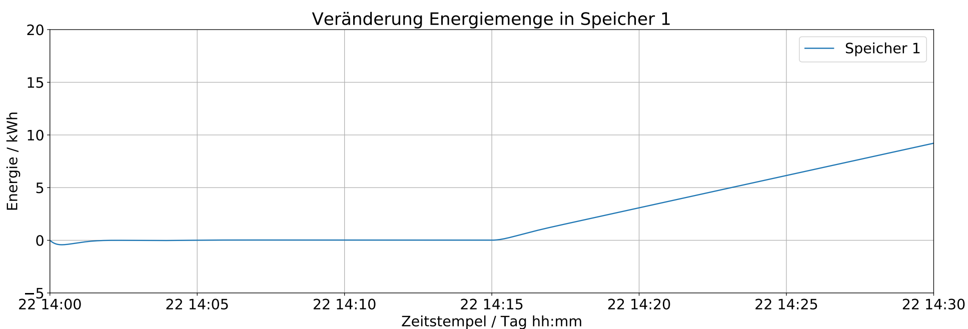

Abbildung 6.21.: Änderung der Energie im Speicher der iWÜST1

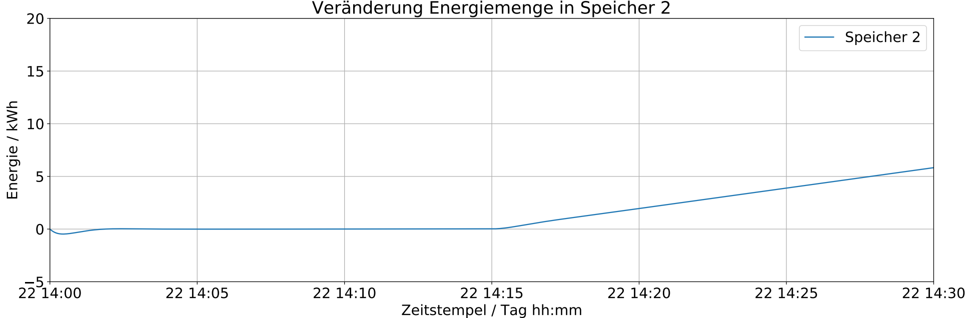

Abbildung 6.22.: Änderung der Energie im Speicher der iWÜST2

Alles in allem konnte Variante b) ebenfalls erfolgreich umgesetzt werden, da die Abweichungen bei 30 kWh relativ gering sind. Es lässt sich jedoch durch die bisherigen Testreihen nicht eindeutig sagen, welchen Mehrwert die Berücksichtigung der räumlichen Entfernung der iWÜST für das Szenario birgt. Es konnte gezeigt werden, dass Temperaturerhöhungen im Netz bei iWÜST1 früher ankommen und dementsprechend Speicher 1 früher damit beladen werden kann (Abbildung 5.2). Um die Vor- und Nachteile der Szenarien besser vergleichen zu können, müssten noch detailliertere Untersuchungen stattfinden bzw. zunächst Anwendungsfälle definiert werden in denen die Unterschiede der Umsetzung zu tragen kommen. Variante b) ist außerdem aufwändiger, da die maximale Erhöhung für die Speicher pro Viertelstunde möglichst genau berechnet werden muss, damit die Kapazität ausgenutzt und die restliche Energie auf den nächsten Speicher übertragen werden kann. Die theoretisch möglichen Werte für die SOC Differenz weichen teilweise von den realen Werten ab. Dies liegt vermutlich daran, dass die Bestimmung von T_{max} und T_{min} immer mit Ungenauigkeiten behaftet ist und die maximale Energiemenge von diesen Größen abhängig ist.

7. Zusammenfassung und Fazit

Ziel der Arbeit war es, einen wärmenetzdienlichen Betrieb intelligenter Wärmeübergabestationen mittels einer übergeordneten Steuerung zu testen. Hierfür sollten erste Regelszenarien definiert und diese anschließend in einem Modell, bestehend aus zwei Subsystemen inklusive Wärmeübergabestationen und Speichern, umgesetzt und simuliert werden. Außerdem sollte der Fokus auf der Kommunikation zwischen den SPS liegen und diese mittels einer Implementierung der Python Bibliothek *Pymodbus* in der Simulationsumgebung ermöglicht werden.

Zusammenfassend lässt sich sagen, dass die Regelungsszenarien für einen wärmenetzdienlichen Betrieb erfolgreich konzeptioniert und simuliert werden konnten. Die Masterthesis konnte vor allem im Szenario *Einhalten eines Profils* Konzepte aufzeigen und umsetzen, mit denen die Flexibilität der Speicher für einen wärmenetzdienlichen Betrieb genutzt werden kann. Die Umsetzung der Szenarien ist noch ausbaubar, was allerdings den Rahmen der Masterarbeit übersteigt.

Die Kommunikation über das Protokoll Modbus TCP konnte erfolgreich in der Simulationsplattform implementiert und ein Testobjekt für einen Kommunikationstest im Feld aufgesetzt werden. Durch die Implementierung wurde die Möglichkeit geschaffen, die Simulation mit Hardware zu verbinden oder auch SPS aus unterschiedlichen Simulationen kommunizieren zu lassen. Sie ist nicht nur innerhalb der Masterarbeit von Interesse, sondern öffnet die Tür für eine Vielzahl an weiteren Anwendungen.

Die Arbeit mit einer in der Entwicklung befindlichen Simulationsumgebung stellte einen erschwerenden Faktor dar. So können derzeit noch keine Daten aus dem *main loop code* in der Datenbank gespeichert und anschließend in Grafana visualisiert werden. Die Daten mussten während der Arbeit behelfsweise in csv Dateien abgespeichert und im Anschluss visualisiert werden. Das Modell musste außerdem stetig an die sich parallel entwickelnde Simulationsumgebung angepasst und Lösungen für bisher unbekannte Probleme gefunden werden. Zusätzlich befinden sich die Komponenten derzeit noch im Prozess der Validierung, wie an dem Temperaturverlauf der Speicherschichten deutlich wird.

Die Modellierung inklusive aller Regelungen verlief trotzdem erfolgreich und bildet eine gute Grundlage für die Umsetzung der Szenarien und Erweiterungen dieser. Außerdem bietet das Modell eine Basis für weitere Forschung bezüglich der Energiezentrale in Hamburg-Altona, da es ein vollständig regelbares Modell der Zentrale enthält.

Darauf aufbauend konnten die entwickelten Regelszenarien erfolgreich simuliert werden. Für das Szenario *Normalbetrieb* ergibt sich eine Vielzahl unterschiedlicher Umsetzungsmöglichkeiten, aus denen zwei ausgewählt wurden. So wurden in der einen Variante alle Speicher gleichzeitig beladen, während in der anderen Variante die Speicher nacheinander beladen wurden. Es konnte gezeigt werden, dass die Flexibilität der Speicher in beiden Fällen genutzt werden kann, indem die vorgegebene Energiemenge in eine SOC Differenz umgerechnet wird. Im Verlauf der Arbeit fiel auf, dass die Bestimmung der minimalen und maximalen Temperatur und somit der SOC Berechnung an sich, entscheidend ist für die Umsetzung der Szenarien und deren Erfolg. Die Wahl dieser Temperaturen und die sich daraus ergebende Temperaturdifferenz hat einen Einfluss auf die angenommene maximale Speicherkapazität. Ihr fällt somit, gerade in Bezug auf die Nutzung für einen netzdienlichen Betrieb, eine wichtige Rolle zu.
Schlussendlich lässt sich sagen, dass die Aufgabenstellung der Masterthesis erfolgreich bearbeitet werden konnte. Für einen wärmenetzdienlichen Betrieb wurden erste grundlegende Erkenntnisse gewonnen. Abgesehen davon wurde die Simulationsplattform um eine wichtige Kommunikationsform für SPS erweitert und es konnte ein Modell entwickelt werden, das auch in späteren Forschung Verwendung finden kann.

8. Ausblick

Während der Masterthesis waren noch nicht alle Funktionalitäten der Simulationsumgebung lauffähig, wodurch sich nach Abschluss der Arbeit Ausbaumöglichkeiten für das Modell ergeben. So können die bestehenden massenstrombasierten Modelle in Modelle mit Druckberechnung überführt werden, um weitere Erkenntnisse vor allem im Bereich der Netzengpassbewältigung zu sammeln. Die Simulationsplattform sollte außerdem um eine generische Lösung zum Speichern und Visualisieren von Daten im *main loop code* erweitert werden.
In Bezug auf die Implementierung der Pymodbus Bibliothek zeigt sich, dass die Sichtweise von Anwender*innen der Simulationsplattform und Softwarearchitekt*innen unterschiedlich sein kann und deswegen auch die angestrebten Formen der Umsetzung unterschiedlich sind. Während der Arbeit geschah die Implementierung aus Softwarearchitektonischer Sicht, allerdings soll der Fokus im Forschungsprojekt gerade in Bezug auf die SPS in der Simulationsumgebung in Zukunft auf der Anwendungsorientierung liegen. Derzeit befindet sich der *heartbeat* im *process code*, wohingegen die Server und Clients in der *PLC Library* definiert sind. Da auf realen SPS die Kommunikation im Hintergrund abläuft, ergibt ein Umzug des Servers und Clients in den *process code* aus anwendungsorientierter Sicht Sinn. Der *heartbeat* würde auf einer realen SPS in der Regel als selbst entwickelter Baustein in einer Bibliothek gespeichert werden. Der Umbau ist für die Funktionalität nicht notwendig, würde aber einer realitätsnahen Modellierung besser entsprechen. Er kann im Rahmen eines geplanten grundlegenden Umbaus der *PLC Library* nach Abschluss der Masterarbeit erfolgen.
Das Szenario *Einhalten eines Profils* ist der erste Teil des Szenarios *Normalbetrieb*. Die beiden entwickelten Varianten sollten im Anschluss an die Arbeit weitergehend verglichen und Anwendungsfälle definiert und simuliert werden, in denen die Vor- und Nachteile der unterschiedlichen Ausführung zum Tragen kommen. Das Modell ist außerdem zurzeit auf drei iWÜST ausgelegt. Es wurde ein automatisiertes Einlesen der Speicherkapazitäten und der Berechnung der Anteile der iWÜST hinzugefügt, welches unabhängig von der Anzahl der Übergabestationen funktioniert. Das derzeitige Modell beruht jedoch momentan noch an anderen Stellen darauf, dass vorab bekannt ist, wie viele iWÜST mit der W-SPS verbunden sind. So sind beispielsweise die Register der W-SPS für die drei iWÜST angelegt. Es wäre sinnvoll die komplette Regelung so zu erweitern, dass die W-SPS die Strukturen automatisch für jede iWÜST bei ihrer Anmeldung an-

legt.
Des Weiteren sind Erweiterungen in Bezug auf Spitzenlastreduktion aus wirtschaftlichen Gründen oder eine Lastverschiebung wegen eines dynamischen Wärmepreises denkbar. Zur optimalen Einsatzplanung sollten die verfügbaren Speicherkapazitäten der iWÜST durchgehend an die Leitwarte übermittelt werden. Es wäre außerdem interessant zu erforschen, ob die verfügbare Speicherkapazität schon im Vorfeld mit Hilfe von Lastprognose vorausgesagt werden können.
Darüber hinaus ist es sinnvoll, eine einheitliche Verwendung der minimalen und maximalen Temperaturen zur SOC Berechnung zu erarbeiten bzw. grundsätzlich eine Methode für die bestmögliche SOC Bestimmung in Bezug auf Höhe und Breite des Speichers zu entwickeln.
Schlussendlich sollten die Erkenntnisse aus dieser Arbeit mit denen aus der Masterarbeit von Moritz Verbeck [24] zusammen getragen werden, da sich erst hieraus ein zusammenhängendes Konzept sowohl für die iWÜST als auch die W-SPS ableiten lässt.

Literaturverzeichnis

[1] KEBAP E.V. (Hrsg.): *KEBAP - Kultur für Altona*. – URL http://kulturenergiebunker.blogspot.com/p/kultur.html. – Zugriffsdatum: 13.04.2019

[2] ARBEITSGEMEINSCHAFT QM FERNWÄRME: *Planungshandbuch Fernwärme*. Bern, 2018

[3] AVERDUNG INGENIEURGESELLSCHAFT MBH: *Technisch-wirtschaftliche Machbarkeitsstudie Kultur- und Energiebunker Altona*

[4] BECK, Jan-Philip: *Smart Heat Grid Hamburg: Modellierung und Simulation des Energiebunkers Wilhelmsburg als Grundlage einer Konzeptentwicklung zur Betriebsoptimierung und Netzrückspeisung*, HAW Hamburg, Masterarbeit

[5] BRÜGGEMANN, Anke ; KFW RESEARCH (Hrsg.): *Keine Energiewende ohne Wärmewende*

[6] BUNDESMINISTERIUM FÜR UMWELT, NATURSCHUTZ UND NUKLEARE SICHERHEIT: *Der Klimaschutzplan 2050 – Die deutsche Klimaschutzlangfriststrategie*. – URL https://www.bmu.de/themen/klima-energie/klimaschutz/nationale-klimapolitik/klimaschutzplan-2050/#c8420. – Zugriffsdatum: 13.04.2019

[7] EICKER, Ursula: *Solare Technologien für Gebäude. Grundlagen und Praxisbeispiele*. 2012

[8] GEVATTER, Hans-Jürgen: *Handbuch der Mess- und Automatisierungstechnik im Automobil ; Fahrzeugelektronik, Fahrzeugmechatronik*. Berlin, Heidelberg und New York, 2006

[9] HAMBURG ENERGIE: *Smart Heat Grid Hamburg*. – URL http://www.hamburgenergie.de/ueber-uns/unternehmen/forschungsprojekte/smart-heat-grid-hamburg/. – Zugriffsdatum: 13.04.2019

[10] KG, WAGO Kontakttechnik GmbH & Co.: *Modbuskommunikation zwischen WAGO Ethernet Kopplern und Controllern*

[11] KIRCHNER, Kerstin: *Ethernet - Ein Überblick über den Netzwerkstandard*. München, 2004

[12] LERCH, Reinhard: *Elektrische Messtechnik: Analoge, digitale und computergestützte Verfahren ; mit 62 Tabellen ; [Studentenversion LabVIEW 8.2.1 und Übungsaufgaben auf DVD]*. Berlin, 2007

[13] LORENZEN, Peter: *Vortrag: Flexibility in District Heating Systems*

[14] LORENZEN, Peter: *Vortrag: Smart Heat Grid Hamburg: Forschungsprojekt Wärmenetzausbau in Wilhelmsburg*

[15] LORENZEN, Peter: *Das Wärmenetz als Speicher im Smart Grid: Betriebsführung eines Wärmenetzes in Kombination mit einem stromgeführten Heizkraftwerk*, HAW Hamburg, Masterarbeit

[16] LUND, Henrik ; WERNER, Sven ; WILTSHIRE, Robin ; SVENDSEN, Svend ; THORSEN, Jan E. ; HVELPLUND, Frede ; MATHIESEN, Brian V.: 4th Generation District Heating (4GDH) Integrating smart thermal grids into future sustainable energy systems

[17] LUNZE, Jan: *Regelungstechnik 1: Systemtheoretische Grundlagen, Analyse und Entwurf einschleifiger Regelungen*. Berlin, Heidelberg, 2014

[18] METTER, Mark ; BUCHER, Rainer: *Industrial Ethernet in der Automatisierungstechnik: Planung und Einsatz von Ethernet-LAN-Techniken*. Erlangen, 2012

[19] NORDDEUTSCHE ENERGIEWENDE 4.0: *Energiebunker Altona*. – URL `https://new4-0.erneuerbare-energien-hamburg.de/de/new-40-projekte/details/energiebunker-altona.html`. – Zugriffsdatum: 13.04.2019

[20] RIGGERT, Wolfgang: *Rechnernetze: Grundlagen - Ethernet - Internet*. München, 2014

[21] SANJAY: *PyModbus Documentation*. 2018

[22] SCHLEICHER, Manfred ; JUMO GMBH & CO. KG (Hrsg.): *Regelungstechnik: Grundlagen und Tipps für den Praktiker*

[23] STEPHAN, Peter ; MEWES, Dieter ; KABELAC, Stephan ; KIND, Matthias ; SCHABER, Karlheinz ; WETZEL, Thomas: *VDI-Wärmeatlas*. Wiesbaden, 2017

[24] VERBECK, Moritz: *Smart Heat Grid Hamburg: Konzeptionierung und prototypische Umsetzung einer intelligenten Wärmeübergabestation zur Erprobung einer angepassten SPS-Programmiermethodik*, HAW Hamburg, Masterarbeit, 2019

[25] YAML: *YAML 1.2*. – URL https://yaml.org/. – Zugriffsdatum: 13.04.2019

A. Anhang

Quelltext A.1: Modbus RTU Server auf Raspberry Pi

```
257 class ModbusServer:
258
259     FX_WRITE_MULTIPLE_REGISTERS = 16
260     FX_WRITE_SINGLE_COIL = 5
261     BAUDRATE = 9600
262     RASP_NUMBER = 5
263     PORT = '/dev/ttyUSB0'
264
265     def __init__(self, n_floats=2, n_boolean=1):
266         self.slave_id = 0x01
267
268         store = ModbusSlaveContext(hr=ModbusSequentialDataBlock(1, [self.RASP_NUMBER]*n_floats),
269                                    co=ModbusSequentialDataBlock(1+n_floats, [False]*n_boolean))
270         self.context = ModbusServerContext(slaves=store, single=True)
271         identity = ModbusDeviceIdentification()
272
273         def modbus_server_thread():
274             StartSerialServer(self.context, framer=ModbusRtuFramer, identity=identity, port=self.PORT, timeout
                     =.005, baudrate=self.BAUDRATE)
275             print("Starting Server")
276         thread = Thread(target=modbus_server_thread)
277         thread.start()
278
279     def write_multiple_registers(self, address, values):
280         self.context[self.slave_id].setValues(fx=self.FX_WRITE_MULTIPLE_REGISTERS, address=address, values=values)
281
282     def write_single_coil(self, address, values):
283         self.context[self.slave_id].setValues(fx=self.FX_WRITE_SINGLE_COIL, address=address, values=values)
284
285
286 def heartbeat_thread():
287
288     context = ModbusServer(n_floats=1)
289     heartbeat = True
290     while True:
291         address = 3
292         heartbeat = not heartbeat
293         values = [heartbeat]
294         print(heartbeat)
295         context.write_single_coil(address=address, values=values)
296         time.sleep(2)
297
298
299 thread_heartbeat = Thread(target=heartbeat_thread)
300 thread_heartbeat.start()
```

Quelltext A.2: Client zur Raspberry Pi Abfrage über Modbus RTU

```
302
303 from pymodbus.client.sync import ModbusSerialClient as ModbusClient
304 import time
305
306 UNIT = 0x01
307
308 client = ModbusClient(method='rtu', port='/dev/ttyUSB0', baudrate=9600)
309
310 while True:
311     succeeded = client.connect()
312     if succeeded:
313         while True:
314             try:
315                 value1 = client.read_coils(3, UNIT)
316                 print(value1.bits[0])
317                 time.sleep(2)
318                 value2 = client.read_coils(3, UNIT)
319                 print(value2.bits[0])
320                 time.sleep(2)
321
322                 if value1.bits[0] == value2.bits[0]:
323                     print("Lost heartbeat from Server")
324                     break
325
326             except AttributeError as e:
327                 print(e)
328
329         break
330     else:
331         print("Connection with Server failed")
```

Tabelle A.1.: PID-Parameter, Einschaltverzögerungen und Mindestlaufzeiten der Komponenten

Energiebunker als Erzeugermodell	K_p	T_v	T_n	**timeWait / min**	**timeWaitOn / min**
Kessel 1	0.000001	∞	0	30	15
Kessel 2	0.000005	∞	0	10	15
Kessel 3	0.000008	∞	0	10	20
Kessel 4	0.00004	∞	0	1	25
BHKW 1	0.00005	∞	0	15	10
BHKW 2	0.00005	∞	0	15	10
Pumpe 1	-0.000001	∞	0		
Pumpe 2	-0.000001	∞	0		
Pumpe 3	-0.000001	∞	0		
Pumpe 8	0.000003	∞	11		
Vereinfachtes Last und Netzmodell	K_p	T_v	T_n	**timeWait / min**	**timeWaitOn / min**
Speicher 1	0.0001	∞	38		
Speicher 2	0.0001	∞	38		
Speicher 3	0.0001	∞	38		
Zusammengeführtes Modell	K_p	T_v	T_n	**timeWait / min**	**timeWaitOn / min**
Kessel 1	0.000001	∞	0	30	15
Kessel 2	0.000005	∞	0	10	10
Kessel 3	0.000008	∞	0	5	10
Kessel 4	0.000007	∞	0	1	25
BHKW 1	0.00005	∞	0	15	10
BHKW 2	0.00005	∞	0	15	10
Pumpe 1	-0.000001	∞	0		
Pumpe 2	-0.000001	∞	0		
Pumpe 3	-0.000001	∞	0		
Pumpe 8	0.000001	∞	11		
Speicher 1	0.00008	∞	38		
Speicher 2	0.00008	∞	38		
Speicher 3	0.00008	∞	38		

Quelltext A.3: *YAML* Datei des bestehenden wärmegeführten BHKW


```yaml
type_name: CHPPlant
canvas_size:
- 10.0
- 10.0
connector_arrays:
 convective_heat_flow:
   connector_type_name: Convective heat flow mT
   connector_array_interfaces:
     mass_flow_quotient:
       readable: true
       writable: false
       trigger: false
     temperature:
       readable: true
       writable: true
       writing_output_name: temp_flow
       trigger: true
       triggering_output_name: temp_trigger
   visible: true
   scalable: false
   default_connector_names:
   - Input socket
   - Output socket
   canvas_orientations:
   - b
   - t
 current_signal:
   connector_type_name: Control current
   connector_array_interfaces:
     current_signal:
       readable: true
       writable: false
       trigger: false
     supply_voltage:
       readable: false
       writable: false
       trigger: false
   visible: true
   scalable: false
   default_connector_names:
   - clamp
   canvas_orientations:
   - l
parameters:
 CHP_COEFFICIENT:
   default_value: '0.6'
   quantity_name: Dimensionless
 HEAT_FLOW_MAX:
   default_value: '350'
   quantity_name: Heat flow
methods:
 initialize: {}
 mass_flow:
   outputs:
     temp_trigger:
       quantity: Logical
   inputs:
     mass_flow_quotient:
       external: true
       connected_connector_array: convective_heat_flow
       connected_interface: mass_flow_quotient
 temperature:
   outputs:
     heat_flow:
       quantity: Power
     temp_flow:
       quantity: Temperature
   inputs:
     control_value:
       external: true
       connected_connector_array: current_signal
       connected_interface: current_signal
     mass_flow_quotient:
       external: true
       connected_connector_array: convective_heat_flow
       connected_interface: mass_flow_quotient
     temp_return:
       external: true
       connected_connector_array: convective_heat_flow
       connected_interface: temperature
panels:
 Temperature:
 - temp_flow
 Heat Flow:
```

```
 - heat_flow
```


Quelltext A.4: *YAML* Datei des stromgeführten BHKW


```
type_name: CHPPlant_power_driven_from_timeseries
canvas_size:
- 10.0
- 10.0
connector_arrays:
 convective_heat_flow:
   connector_type_name: Convective heat flow mT
   connector_array_interfaces:
     mass_flow_quotient:
       readable: true
       writable: false
       trigger: false
     temperature:
       readable: true
       writable: true
       writing_output_name: temp_flow
       trigger: true
       triggering_output_name: temp_trigger
   visible: true
   scalable: false
   default_connector_names:
   - Input socket
   - Output socket
   canvas_orientations:
   - b
   - t
 current_signal:
   connector_type_name: Control current
   connector_array_interfaces:
     current_signal:
       readable: true
       writable: false
       trigger: false
     supply_voltage:
       readable: false
       writable: false
       trigger: false
   visible: true
   scalable: false
   default_connector_names:
   - clamp
   canvas_orientations:
   - l
parameters:
 CHP_COEFFICIENT:
   default_value: '0.6'
   quantity_name: Dimensionless
 HEAT_FLOW_MAX:
   default_value: '550'
   quantity_name: Heat flow
 POWER_SERIES:
   default_value: "No timeseries defined in PI-Project."
   quantity_name: "Text"
 POWER_SOURCE:
   default_value: "No timeseries defined in PI-Project."
   quantity_name: "Text"
methods:
 initialize: {}
 mass_flow:
   outputs:
     temp_trigger:
       quantity: Logical
   inputs:
     mass_flow_quotient:
       external: true
       connected_connector_array: convective_heat_flow
       connected_interface: mass_flow_quotient
 temperature:
   outputs:
     heat_flow:
       quantity: Power
     temp_flow:
       quantity: Temperature
   inputs:
     mass_flow_quotient:
       external: true
       connected_connector_array: convective_heat_flow
       connected_interface: mass_flow_quotient
     temp_return:
```

```
80        external: true
81        connected_connector_array: convective_heat_flow
82        connected_interface: temperature
83      control_value:
84        external: true
85        connected_connector_array: current_signal
86        connected_interface: current_signal
87  panels:
88   Temperature:
89   - temp_flow
90   Heat Flow:
91   - heat_flow
```

Quelltext A.5: Beispiel Modbus TCP Verwendung in Jarvis (*initialize_code*)

```
1   """
2   This code file will be written as a monolith parameter to the database and read during PLC initialization.
3   For this to work, avoid single quotes in the code and use double quotes (") instead.
4
5   Add controllers from plc library.
6   """
7
8   self.temp_controller = plc_lib.PID()
9   self.drive_controller = plc_lib.AnalogDriveController()
10  self.server_context = plc_lib.PyModbusServer(port=config.MODBUS_PORT_SOFT_PLC,heartbeat=True, heartbeat_address=5,
        heartbeat_dt=5, name="Server_WSPS", n_floats=10, n_boolean=10)
11  self.client1 = plc_lib.PyModbusClient("localhost", port=config.MODBUS_PORT_SOFT_PLC + 1,heartbeat=True,
        heartbeat_address=5, heartbeat_dt=5, name="Client_WSPS",)
```

Quelltext A.6: Beispiel Modbus TCP Verwendung in Jarvis (*main_loop_code*)

```
1   """
2   This code file will be written as a monolith parameter to the database and read during PLC initialization.
3   For this to work, avoid single quotes in the code and use double quotes (") instead.
4
5   The "Read inputs" region corresponds to CodeSys input configuration.
6
7   The "Push outputs" region corresponds to CodeSys output configuration.
8
9   One can define multiple PLC main loop tasks, where each corresponds to a single control system.
10
11  The PLCs main loop tasks are combined from the regions "main loop" and "process":
12      - In "Main loop" region, one can choose Python or CodeSys files, which have to be executed (see doc string)
13      - In "Process" region, one can write usual python code directly (see doc string)
14  """
15  import time
16  #  region Read inputs
17
18  # All inputs of one card need to be get at once
19  cpu_inputs = self.get_card_inputs(card="plc_cpu1")
20  # Afterwards, its specific connector inputs can be read from dict
21  d_time = cpu_inputs["clamp"]
22
23  temperature_inputs = self.get_card_inputs(card="plc_temp1")
24  current_temp = temperature_inputs["clamp1"]
25
26  start_address_floats = 0
27  start_address_coils = 0
28  values_registers = [17]
29  values_coils = [False]
30
31  self.server_context.write_multiple_registers(address=address_floats, values=values_registers)
32  self.server_context.write_coils(address=address_coils, values=values_coils)
33
34
35  value_registers = self.client1.read_holding_registers(start_address_floats)
36  value_coils = self.client1.read_coils(start_address_coils)
37  print("Register Value from PLC2 {}".format(value_registers.registers[0]))
38  print("Coil Value from PLC2 {}".format(value_coils.bits[0]))
39
40
41  # region Main loop
42  """
43  Python or CodeSys task files have to be dropped in example_projects/PLC_code/project_name/plc_monolith_name/
        filename
44  To execute them in every time step write down their file names in this region in the following lines
45  """
46  # test.st
47  # endregion
48
49
```

```
50  # region Process
51  """
52  Here one can write python code as usual.
53  """
54
55  # region Task 1: Calculate pump control signal from input temperature
56  u = self.temp_controller.calculate_control_value(d_time=d_time, ACTUAL=current_temp, SET_POINT=90+273.15, KP
        =0.0004, TN=1000, TV=0, Y_MANUAL=0,
57                                                   Y_OFFSET=0, Y_MIN=0, Y_MAX=0.02, MANUAL=False, RESET=False)
58
59  # Calculate the change of the pump position
60  control_signal = self.drive_controller.calculate_position(u=u, y_min=0.004, y_max=0.02)
61  # endregion
62  # endregion
63
64  # region Push outputs
65  # All outputs of one card need to be pushed at once. Therefore, all connector_name-value combinations have to added
         to the following dict
66  connector_value_dict = {"clamp1": control_signal}
67  self.push_card_outputs(card="plc_ao1", connector_value_dict=connector_value_dict)
68  # endregion
```

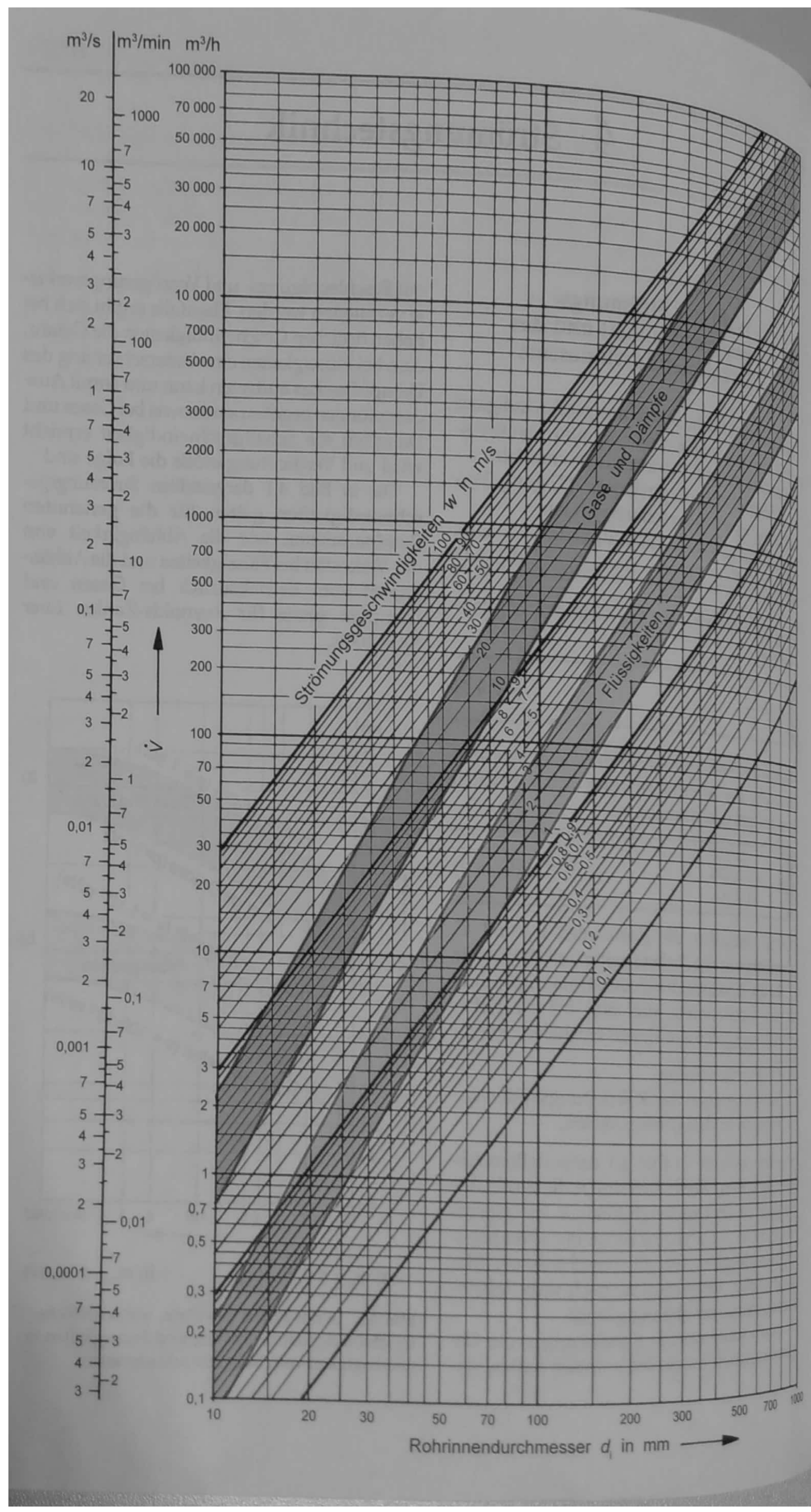

Abbildung A.1.: Rohrinnendurchmesser in Abhängigkeit von Strömungsgeschwindigkeit und Volumenstrom

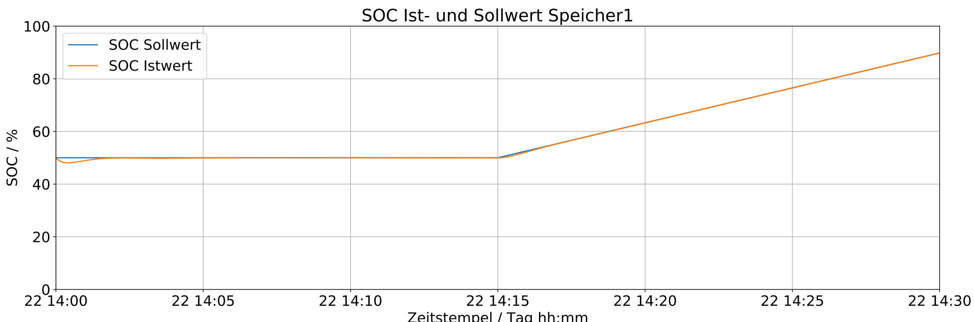

Abbildung A.2.: SOC Verlauf der iWÜST1

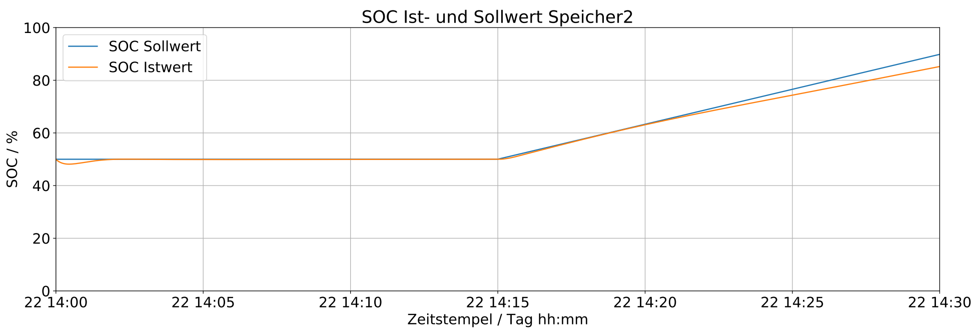

Abbildung A.3.: SOC Verlauf der iWÜST2

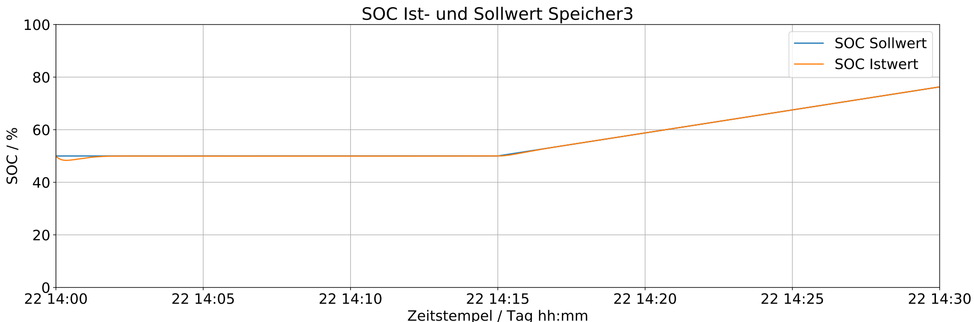

Abbildung A.4.: SOC Verlauf der iWÜST3

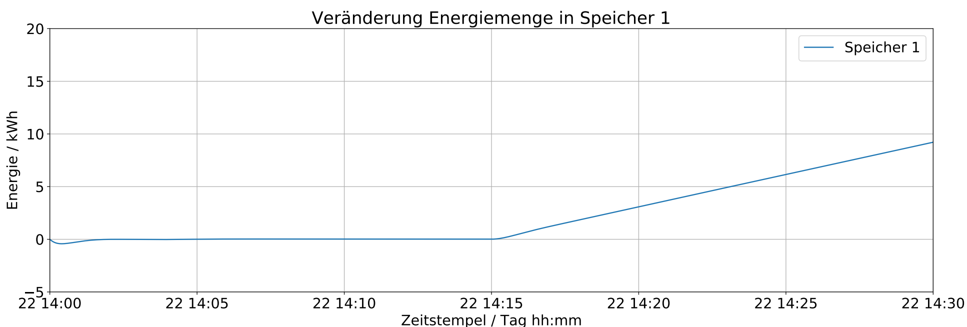

Abbildung A.5.: Änderung der Energie im Speicher der iWÜST1

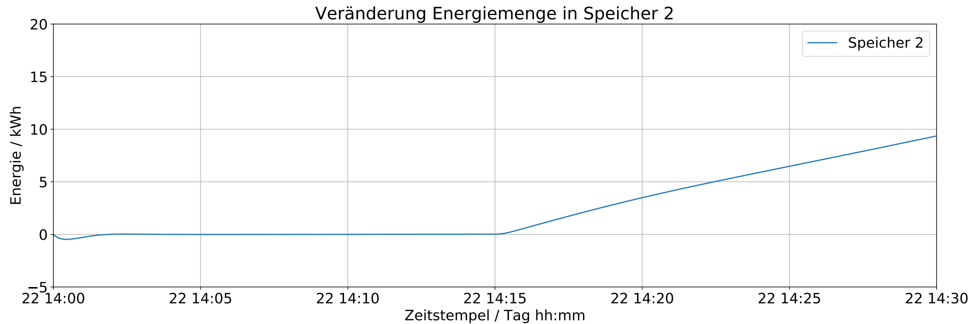

Abbildung A.6.: Änderung der Energie im Speicher der iWÜST2

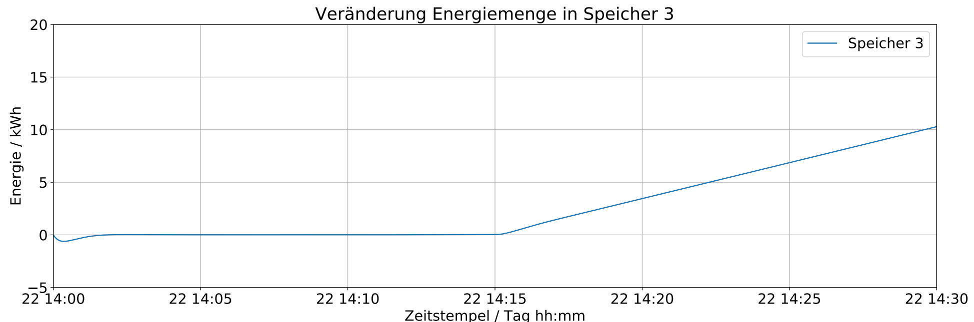

Abbildung A.7.: Änderung der Energie im Speicher der iWÜST3

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 8. Mai 2019	
Ort, Datum	Unterschrift