Hochschule Merseburg

University of Applied Sciences

Fachbereich Ingenieur- und Naturwissenschaften

Bachelor-Studiengang Green Engineering „Gestaltung nachhaltiger Prozesse“

Bachelorarbeit

Programmierung einer Reglerplattform in Python

Verantwortlicher Hochschullehrer:

Prof. Dr. -Ing. Andreas Ortwein

betriebliche Betreuerin:

Dr.-Ing. Steffi Theurich

eingereicht von:

Ferdinand Staude

Matrikelnummer: 24453

Abgabedatum: 19.05.2021

Inhalt

Abbildungsverzeichnis v

Tabellenverzeichnis vii

Abkürzungsverzeichnis viii

1 Einleitung 1

1.1 Motivation 1

1.2 Aufgabenstellung 4

2 Grundlagen 5

2.1 Technologischer Hintergrund 5

2.1.1 Reglerplattformen für Gebäude 5

2.1.2 Modbus-Technologie 5

2.2 Programmiertechnischer Hintergrund 6

2.2.1 LabVIEW 6

2.2.2 Python 7

2.2.3 Objektorientierte Programmierung 8

2.2.4 Unified Modeling Language (UML) 9

2.2.5 ModbusPal 10

2.2.6 Multithreading 11

2.3 Softwarevalidierung 12

2.3.1 Funktionstests 12

2.3.2 Nicht-funktionale Tests 13

3 Darstellung des LabVIEW-Programms 14

3.1 Herangehensweise 14

3.2 Beschreibung der LabVIEW-Plattform 14

3.3 Beschreibung der grafischen Oberfläche des LabVIEW-Programmes 17

4 Python-Programm 24

4.1 Programmierparadigma 24

4.2 Anforderungen an die Reglerplattform 24

4.3 Unterschiede zwischen der Python- und LabVIEW-Version 25
4.4 Programmaufbau 26
4.5 Umsetzung der Bausteine der Reglerplattform 26
4.5.1 Graphical User Interface (GUI) 28
4.5.2 GlobaleDaten 33
4.5.3 Regler-Modbus-Schnittstelle 36
4.5.4 Modbus-Schnittstellen: ModbusFA und ModbusHR 38
4.5.5 Regler 43
5 Testläufe 45
5.1 Komponententests 45
5.1.1 Komponententest: GUI 45
5.1.2 Komponententest: ModbusFA und ModbusHR 46
5.2 Integrationstests 46
5.2.1 Integrationstest: GUI 46
5.2.2 Integrationstest: Regler-Modbus-Schnittstelle 46
5.2.3 Integrationstest Reglerplattform ohne Regler 46
5.3 Gesamttest 47
6 Fazit 49
6.1 Zusammenfassung 49
6.2 Ausblick 50
7 Literaturverzeichnis 52
8 Anhang 54
Anhang A Komponententest 54
Anhang B Integrationstests 61
Anhang C Gesamttest 72

Abbildungsverzeichnis

Abbildung 1-1: Anteil erneuerbarer Energien in verschiedenen Sektoren [4] 2
Abbildung 1-2: Plan einer Heizungsanlage mit Feuerungsstelle und Solarthermie [5] 3
Abbildung 2-1: LabVIEW Sequenzstruktur 7
Abbildung 2-2: Beispiel für eine Klasse und ein abgeleitetes Objekt 9
Abbildung 2-3: Oberfläche ModbusPal..11
Abbildung 3-1: UML-Diagramm: LabVIEW-Programm Reglerlplattform..................15
Abbildung 3-2: LabVIEW-GUI: Reiter „Heating system“ 18
Abbildung 3-3: LabVIEW-GUI: Reiter "Heat meter" 19
Abbildung 3-4: LabVIEW-GUI: Reiter "Settings" 20
Abbildung 3-5: LabVIEW-GUI: Reiter "DWD Data" 21
Abbildung 3-6: LabVIEW-GUI: Reiter "Data logging"...22
Abbildung 3-7: LabVIEW-GUI: Reiter "Charts" 23
Abbildung 4-1: Wirkungsschema des Python-Programms 26
Abbildung 4-2: Python-GUI: Reiter: "Settings" 29
Abbildung 4-3: Python-GUI: Quellcode "FAFrame" 31
Abbildung 4-4: Python-GUI: Reiter: "Heating system"...32
Abbildung 4-5: Python-GUI: Reiter: "Data logging"...33
Abbildung 4-6: UML-Diagramm: Python „Regler_Modbus_Schnittstelle“..................36
Abbildung 4-7: Quellcode der Funktion „__init__“ ModbusFA 39
Abbildung 4-8: Quellcode der Funktion „initialisieren“ ModbusFA............................39
Abbildung 4-9: Quellcode der Lesefunktion von ModbusFA 40
Abbildung 4-10: Quellcode der Schreibfunktion von ModbusFA................................40
Abbildung 4-11: Funktion Trim_Amb_Temp in ModbusHR zur Umrechnung negativer Temperaturen 43
Abbildung 6-1: Schema für einen erweiterten Puffer 51
Abbildung 8-1: Zusatzanweisungen für den Funktionstest der GUI 54
Abbildung 8-2: Funktionstest GUI – Überprüfung der Eingabefunktion 55
Abbildung 8-3: Funktionstest GUI – Überprüfung der Modbus-Steuerung..................56
Abbildung 8-4: Funktionstest ModbusFA – HoldingRegister ModbusPal 57
Abbildung 8-5: Funktionstest ModbusFA – Zusatzanweisungen im Quellcode 58
Abbildung 8-6: Funktionstest Modbus – Visuelles Feedback in ModbusPal.................58
Abbildung 8-7: Funktionstest ModbusFA – Ergebnis in der Konsole 59
Abbildung 8-8: Funktionstest ModbusHR – Zusatzanweisungen und HoldingRegister 60

Abbildung 8-9: Funktionstest ModbusHR – Ergebnis in der Konsole 61
Abbildung 8-10: Funktionstest ModbusHR – Geänderte Werte im HoldingRegister ...61
Abbildung 8-11: Integrationstest GUI..64
Abbildung 8-12: Integrationstest GUI – verwendeter Quellcode 65
Abbildung 8-13: Integrationstest GUI –Konsolenausgabe...65
Abbildung 8-14: Integrationstest: Regler-Modbus-Schnittstelle –Quellcode 66
Abbildung 8-15: Integrationstest Regler-Modbus-Schnittstelle Modbuseinstellungen ..66
Abbildung 8-16: Integrationstest Regler-Modbus-Schnittstelle – Konsolenausgabe 67
Abbildung 8-17: Integrationstest – ohne Regler GUI Reiter: „Settings“.......................68
Abbildung 8-18: Integrationstest – ohne Regler GUI Reiter „Heating System“ 69
Abbildung 8-19: Integrationstest–ohne Regler – Manueller Modus Ergebnis in ModbusPal 70
Abbildung 8-20: Integrationstest – ohne Regler – GUI Manueller Modus....................71

Tabellenverzeichnis

Tabelle 1: Objekttypen des Modbus-Protokolls 6
Tabelle 2: Übersicht der Reiter des LabVIEW Programmes 17
Tabelle 3: Übersicht der importierten Module und deren Einbindung in den erstellten Modulen 27
Tabelle 4: Daten im globalen Speicher 34
Tabelle 5: Daten Regler_Modbus_Schnittstelle 37
Tabelle 6: Werte für KOC-Mode 38
Tabelle 7: Übersicht der Funktionen der Modbus-Module 39
Tabelle 8: Gelesene und geschriebene Werte von ModbusFA 41
Tabelle 9: Gelesene und geschriebene Werte von ModbusHR 42
Tabelle 10: Schnittstelle Regler-Reglersteuerung 44
Tabelle 11: Übersicht der Variablen für den Regler 48
Tabelle 12: Vergleich des Ergebnis der Reglerplattform mit dem Ergebnis aus dem Praktikum 48
Tabelle 13: Funktionstest ModbusFA – Änderung zwischen den Ausgaben 59
Tabelle 14: Werte für den Integrationstest der GUI 63
Tabelle 15:Integrationstest – ohne Regler Werte Manueller Modus 70
Tabelle 16: Auswahl vom Ergebnis des Gesamttests 73

Abkürzungsverzeichnis

csv	Character separated values
CO_2	Kohlenstoffdioxid
DBFZ	Deutsches Biomasseforschungszentrum gemeinnützige GmbH
EEG	Erneuerbare Energien Gesetz
DWD	Deutscher Wetterdienst
FTP	File transfer protocol
GEG	Gebäude Energie Gesetz
IPCC	Intergovermental Panel on Climate Change
KWK	Kraft-Wärme-Kopplung
LabVIEW	Laboratory Virtual Instrumentation Engineering Workbench
SNuKR	Steigerung des Nutzens von kleinen, biomassebefeuerten BHKWs durch bedarfsgerechte Regelung
TCP	Transmission Control Protocol
UML	Unified Modeling Language
GUI	Graphical User Interface
VI	Virtuelle Instrumente

1 Einleitung

Seit Beginn der Industrialisierung werden Treibhausgase, insbesondere Kohlenstoffdioxide (CO_2), freigesetzt. In einem Bericht von 2007 kommt das IPCC (Intergovermental Panel on Climate Change) zu dem Ergebnis, dass steigende Temperaturen in der Atmosphäre und in den Ozeanen vor allem auf vom Menschen freigesetzte Treibhausgase zurück zu führen sind [1]. Durch den Anstieg der globalen Temperatur nehmen Extremwetterereignisse immer mehr zu. Folgen davon können, neben den offensichtlichen Gefahren von Waldbränden und Überschwemmungen, auch globale Konflikte im großem Ausmaß sein. Um diesen Effekten entgegen zu wirken, müssen die Treibhausgasemissionen stark verringert werden.

1.1 Motivation

Im Jahr 2015 einigten sich 195 Vertragsparteien auf das „Paris Agreement“. Damit erklären die Vertragsparteien die Bereitschaft an einem globalen Klimaschutz mitzuwirken. Das Gesamtziel globaler Klimaschutz wird darin in Artikel 2 in drei Unterpunkte unterteilt. [2]

a) Der Anstieg der globalen Temperatur ist, im Vergleich zum vorindustriellen Niveau, auf 2 °C zu begrenzen, wobei sogar versucht werde soll einen Temperaturanstieg von 1,5 °C zu erreichen.
b) Die Anpassungsfähigkeit an Auswirkungen durch den Klimawandel soll erhöht werden. Außerdem sollen Treibhausgase durch die Förderung von klimaneutraler Entwicklung, reduziert werden, ohne dass die Nahrungsmittelproduktion gefährdet wird.
c) Finanzströme und die Ziele zur Stabilisierung des Klimas müssen miteinander vereinbar sein.

In Deutschland wurde vor allem im Bereich der elektrischen Energieversorgung viel entwickelt, um einen Umstieg auf erneuerbare Energien zu ermöglichen. Bereits im Jahr 2000 wurde dafür das Erneuerbare-Energien-Gesetz (EEG) verabschiedet [3]. Im Jahr 2020 hatten, aufgrund von finanzieller Unterstützung und gesetzlicher Förderung, erneuerbare Energieträger in Deutschland einen Anteil von fast 45,4 % am Bruttostromverbrauch. Durch die Nutzung von erneuerbaren Energien bei der Stromerzeugung, konnten alleine im Jahr 2020 ca. 181 Millionen Tonnen CO_2-Aquivalente eingespart werden [4].

Während aber für die Bereitstellung von elektrischer Energie bereits eine große Verringerung von CO_2-Emissionen erreicht wurde, steht der Ausbau von erneuerbaren Energien im Bereich der Wärmebereitstellung noch am Anfang.

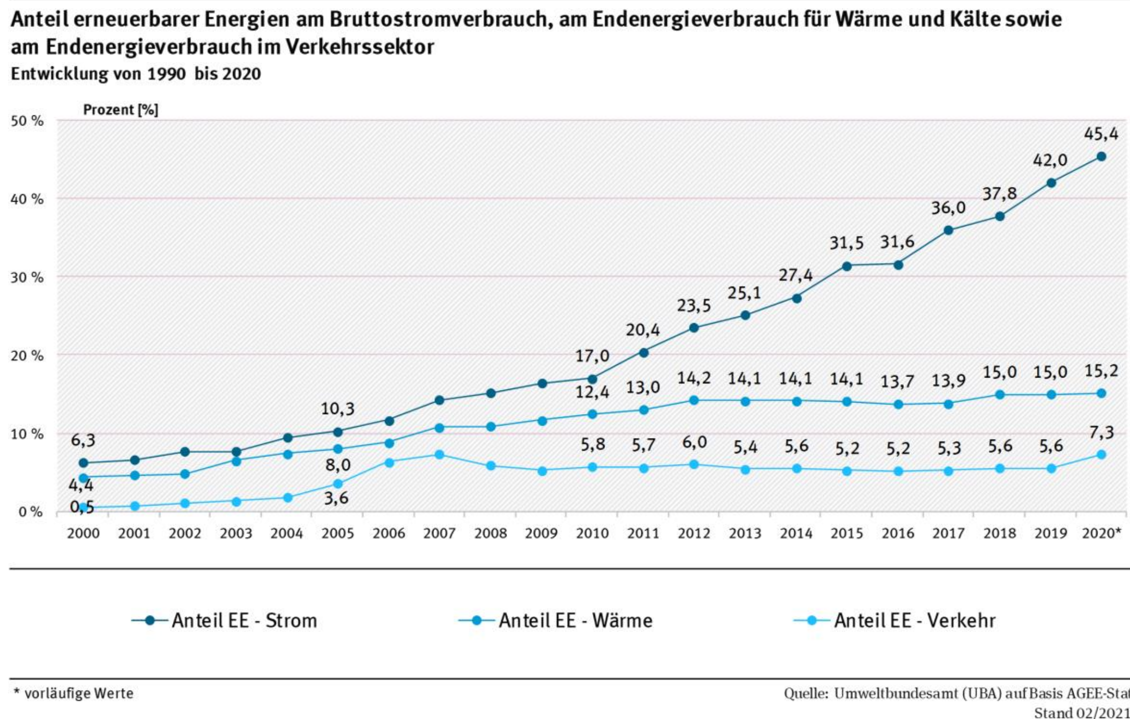

Abbildung 1-1: Anteil erneuerbarer Energien in verschiedenen Sektoren [4]

In Abbildung 1-1 ist zu erkennen, dass der Anteil an erneuerbaren Energien im Stromsektor seit 2010 stark gestiegen ist und zwar von 17,0 % im Jahr 2010 auf 45,4 % im Jahr 2020. Im Wärmesektor ist der Anteil an erneuerbaren Energien ist lediglich von 12,4 % im Jahr 2010 auf 15,2 % im Jahr 2020 gestiegen. Durch den bisher geringen Ausbau von erneuerbaren Energien bei der Wärmeenergieversorgung, gibt es in diesem Sektor noch ein großes Potential, um CO_2-Emissionen zu verringern. Das im November 2020 verabschiedete Gebäude-Energien-Gesetz soll die Grundlage dafür schaffen dieses Potenzial besser nutzen zu können.

Das Deutsche Biomasseforschungszentrum GmbH (DBFZ) setzt zur Verringerung von CO_2-Emissionen auf biomassebasierte Technologien. Aus der Kombination einer Kesselheizung mit einer Solarthermieanlage und/oder einer Wärmepumpe ergeben sich viele unterschiedliche Konzepte zur Bereitstellung von Wärmeenergie in privaten Haushalten. In Systemen mit kombinierter Wärmebereitstellung wird häufig jedes Teilsystem unabhängig voneinander geregelt. Daraus resultieren Systeme, die sich aus vielen unterschiedlichen Systemen zusammensetzen, die nicht miteinander kommunizieren, was wiederum

zu einer geringeren Gesamteffizienz der Anlage führt. Die geringe Gesamteffizienz einer solchen Anlage, ist wegen fehlender Messinstrumente häufig nicht erkennbar.

Im Projekt „KombiOpt“[1] vom DBFZ wird ein Haus mit kombinierter Wärmebereitstellung untersucht. Das Haus, das in dem Projekt untersucht wurde, hatte eine Pelletheizung, eine Solarthermieanlage und einen Pufferspeicher zur Bereitstellung von Wärme. Außerdem verfügte das Haus über zwei Heizkreise und ein Trinkwasserkreislauf. Der Aufbau des Untersuchungsgegenstands ist in Abbildung 1-2 dargestellt. Ergebnis des Projekts „KombiOpt“ war eine Plattform für Feldmessungen. Die Plattform wurde an dem in Abbildung 1-2 dargestellten Untersuchungsgegenstand erfolgreich getestet [5]. Die gesammelten Daten sollen der Entwicklung von Regelstrategien zur kombinierten Wärmebereitstellung dienen.

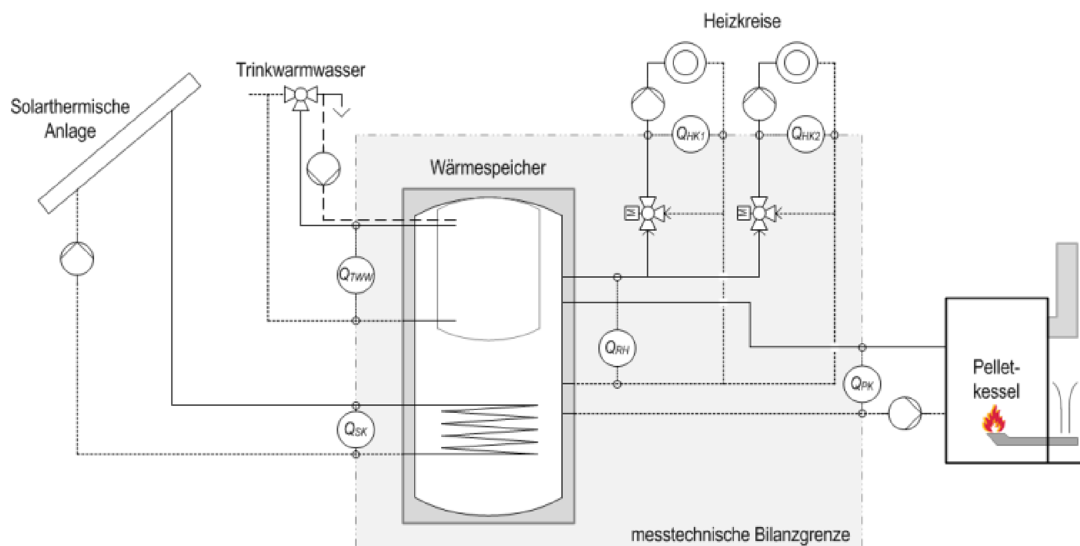

Abbildung 1-2: Plan einer Heizungsanlage mit Feuerungsstelle und Solarthermie [5]

Im Projekt „SNuKR“[2] vom DBFZ wurde ein prädiktiver Regler auf Grundlage der Daten aus dem Projekt „KombiOpt“ entwickelt. Der Regler soll, durch die Kombination einer Biomasse-KWK-Anlage und entsprechenden Speichertechnologien, eine dezentrale bedarfsorientierte Energieversorgung ermöglichen. Unterschiedliche Regelstrategien sollen für den Betreiber den maximalen Nutzen aus der Anlage holen und gleichzeitig eine hohe Netzdienlichkeit gewährleisten, indem die erzeugte elektrische Energie in das Stromnetz

[1] https://www.fnr.de/index.php?id=11150&fkz=22403113 (letzter Zugriff am: 20.05.2021)
[2] .https://www.energetische-biomassenutzung.de/projekte-partner/details/project/show/Project/SNuKR-545 (letzter Zugriff am: 20.05.2021)

gespeist werden kann. Zur Untersuchung von unterschiedlichen Regelungsstrategien wurde deshalb die modulare Reglerplattform aus dem „KombiOpt“-Projekt mit diversen Schnittstellen weiterentwickelt. [6]

Die entwickelte Reglerplattform basiert auf dem Programmiersystem LabVIEW. Diese Plattform soll in Python übersetzt werden, da bei dem Einsatz von LabVIEW immer eine Entwicklungslizenz der Firma National-Instruments benötigt wird. Weiterhin sind LabVIEW basierte Regler an National-Instruments-Hardware Komponenten gebunden.

1.2 Aufgabenstellung

Ausgangspunkt

Im Rahmen verschiedener Projekte wird am Deutschen Biomasseforschungszentrum in der Arbeitsgruppe „Bedarfsgerechte Kraft-Wärme-Kopplung“ eine modulare Reglerplattform als Basis für die Prototypenentwicklung und den Test unterschiedlicher Regelungsansätze entwickelt. Sie soll wesentliche Grundfunktionen beinhalten, sich mit geringem Aufwand an den jeweiligen Anwendungsfall und das dazugehörige Betriebsregime adaptieren als auch sich um weitere Funktionalitäten ergänzen lassen. Die Entwicklungen können entweder intern oder einem Hardware-in-the-Loop (HiL)-Simulator praxisnah getestet und validiert werden.

Teilaufgaben

Vor diesem Hintergrund sollen in der Bachelor-Arbeit die folgenden Schwerpunkte bearbeitet werden:

1. Dokumentation der in LabVIEW vorhanden Reglerplattform mit Hilfe von UML Aktivitätsdiagrammen
2. Modularisierung der Reglerplattform: Aufteilen der Plattform in einzelne Module, die als Unterprogramme erstellt und anschließend wieder verknüpft werden können
3. Programmierung der wesentlichsten Module in Python
4. Beschreibung der in LabVIEW vorhanden graphischen Benutzeroberfläche (GUI)
5. Erstellen einer GUI zur Interaktion mit dem Python-Programm der Reglerplattform (Umsetzung der GUI soll exemplarisch anhand der Reiter ‚Heating System‘ und ‚Settings‘ erfolgen)
6. Dokumentation der softwaretechnischen Umsetzung

2 Grundlagen

2.1 Technologischer Hintergrund

2.1.1 Reglerplattformen für Gebäude

In der Gebäudeautomation können Reglerplattformen vielfältige Aufgaben wahrnehmen. Die umfassende Messung und Dokumentation von Daten stellt die Grundlage für ein gutes Energiemanagement und ermöglicht die gezielte Planung von Maßnahmen, um die Energieeffizienz zu steigern. Das digitale Erfassen und Speichern von Daten ermöglicht, dass deutlich größere Menge an Daten gesammelt und ausgewertet werden können. Weiterhin ist mit diesen Daten eine Energievermarktung möglich. Wegen der umfassenden Analyse der Daten kann das Verhalten der Anlage präzise bestimmt werden. Daraus kann anschließend ermittelt werden, wann wie viel Energie in das Netz eingespeist werden kann. Der Vorteil einer Reglerplattform ist auch, dass diese Prozesse nicht für jedes Gebäude neu ermittelt werden müssen. Mit steigender Anzahl an Datensätzen lassen sich auch neue Projekte besser planen [7] [8].

2.1.2 Modbus-Technologie

Für die Reglerplattform ist ein Modbus-Kommunikation von zentraler Bedeutung. Über die Schnitstellen werden die Daten von der Feuerungsanlage und von den Heizregistern eingesammelt. Das Modbus-Protokoll ist ein Kommunikationsprotokoll zwischen mindestens zwei Endgeräten. Das TCP/IP-Protokoll ist dabei anerkannter Standard für den Datenaustausch in heterogenen Netzen. Bei heterogenen Netzen können Hardware, Topologie und Betriebssysteme voneinander unterschiedlich sein. Das Buszugriffverfahren läuft dabei kontrolliert und über eine zentrale Busverteilung ab. Die Kommunikationsteilnehmer werden in zwei Kategorien unterteilt. Der Server stellt den passiven Teilnehmern, der nur auf Anfragen des aktiven Teilnehmers der Bussteuereinheit/des Clients, reagiert [9].

Die Datenübertragung erfolgt in Datenblöcken. Jeder Datenblock verfügt über einen Header in dem die für Übertragung und Auswertung relevanten Informationen gespeichert sind. Durch die Verwendung von Portnummern können mehrere Prozesse parallel auf ein Netz zugreifen, ohne dass es dabei zu einem Datenverlust kommt. Die Verbindung zwischen zwei Kommunikationsteilnehmern wird über das „socket“, eine Kombination aus der Portnummer und der IP-Adresse, geregelt. Um eine fehlerfreie Übermittlung von Telegrammen zu gewährleisten, wird beim Verbindungsaufbau, beim Datenverkehr und

beim Verbindungsende über ein Handshake-Verfahren mit Timeout-Überwachung nach Fehlern gesucht. Dieses System erweist sich als sehr kostengünstig, da in der Regel eine einfache Schaltung vorliegt. Lediglich der Client muss über eine gewisse Komplexität verfügen, da dieser den Bus steuern muss [9].

Ein ModbusTCP verfügt über vier unterschiedliche Adressbereiche. Jedem Adressbereich wird ein bestimmter Variablen-Typ zugewiesen. Jeder Adressbereich hat in Hinblick auf Lese– und Schreibzugriff jeweils unterschiedliche Eigenschaften.

Tabelle 1: Objekttypen des Modbus-Protokolls

Objekttyp	Objektname	Zugriff	Größe
Binärer Ausgang	Coil	Lesend/Schreibend	1-Bit
Binärer Eingang	Discrete Input	Lesend	1-Bit
Analoger Ausgang	Input Register	Lesend	16-Bit
Analoger Eingang	Holding Register	Lesend/Schreibend	16-Bit

Tabelle 1 gibt eine Übersicht über die Objekttypen innerhalb der Modbus-Kommunikation. Ein Modbus-Server kann jeweils über mehrere der Objekttypen aus Tabelle 1 verfügen. Der Speicherort ist dann beispielsweise „Holding Register (1)“. Die beiden binären Objekte verfügen über eine Speichergröße von einem Bit. Dabei handelt es sich um eine boolesche Variable, die entweder den Wert „True“ oder den Wert „False“ annehmen kann. Die analogen Objekte haben eine Speichergröße von 16 Bit und können dadurch in jedem Register einen Wert vom Typ „Integer“ speichern. Die Eingänge können jeweils nur Daten bereitstellen, die von einem Client gelesen werden. In den Ausgängen können Daten sowohl geschrieben als auch gelesen werden [10].

2.2 Programmiertechnischer Hintergrund

2.2.1 LabVIEW

LabVIEW ist ein grafisches Programmiersystem der Firma „National Instruments“. Die Abkürzung steht für **Lab**oratory **V**irtual **I**nstrumentation **E**ngineering **W**orkbench. Mit LabVIEW entwickelte Programme werden als Virtuelle Instrumente (VI) bezeichnet. Ein solches VI hat immer zwei Bestandteile. Zum einen das Frontpanel, es stellt die Benutzeroberfläche des Programms dar. Der zweite Teil ist das Blockdiagramm, welches grafisch den Programmablauf darstellt.

Das Blockdiagramm arbeitet ohne einen Interpreter und hat deswegen eine mit anderen Hochsprachen vergleichbare Leistung. Jedes LabVIEW Programm kann über mehrere

VIs verfügen. Solche VIs werden als SubVIs bezeichnet. Jedes VI verfügt über ein eigenes Frontpanel und Blockdiagramm. Die Logik innerhalb des Blockdiagramms wird mit Hilfe von virtuellen Drähten, die die jeweiligen Funktionen miteinander verknüpfen, dargestellt. Bei fast jeder Funktion aus der LabVIEW-Bibliothek handelt es sich um VIs, deren Funktionalität lässt sich auf wenige Grundoperationen zurückführen. Alle VIs haben bestimmte Eingangs- und/oder Ausgangswerte. Ein VI wird dann ausgeführt, wenn alle Eingangswerte anliegen. Die Abarbeitung der einzelnen VIs ergibt sich aus der Abhängigkeit der Eingangswerte zueinander. Wenn zwei Funktionen zum gleichen Zeitpunkt alle Eingangswerte haben, werden diese gleichzeitig ausgeführt [11]. Für jede Funktion lassen sich in LabVIEW die Eingangs- und Ausgangswerte einfach bestimmen. In der Reglerplattform stellen diese Werte in den meisten Fällen die Werte dar, die in dem Python-Programm an den Schnittstellen übergeben werden müssen. Der Programmfluss von einem LabVIEW Programm kann zusätzlich von einer Sequenzstruktur gesteuert werden.

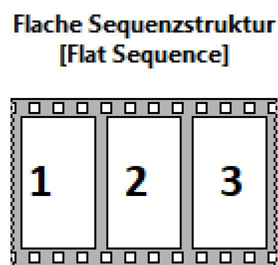

Abbildung 2-1: LabVIEW Sequenzstruktur

Abbildung 2-1zeigt diese Sequenzstruktur innerhalb von LabVIEW. In den Kästen befindet sich die Logik, die behandelt werden soll. Die Reihenfolge der Abarbeitung ist in der Abbildung durch die Zahlen dargestellt.

Die Aufgaben des Frontpanels werden in anderen Programmiersprachen von einer separat zu programmierenden GUI (Graphical User Interface) übernommen. Das Frontpanel ist damit die Schnittstelle zwischen Mensch und Programm. Mit wenig Aufwand lässt sich auf dem Frontpanel mittels Anzeigen und virtuellen Schaltern eine Benutzeroberfläche erstellen.

2.2.2 Python

Python zählt zu den höheren Programmiersprachen und wurde mit dem Ziel einer guten Verständlichkeit und eines kurzen Programmierstils entwickelt. Dies spiegelt sich in der

reduzierten Syntax von Python wieder, welche mit wenigen Schlüsselwörtern arbeitet. Der Quellcode wird durch Einrücken strukturiert und ersetzt damit Klammer, wie sie in anderen Programmiersprachen Anwendung finden. Die Skripte in Python sind aus diesem Grund in der Regel kürzer als in anderen Programmiersprachen [12]. Python verfügt über eine große größtenteils plattformunabhängige Standartbibliothek. Die Standartbibliothek verfügt, aufgrund des offenen Entwicklungsmodells, über viele, einfach zu implementierende Erweiterungen. Eine dieser Erweiterungen ist das, für die Reglerplattform verwendete, Modul zur Modbus-Kommunikation mittels TCP/IP.

Python unterstützt mehrere Programmierparadigmen. Dabei ist es nie zwingend nötig sich an eines der Programmierparadigmen zu halten. Sowohl objektorientierte als auch strukturierte Programmierung ist mit Python ohne Einschränkungen möglich.

2.2.3 Objektorientierte Programmierung

Das Ziel der objektorientierten Programmierung ist es, eine hohe Wiederverwendbarkeit des erzeugten Quellcodes zu erreichen. Der Erfinder des Begriffes objektorientiert, Alan C. Kay, definierte diesen mit den folgenden Unterpunkten [13].

1. Alles ist ein Objekt.
2. Objekte kommunizieren durch das Senden und Empfangen von Nachrichten.
3. Objekte haben ihren eigenen Speicher.
4. Jedes Objekt ist die Instanz einer Klasse.
5. Die Klasse beinhaltet das Verhalten aller Instanzen.
6. Um eine Programmliste auszuführen, wird die Ausführungskontrolle dem ersten Objekt gegeben und das Verbleibende als dessen Nachricht behandelt.

Objekte entwickeln sich aus Klassen. Klassen dienen als Baupläne für ein Objekt und repräsentieren damit auch den Objekttypen. Das Verhalten eines Objektes wird von den Methoden der Klasse bestimmt, aus der sich das Objekt entwickelt. Die Attribute der Klasse bestimmen die Methoden nach den das Objekt arbeitet. Eine Klasse ist immer eine Abstraktion, eine Vereinfachung, erst durch die Erzeugung eines Objektes aus der Klasse entsteht ein konkretes Abbild. Die Klassen werden in einer externen Datei gespeichert. Innerhalb eines Programmes wird die Klasse importiert und ein Objekt vom Typ der Klasse erstellt.

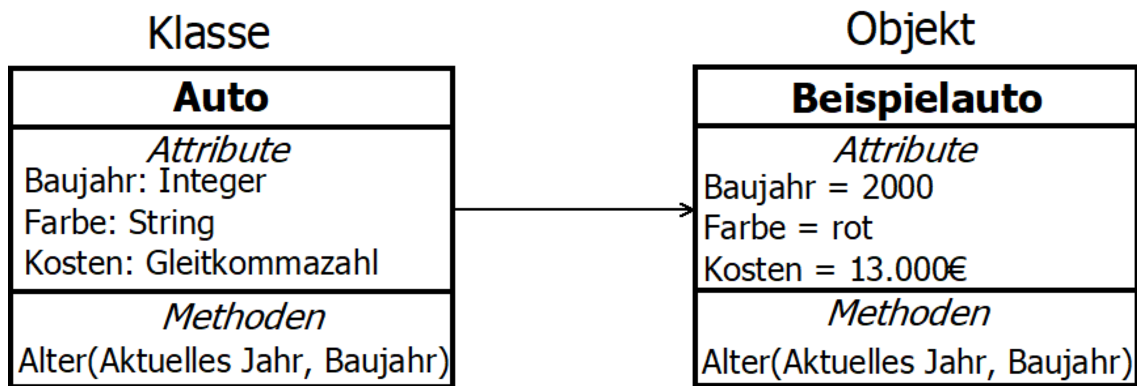

Abbildung 2-2: Beispiel für eine Klasse und ein abgeleitetes Objekt

In Abbildung 2-2 wird das Verhältnis zwischen einer Klasse und einem Objekt schematisch dargestellt. Auf der linken Seite ist die Klasse „Auto". Die Attribute der Klasse sind: Baujahr, Farbe und Kosten. Jedem Attribut ist ein Datentyp zugewiesen. Die Methode der Klasse ist das Alter des Autos und errechnet sich aus dem aktuellen Jahr und dem Baujahr. Auf der rechten Seite von Abbildung 2-2 ist ein aus dieser Klasse erzeugtes Objekt. In diesem wurden den Attributen konkrete Wert zugewiesen. Objekte kommunizieren untereinander nur über Nachrichten. Andere Kommunikationswege, wie zum Beispiel globale Variablen, gibt es in einer streng objektorientierten Programmierung nicht. [13]. Python gibt aber die Möglichkeit, von einem Programmierparadigma abzuweichen.

2.2.4 Unified Modeling Language (UML)

UML ist eine Modellierungssprache und dient zur Visualisierung und Dokumentation von Programmabläufen. Nachdem ein Programm mit UML dokumentiert wurde, kann anschließend aus den Diagrammen der Softwareaufbau in der gewünschten Sprache konstruiert werden. Dabei dient UML in erster Linie der grafischen Darstellung von einem Programm. UML wird immer wieder weiterentwickelt und ist aktuell bei der Version UML2.3. Aufgrund des einfachen logischen Aufbaus ist UML die dominierende Sprache bei der Softwaremodellierung [14].

Für die Dokumentation stehen einem Anwender unterschiedliche Diagrammtypen zur Verfügung. Diese lassen sich allgemein in die Kategorien „Strukturdiagramme" und „Verhaltensdiagramme", unterteilen. Jeder dieser Diagrammtypen hat sieben Untertypen, wodurch insgesamt vierzehn Diagrammtypen zusammenkommen. Die Grenzen zwischen den Diagrammtypen sind dabei fließend und seit der Version UML2 ist es explizit erlaubt, die Elemente aus anderen Diagrammtypen zu verwenden [14]. Dabei können Kombinationen der jeweiligen Diagrammtypen entstehen.

Das Aktivitätsdiagramm ist ein Verhaltensdiagramm und dient zur Darstellung von Datenflüssen. Dadurch werden verschiedenen Aktionen veranschaulicht. Der Anfang eines Aktivitätsdiagrammes wird mit einem ausgefüllten Kreis angezeigt. Von dem Kreis ausgehend entwickelt sich das Diagramm. Der Programmablauf ist mit Pfeilen dargestellt, die auf die Aktivitäten oder Verzweigungen zeigen. Aktivitäten werden durch Rechtecke mit abgerundeten Ecken dargestellt. Aktivitäten stehen für Aktionen in dem Programm, die aufgeführt werden, sobald der Programmablauf an der entsprechenden Stelle ist. Eine leere Raute, zeigt eine Verzweigung mit Bedingung. An den Pfeilen, die von der Raute wegzeigen, stehen die Bedingungen, die erfüllt sein müssen damit das Programmablauf in diese Richtung weiterführt. Ein schwarzer Balken kennzeichnet zwei Programmschritte die parallel abgearbeitet werden. Die Zusammenführung des Programmflusses ist auch durch einen schwarzen Balken oder eine Raute gekennzeichnet. Ein Kreis mit einem schwarzen Kern und einem konzentrischen Ring darum symbolisiert das Ende eines Aktivitätsdiagramms [14]. Die Pfeile, die zwei Module miteinander verbinden, stellen damit die Schnittstellen zwischen diesen dar.

2.2.5 ModbusPal

Um feststellen zu können, ob die Schnittstelle zwischen Plattform und den Modbus-Servern funktionstauglich ist, müssen die Server simuliert werden. Dafür dient die Modbus-Server-Simulation „ModbusPal". ModbusPal stellt eine einfach zu bedienende Benutzeroberfläche, mit der Möglichkeit eine komplexe und realistische Modbus-Umgebung zu simulieren. ModbusPal basiert auf der Programmiersprache Java. Mit der Java-Implementierung von Python (Jython) lassen sich Skripte für die Server-Simulation schreiben [15]. Für die Benutzung von ModbusPal ist es nicht zwangsläufig notwendig, die Anwendung mit Skripten zu erweitern. Grundlegende Funktionen stehen standardmäßig zur Verfügung. ModbusPal erzeugt, sobald ein Client mit der richtigen IP und dem richtigen Port einen Server anspricht, einen virtuellen Server, der über den angefragten Objekttypen verfügt. Im Fall der Reglerplattform wird ein Server mit Holding-Registern erzeugt. Über die Benutzeroberfläche können die Holding-Register angezeigt und die enthaltenden Werte verändert werden.

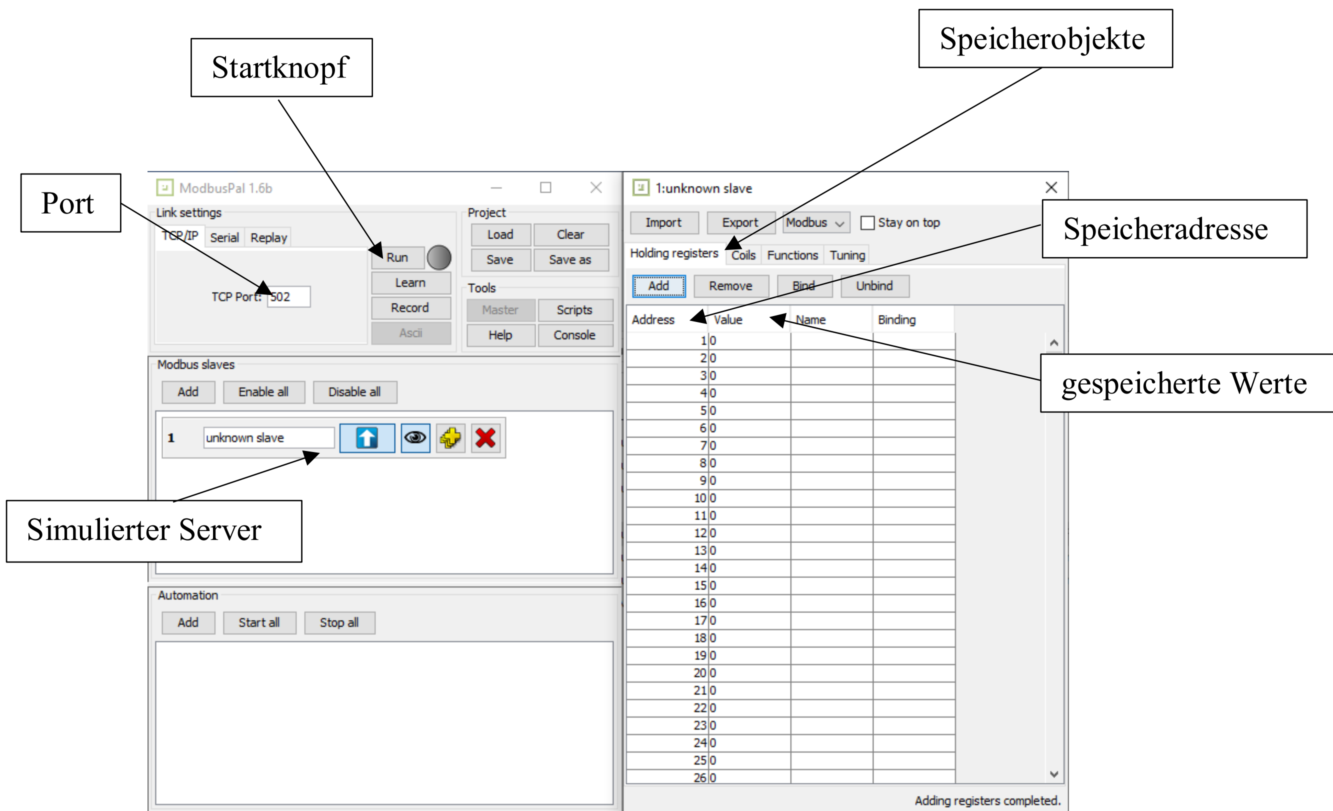

Abbildung 2-3: Oberfläche ModbusPal

Der linke Teil der Abbildung 2-3 stellt die Oberfläche der ModbusPal Anwendung dar. Die wesentlichen Bestandteile sind beschriftet. Ist zusätzlich zu dem Startknopf der Knopf „Learn" unter dem Startknopf gedrückt, erstellt das Programm automatisch einen Server, wenn im gleichen IP-Bereich eine Verbindung mit einem Server hergestellt werden soll. Der rechte Teil von Abbildung 2-3 zeigt die Daten des Servers. Die benötigten Speicher können erstellt und mit Werten versehen werden. Ist der „Learn" Knopf gedrückt, erstellt das Programm automatisch die Speicher, die gelesen und/oder geschrieben werden sollen [16].

2.2.6 Multithreading

Unter Multithreading versteht man das gleichzeitige Abarbeiteten von mehreren Threads. Ein Thread ist Ausführungsstrang innerhalb eines Prozesses. Bei der Reglerplattform kommen drei Threads zum Einsatz, wobei einer der drei, der Hauptthread, nur die beiden anderen startet. Einer der Threads steuert die GUI und der letzte führt die Modbus-Kommunikation und die Steuerungsfunktionen aus. Dies ermöglicht, dass die GUI unabhängig vom Rest der Reglerplattform ausgeführt werden kann. Die beiden Threads sind voneinander nicht abgeschottet, so dass beide Threads gleichzeitig auf dasselbe Objekt zugrei-

fen können. Dabei können Speicherfehler auftreten, die es zu vermeiden gilt. Um Speicherfehler zu vermeiden, wird das Objekt, auf das beide Threads Zugriff haben, so lange für einen Thread gesperrt, wie der andere Thread auf das Objekt zugreift. Der Thread, der keinen Zugriff auf das Objekt hat, läuft in der Zeit in der Regel weiter. Ist das Objekt auf das die Threads zugreifen ein Speicher gehen die Daten, die nicht in dem Objekt gespeichert werden können, verloren, solange der Thread keinen Zugriff hat.

Eine Möglichkeit, diese Zugriffsfehler zu vermeiden und trotzdem keinen Datenverlust zu haben, ist die Anwendung von Puffern. Dabei werden die einzelnen Zeitschritte zwischengespeichert. In dem Puffer können dann mehrere Zeitschritte gespeichert sein. Solange der gemeinsame Speicher für einen der Threads gesperrt ist, können die Daten in dem Puffer gespeichert werden.

2.3 Softwarevalidierung

Bei der Softwareentwicklung sind Softwaretests ein wichtiger Bestandteil. Durch Tests werden die Funktionalität, die Leistung und die Benutzerfreundlichkeit eines Programmes geprüft. Das Testen garantiert dabei nicht, dass es nicht mehr zu Fehlern kommen kann, vielmehr wird nachgewiesen, wie sicher man bezüglich der oben genannten Kategorien sein kann. Man unterscheidet zwischen zwei Typen von Tests: Funktionstest und nicht-funktionale Tests. [17]

2.3.1 Funktionstests

Funktionstest überprüfen Benutzerfreundlichkeit und geschäftskritische Funktionen. Durch diese Tests wird sichergestellt, dass die Softwarefunktionen wie erwartet und ohne Fehler ablaufen. Zu den Funktionstests zählen: Komponententests und Integrationstests [17].

Komponententests

Bei Komponententests werden die einzelnen Bausteine des Programmes getestet. Jeder Bestandteil eines Programmes, ob Funktion, Modul oder Methode, muss separat einer Komponentenprüfung unterzogen werden. Dadurch wird die Richtigkeit des zu erwartenden Verhaltens bewiesen [17].

Integrationstests

Bei Integrationstests werden mehrere Module in einer Softwaregruppe zusammen getestet. Diese Gruppe besteht aus Modulen, die für bestimmte Funktionalitäten zusammenarbeiten. Mit diesem Test wird das Zusammenspiel verschiedener Module validiert und damit im Zusammenhang stehende Fehler und Probleme werden identifiziert [17].

2.3.2 Nicht-funktionale Tests

Der Unterschied zu Funktionstests besteht bei nicht-funktionalen Tests darin, dass die Funktionen unter Last durchgeführt werden. Dadurch werden Geschwindigkeit, Stabilität, Benutzerfreundlichkeit und Skalierbarkeit der Software getestet [17]. Derartige Tests sind für die Plattform nicht geplant, da es in erster Linie um eine softwaretechnische Umsetzung ging und nicht um die Demonstration im Feld oder in Echtzeit.

3 Darstellung des LabVIEW-Programms

3.1 Herangehensweise

Als Vorlage für das Python-Programm dient die in LabVIEW vorhandene Reglerplattform. Die Funktionalität der LabVIEW-Plattform wird in dem ersten Schritt mittels eines UML-Aktivitätsdiagrammes dargestellt. Die Notation des Aktivitätsdiagrammes für die Reglerplattform orientiert sich an UML1. Nachdem die Funktionen der LabVIEW-Plattform dokumentiert sind, wird bestimmt, anhand der Grundlegenden Funktionen der Plattform und durch die Einhaltung des Zeitrahmens wird festgelegt, welche Module umgesetzt werden.

Ist das UML-Diagramm von dem LabVIEW-Programm erstellt und sind die umzusetzenden Module bestimmt, werden für die einzelnen Bausteine des Python-Programms ebenfalls UML-Diagramme erstellt. Eine Ausnahme stellt hier die GUI dar, hier wird kein UML-Diagramm erstellt, da diese sich für den Aufbau einer GUI nicht eignen. Die GUI führt, mit Ausnahme von rotem oder grünem Hintergrund, keine eigene Logik aus, die es zu dokumentieren gäbe. Aus den entstandenen UML-Diagrammen kann dann der Python-Code entwickelt werden.

3.2 Beschreibung der LabVIEW-Plattform

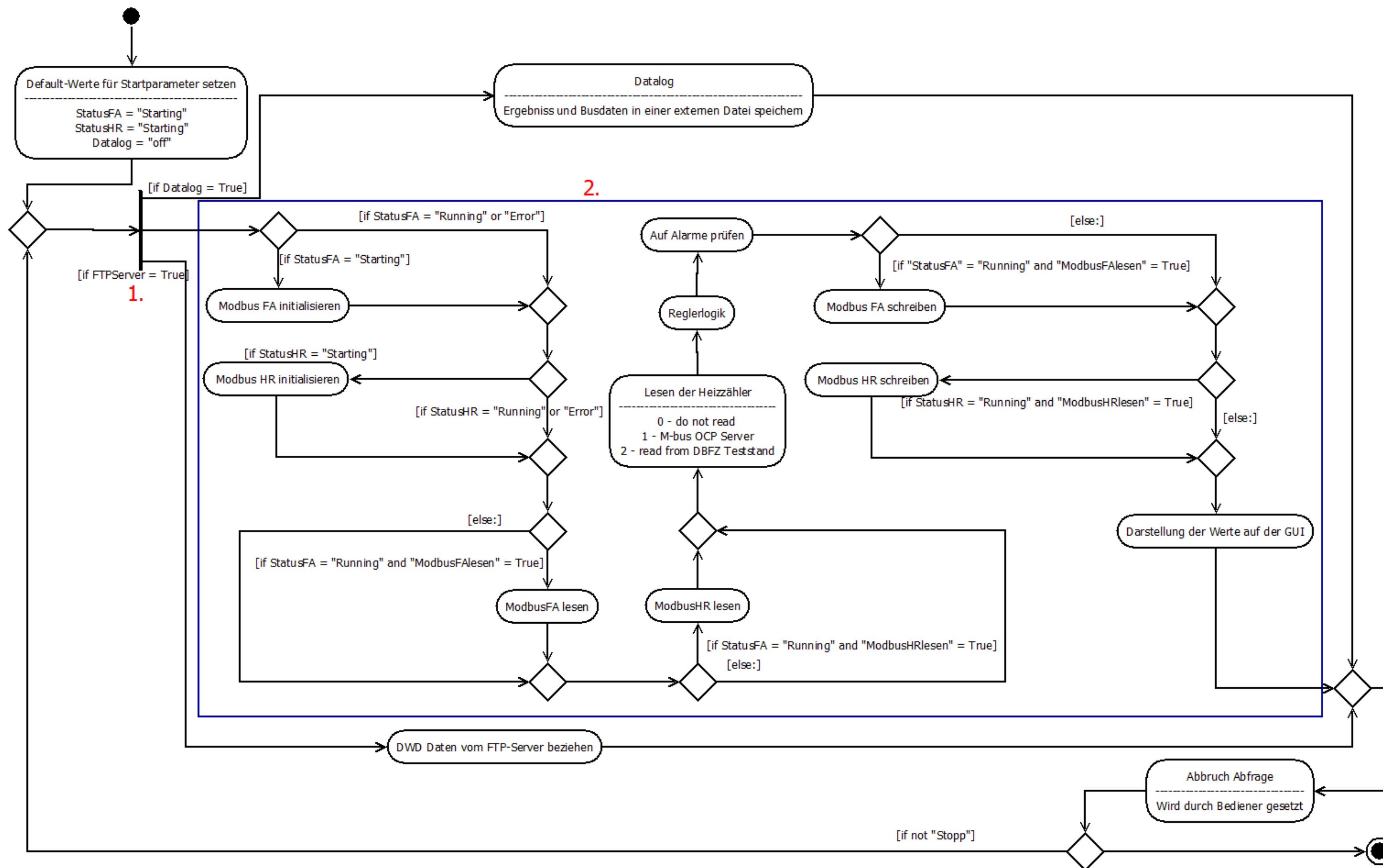

Abbildung 3-1: UML-Diagramm: LabVIEW-Programm Reglerlplattform

In Abbildung 3-1 ist die Funktionalität der in LabVIEW vorhandenen Reglerplattform abgebildet. Die Verzweigung bei (1.) gibt an, dass die drei von dem senkrechten Balken abgehenden Pfeile gleichzeitig durchgeführt werden. Für die obere und die untere Anweisung gilt dies nur, wenn die entsprechende Bedingung erfüllt ist. Die obere Aktivität ‚Datalog‘ wird über das Frontpanel aktiviert. So lange diese Aktivität ausgeführt wird, werden die Ergebnisse der Reglerfunktion in einer externen Datei gespeichert. Die untere Aktivität öffnet eine Verbindung zu einem FTP-Server des Deutschen Wetterdiensts. Von dem FTP-Server werden aktuelle Wetterdaten bezogen. Das Nutzen der Wetterprognose soll in Zukunft zu einer Verbesserung des Steuerungsalgorithmus führen. Zum Zeitpunkt dieser Arbeit war die Entwicklung des Reglers nicht so weit, dass diese Werte verarbeitet werden konnten. Die Funktionen innerhalb des blauen Rahmens (2.) beschreiben die eigentliche Funktionalität der Reglerplattform.

Nachfolgend wird die Funktionalität innerhalb des blauen Rahmens beschrieben. In dem ersten Schritt wird, wenn „StatusFA“ den Wert „Starting“ hat, der ModbusFA initialisiert. Die zum Initialisieren notwendigen Werte wie die Bus-IP und der Bus-Port werden über das Frontpanel entgegengenommen. Hat der Modbus hingegen den Status „Running“ oder „Error“, wird dieser Schritt übersprungen. Die gleiche Aktivität wird anschließend für den ModbusHR unter Beachtung des Wertes von StatusHR aufgeführt. In dem nächsten Schritt wird, beginnend mit ModbusFA geprüft, ob die beiden Modbusse jeweils Werte von dem Server lesen sollen. Die Modbusse können nur dann lesen, wenn der Status des jeweiligen Modbusses den Wert „Running“ hat. Ist dies der Fall, werden von den Clients FA und HR die entsprechend Holding Registers ausgelesen und das Ergebnis in dem SubVI „GlobalDataArray“ gespeichert. Soll keiner oder nur einer der Modbusse Werte entgegennehmen, kann diese Funktion über das Frontpanel entsprechend geschalten werden. Im nächsten Schritt wird der Heizzähler, entsprechend dem dazugehörigen Parameter, abgearbeitet. Hat der Parameter den Wert „0“ werden die Heizungszähler nicht ausgelesen. Bei einer „1“ werden die Heizzähler von einem M-Bus gelesen und bei einer „2“ werden die Werte von aus dem Teststand des DBFZ gelesen. Anschließend wird die in der SubVI DBFZ-Regler hinterlegte Reglerlogik aufgeführt. Diese SubVI beinhaltet auch das Lesen von manuellen Werten und den Bypass. Bei letzterem werden die Werte unverarbeitet weitergegeben. Über eine Schnittstelle werden die für die Regelung benötigten Daten aus dem „GlobalDataArray“ gelesen und entsprechen der Logik verarbeitet. Nachdem die Reglerlogik beendet ist, wird das Regelergebnis in dem „GlobalDataArray“ gespeichert. Nach der Reglerlogik wird die Plattform auf Alarme geprüft.

Sollte in den vorhergehenden Schritten ein Alarm aufgetreten sein, wird dieser unter Verwendung eines Fehlercodes in einem „Error Array“ gespeichert. Wird hier beispielsweise ein Alarm in den Bussen festgestellt, ändert sich der Status der beiden Modbusse auf „Error“. Die Clients können dann weder lesen noch schreiben. In den nächsten Schritten wird geprüft, ob die beiden Modbusse das Ergebnis der Regelung wieder in die Holding Register des Servers schreiben sollen. Dies ist dann der Fall, wenn der Status des jeweiligen Modbusses „Running“ ist und die Variable „WriteModbusFA“ bzw. „WriteModbusHR“ durch das Frontpanel den Wert „True“ hat. Andererseits wird die jeweilige Aktivität übersprungen. Die letzte Funktion innerhalb des blauen Rahmens übergibt die relevanten Daten an das Frontpanel, auf dem diese angezeigt werden. Wird durch das Frontpanel kein Abbruchbefehl gegeben, läuft die zuvor beschriebene Logik in einer Schleife immer wieder durch.

3.3 Beschreibung der grafischen Oberfläche des LabVIEW-Programmes

Das Frontpanel, die grafische Oberfläche der LabVIEW-Anwendung, verfügt über sechs Reiter Tabelle 2 gibt eine Übersicht der Reiter des LabVIEW Programmes und deren Funktionen. Zusätzlich zu den Reitern hat die grafische Oberfläche eine Fußzeile. Nachfolgend werden die Reiter einzeln gezeigt und deren Funktion erklärt. Unter dem ersten Bild wird zusätzlich die Fußzeile erläutert.

Tabelle 2: Übersicht der Reiter des LabVIEW Programmes

Reiter (Tab)	**Funktion**
Heating System	Anzeige der Modbus Daten
Heat meter	Anzeige der Heizdaten
Settings	Einstellungen für Server-Verbindungen und den Regler
DWD data	Anzeige der Daten des deutschen Wetterdients
Data logging	Start und Stopp des Datenloggings
Charts	Anzeige von Graphen

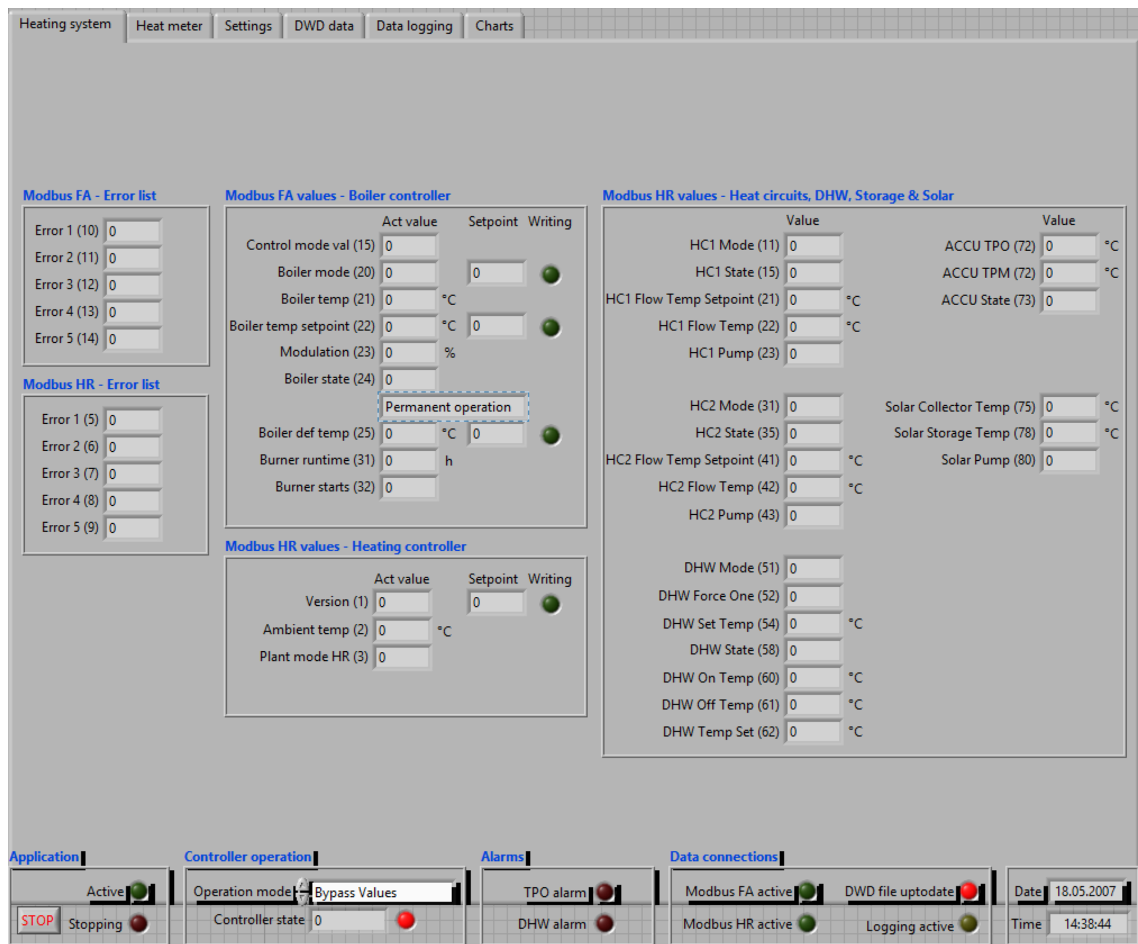

Abbildung 3-2: LabVIEW-GUI: Reiter „Heating system“

In den Reitern sind alle zusammenhängenden Eingabefelder mit einem Rahmen verbunden und mit einer Überschrift versehen. In dem unteren Bereich von Abbildung 3-2 ist die Fußzeile des LabVIEW-Programmes zu sehen. Hier sind die Status der wesentlichen Funktionen der Plattform auf einen Blick zu sehen. Rotes Licht bedeutet „inaktiv“ und grünes Licht „aktiv“. Weiterhin kann der Ablauf der Funktion der Plattform gestoppt und der Betriebsmodus des Reglers gewählt werden.

Abbildung 3-2 zeigt den Reiter „Heating Systems“. Die Zahlen in der Klammer nach dem Variablennamen beschreiben die Position des Wertes in dem Holding Registern der Modbus-Server. Die Spalten mit der Überschrift „Act Value“ zeigt den aktuellen Wert der jeweiligen Variable an. „Setpoint“ ist der Wert aus dem vorhergehenden Zeitschritt. Ist der aktuelle Werte identisch mit dem Wert aus dem vorhergehenden Zeitschritt leuchtet die Anzeige bei „Writing“ grün. Bei abweichenden Werten leuchtet die Anzeige rot. Die Einheiten der Werte sind hinter den jeweiligen Feldern angegeben. Ist keine Einheit angegeben, handelt es sich um einen einheitenlosen Wert. Dabei handelt es sich in der Regel um unterschiedliche Modi.

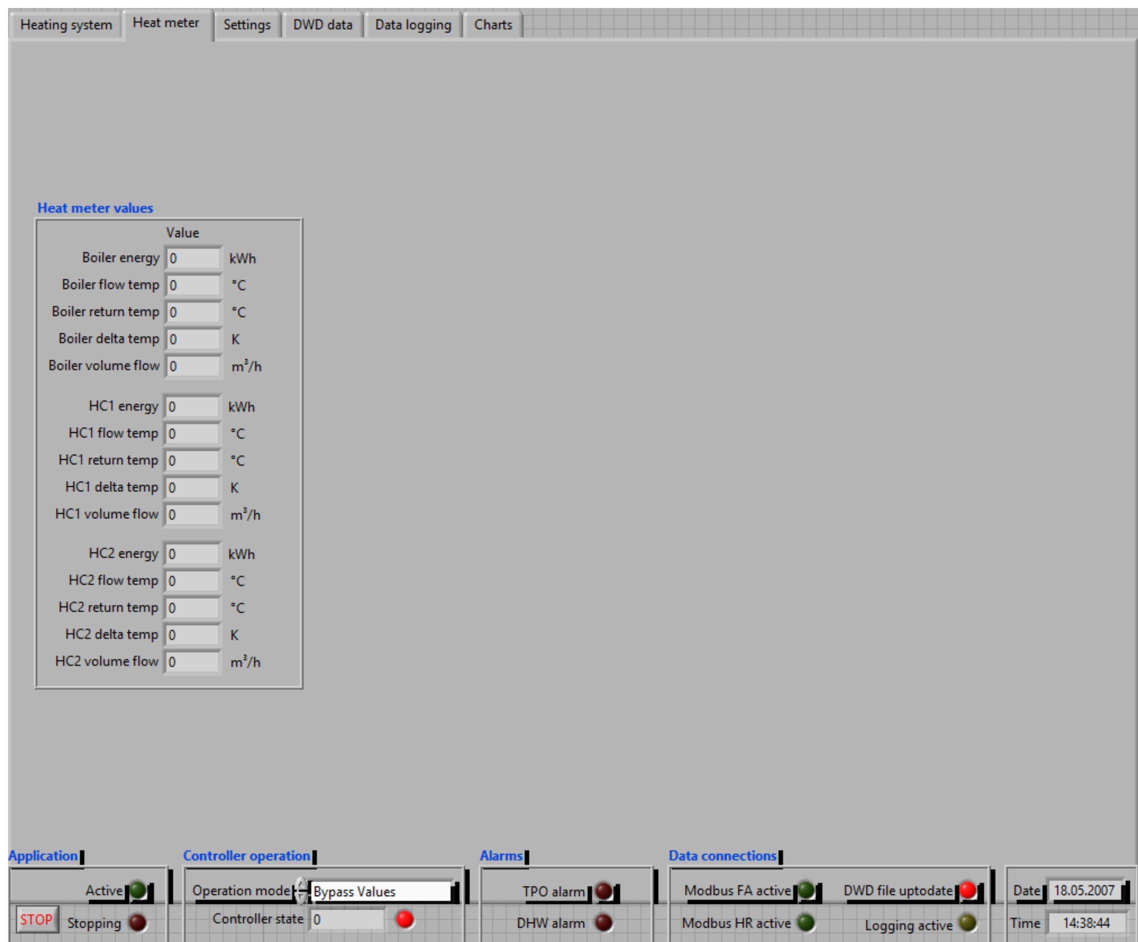

Abbildung 3-3: LabVIEW-GUI: Reiter "Heat meter"

Abbildung 3-3 zeigt im oberen Bereich Daten zum Heizkessel. In der Mitte sind die Daten des ersten und unten die Daten des zweiten Heizkreises. Diese werden von einem separaten M-Bus eingelesen und in dem „GlobalDataArray“ zwischengespeichert. Die hier angezeigten Werte werden dabei für die Reglerlogik nicht benötigt und dienen in erster Linie der Datenerfassung. Diese Daten könnten dazu genutzt werden, den Regelalgorithmus zu optimieren oder Unstimmigkeiten im System zu lokalisieren.

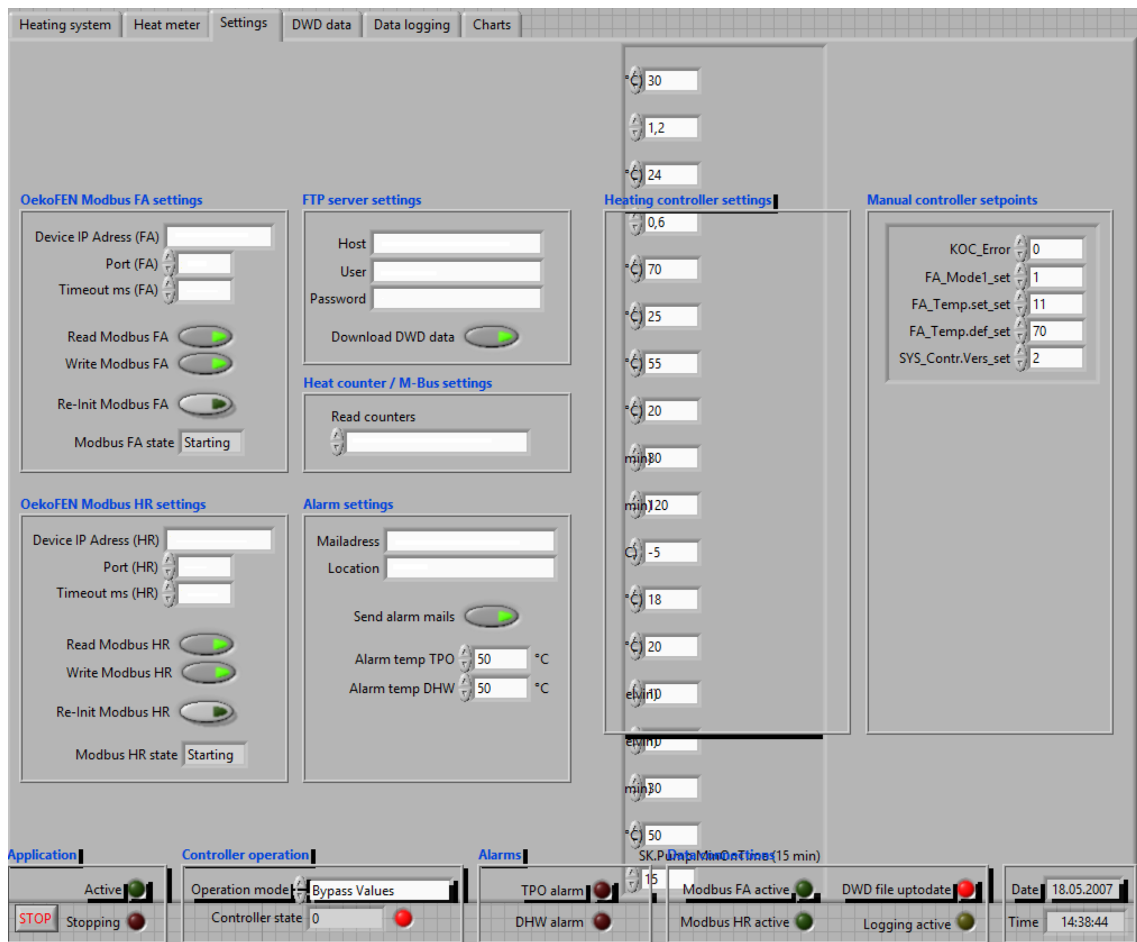

Abbildung 3-4: LabVIEW-GUI: Reiter "Settings"

Abbildung 3-4 zeigt den Reiter „Settings“. Auf der linken Seite des Reiters sind die für den ModbusTCP wichtigen Bedienfelder. Die beiden Felder sind identisch aufgebaut. Über die Eingabefelder werden IP-Adresse, Port und die Zeit für die Timeout-Abfrage entgegengenommen. Mit den Knöpfen darunter werden der lesende Zugriff (Read Modbus) und der schreibende Zugriff (Write Modbus) eingeschalten. Mit dem Knopf (Re-Init Modbus) wird der Modbus-Client neu initialisiert. In dem letzten Feld wird der aktuelle Status des Modbus-Clients angegeben. Unter „FTP server settings“ befinden sich die Eingabefelder für die FTP-Verbindung mit dem Deutschen Wetterdienst. Zusätzlich befindet sich dort noch ein Knopf, mit dem die Verbindung ein- bzw. ausgeschalten werden kann. Bei dem Feld „Heat counter / M-Bus settings“ kann ausgewählt werden, von welchem System der Heizzähler ausgelesen wird. Zur Auswahl stehen zur Zeit der im Untersuchungsgegenstand des Projekts „KombiOpt“ verwendete M-Bus oder der Teststand des DBFZs. Bei „Alarm settings“, kann eingegeben werden, an welche Email-Adresse die Fehlermeldungen geschickt werden sollen. In dem Feld „Location“ wird der Name der Anlage eingegeben, damit die Fehlermeldungen zugewiesen werden können. In den beiden Eingabefeldern können die Temperaturen eingegeben werden, bei denen

eine Fehlermeldung erfolgen soll. Im nächsten Abschnitt (Heating controller settings) werden die Einstellungen für den Regler entgegengenommen. Hier ist die Formatierung fehlerhaft. Grund dafür ist, dass im Laufe der Entwicklung der Plattform die Anzahl der Regler-Einstellungen gestiegen ist, weshalb diese nicht mehr in den Rahmen passen. Da dies die Funktion nicht eingeschränkt, ist es sinnvoller, dies erst bei der fertig entwickelten Plattform anzupassen. Auf der rechten Seite befinden sich unter „manual controller settings“ die Eingabefelder für die Werte des manuellen Betriebs.

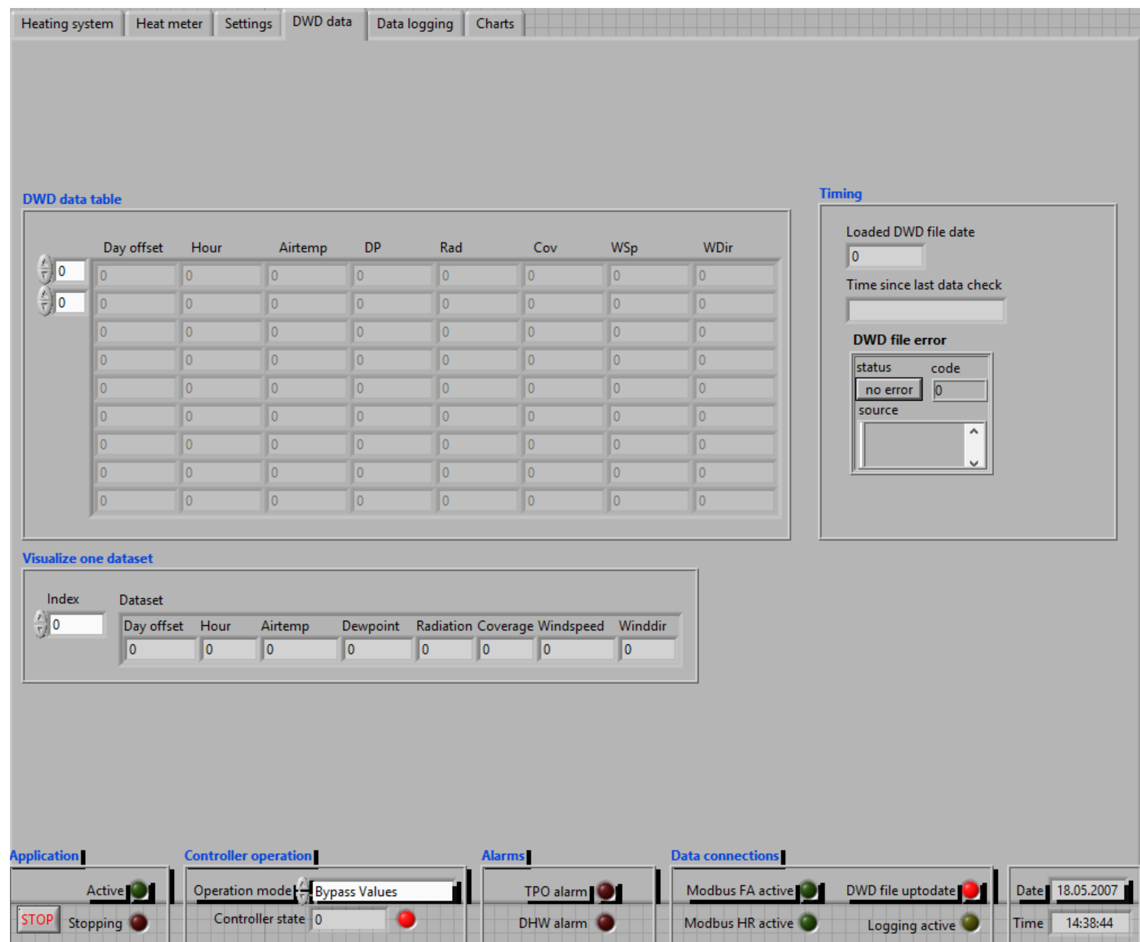

Abbildung 3-5: LabVIEW-GUI: Reiter "DWD Data"

Der Reiter „DWD data“ in Abbildung 3-5 gibt eine Übersicht über die Verbindung mit dem Deutschen Wetterdienst. In der Tabelle oben links werden die aktuellen Datensätze angezeigt. Darunter kann ein bestimmter Datensatz visualisiert werden. Auf der rechten Seite befindet sich eine Übersicht darüber, wie viele Datensätze geladen sind, wann zuletzt nach Datensätzen gesucht wurde und ob es einen Fehler bei der Verbindung gibt.

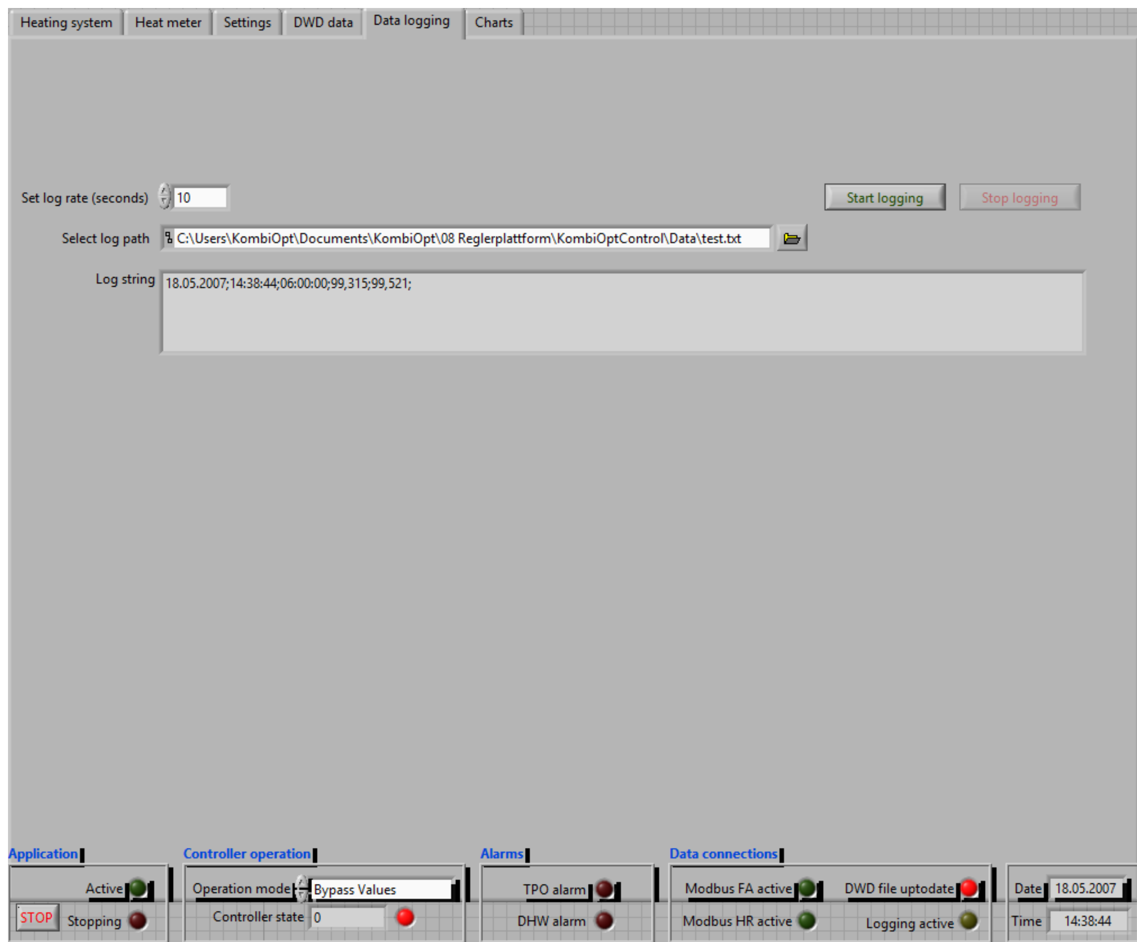

Abbildung 3-6: LabVIEW-GUI: Reiter "Data logging"

In Abbildung 3-6 ist der Reiter „Data logging“ zu sehen. Der Reiter dient dazu, die von der Plattform ermittelten Daten in einer externen Datei zu speichern. In dem Eingabefeld „Set log rate“ wird die Schrittweite der aufzuzeichnenden Zeitschritte angegeben. In dem Feld „Select log path“ wird der Speicherort Datei angegeben. „Log string“ zeigt das Format eines Zeitschritts an, dies kann nicht beeinflusst werden. Die Knöpf oben rechts starten bzw. stoppen das Speichern in einer externen Datei.

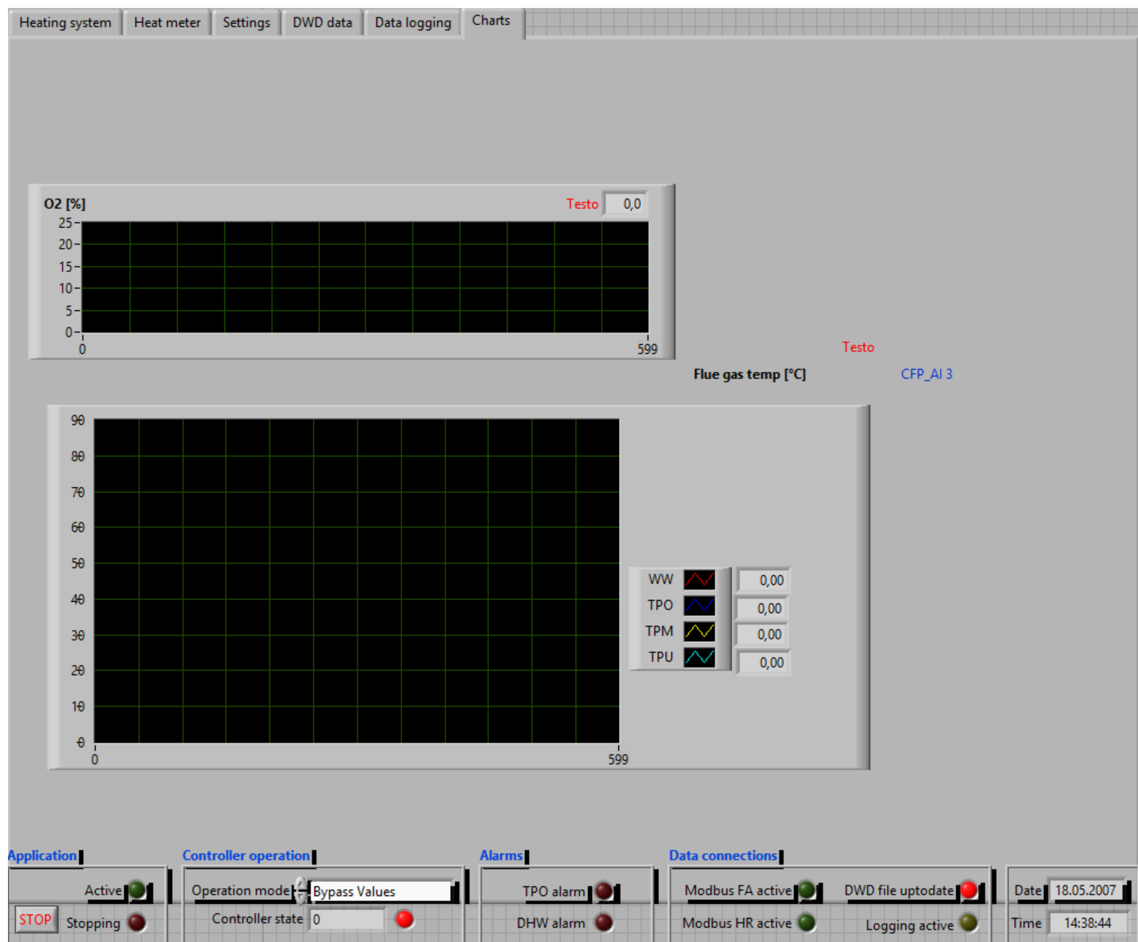

Abbildung 3-7: LabVIEW-GUI: Reiter "Charts"

Mit dem letzte Reiter in Abbildung 3-7 lassen sich verschiedene Graphen anzeigen. Der obere Graph zeigt das Verhältnis zwischen dem Sauerstoffgehalt im Abgas und der Temperatur im Kamin. In dem unteren Graph lassen sich die Warmwassertemperatur (WW) sowie die obere (TPO), die mittlere (TPM) und die untere Pufferspeichertemperatur (TPU) im Zeitverlauf anzeigen.

4 Python-Programm

4.1 Programmierparadigma

Wie in Kapitel 2.2.2 beschrieben, unterstützt Python mehrere Programmierparadigmen. Bei der Erstellung der Reglerplattform wird weitestgehend auf einen objektorientierten Ansatz gesetzt. Dadurch kann viel Code in externen Dateien gespeichert werden. Innerhalb des Threads wird nur das Objekt erstellt und aufgerufen, welches für den Programmablauf benötigt wird. In Python muss dafür keine streng objektorientierte Herangehensweise gewählt werden. Da objektorientierte Programmiersprachen weit verbreitet sind und sich gut mit dem modularen Aufbau kombinieren lässt, ist es sinnvoll, sich an diesen zu orientieren. Dadurch kann der Programmaufbau schnell erkannt und nachvollzogen werden. Da der Regler, der im Rahmen der vorhergehenden Arbeit erstellt wurde, keinen objektorientierten Aufbau hat, bildet dieser eine Ausnahme. Aufgrund des modularen Aufbaus, kann die Datei des Reglers, unter Beachtung der jeweiligen Schnittstellen, einfach ausgetauscht werden.

4.2 Anforderungen an die Reglerplattform

Das Ziel dieser Arbeit ist die Erstellung einer Python-basierten Reglerplattform in einer ersten Version, die die Grundlage für eine Weiterentwicklung stellt. In einer dieser Arbeit vorrausgehenden Praktikumsarbeit wurde der für die Regelung notwendige Heizungsregler in Python erstellt und getestet. Die Reglerplattform, die in dieser Arbeit erstellt wird, besteht aus mehreren Modulen, die über klar definierte Schnittstellen miteinander kommunizieren. Dadurch soll es später möglich sein, einzelne Komponenten aus der Plattform auszutauschen, ohne dass die Funktionalität der gesamten Plattform beeinträchtigt wird. Dieser modulare Aufbau sorgt dafür, dass die Plattform für jeden Anwendungsfall individuell angepasst werden kann. Dafür muss dann kein neues Programm geschrieben werden, sondern es können (vergleichbar mit einem Bausatz) die Module wiederverwendet werden, die benötigt werden. Das zentrale Modul dieser Reglerplattform ist das Graphical User Interface (GUI). Ausgehend davon werden die anderen Module der Plattform angesteuert. Aus zeitlichen Gründen wurden zuerst die wichtigsten Bausteine umgesetzt. Diese Bausteine sollen ein funktionsfähiges Programm ergeben, das Werte von einem Modbus-Server beziehen und verarbeiten und das Ergebnis wieder auf den Bus schreiben kann. Weiterhin soll die Möglichkeit bestehen, das Ergebnis der Steuerung bei Bedarf in einer externen Datei zu speichern. Die wesentlichen Ergebnisse des Regelungs-/Steuerungsbetriebs sowie der Status der Modbus-Kommunikation sollen auf der grafischen

Oberfläche angezeigt werden. Die grafische Oberfläche soll alle Reglerparameter und Modbus-Einstellungen an die Regler-Modbus-Schnittstelle übergeben können. Die Steuerung des Regelablaufs wird mittels entsprechenden Variablen ebenfalls über die grafische Oberfläche erfolgen. Die Reglerplattform soll mehrere Betriebsmodi unterstützen. Der erste Modus ist ein Bypass. Dabei werden die Werte unverarbeitet ausgegeben. Die Funktion Bypass dient in erster Linie zur Starten und Testen der Plattform. Der manuelle Modus gibt die Möglichkeit, vorgegebene Werte auf den Modbus-Server zu schreiben, auch in diesem Modus wird keine Steuerungslogik ausgeführt. Der letzte Betriebsmodus ist die Steuerungsfunktion des Reglers. Hier werden Werte entgegengenommen, verarbeitet und ausgegeben. Die fertige Reglerplattform bedient zwei Heizkreise mit einem gemeinsamen Pufferspeicher, der über einen Speicherbehälter verfügt, eine Feuerungsanlage, die zusätzlich von einer Solarthermieanlage unterstützt wird und einen Warmwasserkreislauf.

4.3 Unterschiede zwischen der Python- und LabVIEW-Version

Die Verbindung mit dem FTP-Server des Deutschen Wetterdiensts wurde in dieser Version nicht umgesetzt. Auf Grundlage dieser Daten soll später die Effizienz der Anlage weiter gesteigert werden, indem eine Steuerung anhand aktueller Wetterdaten erfolgt. Da dieser Steuerungsansatz noch in der Entwicklung ist, wird dieser Baustein in der Python-Plattform nicht umgesetzt.

Weiterhin wurden die Error-Meldungen weggelassen. Diese dienen zum einen der Stabilität des Programmes und zum anderen dienen sie zur Lokalisierung von Fehlern. Diese benötigen einerseits alle Bausteine, damit für jeden Fehler der dazugehörige „Error“ angezeigt werden kann. Andererseits wird das Anzeigen und Auswerten von Fehlern erst in einem Feldversuch benötigt. Ein Feldversuch wird mit dieser Version der Plattform jedoch nicht durchgeführt wird. Da die Fehler nicht durch die Error-Meldungen aufgefangen werden, führt ein Fehler in einem der beiden Threads immer dazu, dass der jeweilige Thread beendet wird.

Bei der grafischen Oberfläche wurden nur drei von den in Kapitel 3.3 beschriebenen sechs Reitern des LabVIEW-Programmes umgesetzt. Die anderen Reiter wurden nicht umgesetzt, weil die aktuelle Version der Python-Plattform für die Reiter noch keine Werte ausgibt. Die Reiter zeigen die grundlegende Funktionalität der Reglerplattform (siehe Kapitel 4.2).

4.4 Programmaufbau

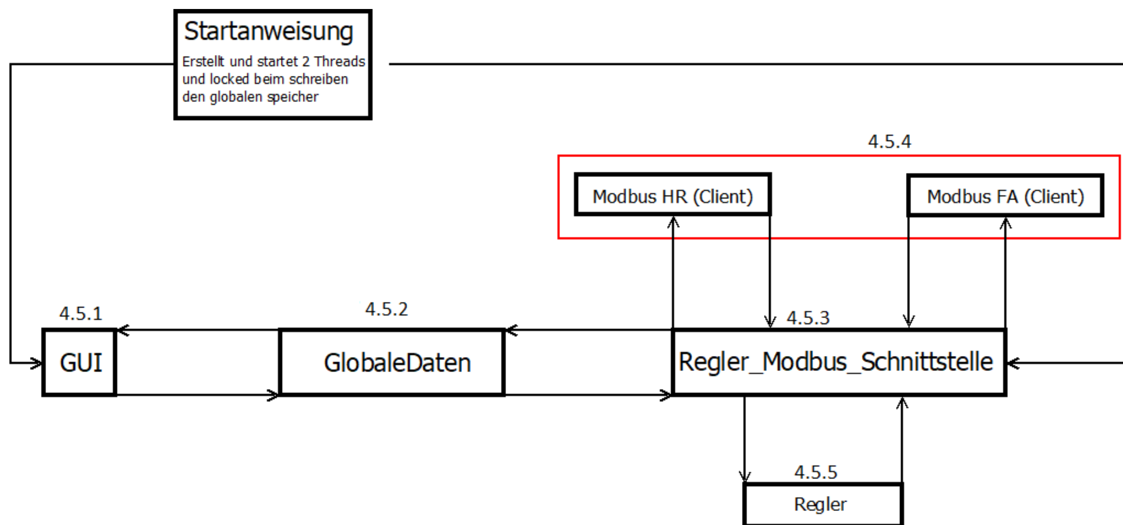

Abbildung 4-1: Wirkungsschema des Python-Programms

Das Programm startet durch das Modul Reglerplattform. Dieses Modul erzeugt zwei Threads die GUI und die Regler-Modbus-Schnittstelle. Die GUI ist die grafische Oberfläche des Programmes. Darin können sowohl Werte angezeigt und entgegengenommen als auch verschiedene Einstellungen an dem Steuerungsablauf vorgenommen werden. Die Regler-Modbus-Schnittstelle erstellt einerseits die Modbus-Kommunikation und andererseits führt sie die Steuerungslogik aus. Die Modbus-Kommunikation wird dabei von den Modulen ModbusHR und ModbusFA bereitgestellt. Die Steuerungslogik wird von dem Regler übernommen. Beide Threads werden mit Start des Programms gestartet. Die Regler-Modbus-Schnittstelle wird über eine Schleife gesteuert. Diese Schleife wiederum kann über die GUI ein- und ausgeschalten werden. Die Kommunikation zwischen den beiden Threads läuft über einen globalen Speicher.

4.5 Umsetzung der Bausteine der Reglerplattform

Der Regler, der in dieser Reglerplattform verwendet wird, wurde während des Praktikums erstellt [12]. Der Regler war so programmiert, dass dieser in einer Schleife Werte zyklisch einliest. Damit wurde die Schnittstelle zur Regler-Plattform simuliert, welche in jedem Zyklus Werte übergibt. Diese Schleife wird aus dem Regler-Programm entfernt und das Programm so abgeändert, dass Werte von der Regler-Modbus-Schnittstelle entgegengenommen werden.

Die für die Python-Reglerplattform verwendeten Module lassen sich in zwei Kategorien einteilen. Zur ersten Kategorie gehören die Module aus der Standartbibliothek und im-

portierte Module, wie zum Beispiel pyModbusTCP. Diese Module werden für den Programmablauf installiert. Die der Module aus der ersten Kategorie werden in der zweiten Kategorie, den selbst erstellten Modulen, aufgerufen. Bei den selbst erstellten Modulen handelt es sich um die Module, die die einzelnen Elemente der Reglerplattform darstellen.

Tabelle 3: Übersicht der importierten Module und deren Einbindung in den erstellten Modulen

Importierte Module / **Erstellte Module**	**Time**	**Date-time**	**pyMod-busTCP**	**Thread-ing**	**TKinter**
Regler	X	X			
Regler-Modbus-Schnittstelle	X				
GlobaleDaten					
GUI	X				X
ModbusFA			X		
ModbusHR			X		
Reglerplattform	X			X	X

In der Kopfzeile von Tabelle 3 sind die importieren Module dargestellt. In der ersten Spalte sind die erstellten Module aufgezählt. Die Tabelle zeigt welche importieren Module in welchem erstellten Modul verwendet werden. Nachfolgend wird der Nutzen der einzelnen importieren Module kurz erläutert.

- **Time**
  Mit Hilfe dieser Bibliothek können Informationen über die aktuelle Uhrzeit abgerufen werden. Die Bestimmung der aktuellen Zeit erfolgt in Stunden, Minuten und Sekunden.
- **Datetime**
  Die Bibliothek Datetime dient in dem Python-Regler zur Bestimmung von Zeitdifferenzen. Anwendung findet dieses Modul in der Steuerungslogik.
- **TKinter**
  TKinter ist das GUI Toolkit. Der Name steht für „**T**ool**K**it **I**nterface“. Mit Hilfe dieses Moduls kann die GUI mit allen nötigen Gadgets erstellt werden.
- **Threading**
  Mit dem Modul Threading können mehrere Threads erstellt werden, die entsprechend der Logik parallel ablaufen.
- **pyModbusTCP**

pyModbusTCP gibt die Möglichkeit, mittels Python ModbusTCP Server und Clients zu erstellen. Bei dem Modul handelt es sich um eine frei verfügbare Erweiterung zu der Standartbibliothek von Python. Dieses Modul muss, wenn die Reglerplattform gestartet wird, zusätzlich zu Python installiert sein.

4.5.1 Graphical User Interface (GUI)

Die GUI übernimmt alle Steuerungsaufgaben innerhalb der Reglerplattform. Dabei kommuniziert die GUI nur über den globalen Speicher mit der Regler-Modbus-Schnittstelle, die die Steuerung ausführt. Die GUI hat eine Schrittweite von $t = 2$ Sekunden. Dieser Wert ist als Standardwert zu betrachten und kann bei Bedarf direkt im Quellcode angepasst werden. Von den in Kapitel 3.3 aufgelisteten Reitern wurden „Settings“ (Abbildung 3-4), „Heating System“ (Abbildung 3-2) und „Data logging“ (Abbildung 3-6) umgesetzt. In dem Tab „Heating System“ können mehr Werte ausgelesen werden als der aktuelle Stand der Reglerplattform benötigt. Beispielsweise gibt es in der GUI Felder für die Fehlermeldungen der Busse. Die Fehler werden in der LabVIEW-Plattform in dem „GlobalDataArray“ gespeichert. Die in Python hinterlegte Liste hat den gleichen Umfang wie das Array in LabVIEW, damit gegebenenfalls in einem weiterführenden Projekt zusätzliche Informationen verarbeitet werden können. Zum jetzigen Zeitpunkt haben die Anzeigefelder keine Funktion und werden deswegen unabhängig von dem Steuerungsbetrieb den Wert „0“ anzeigen. Bei dem Reiter „Settings“ wurden die Einstellungen für den M-Bus, die FTP-Verbindung und die Sendung der Fehlermeldungen an die E-Mail nicht umgesetzt, da die Funktionen nicht Teil des Python-Programmes sind. Ansonsten ist die GUI durch umfangreiche Beschriftungen und einer Orientierung anhand der LabVIEW-GUI selbsterklärend gestaltet.

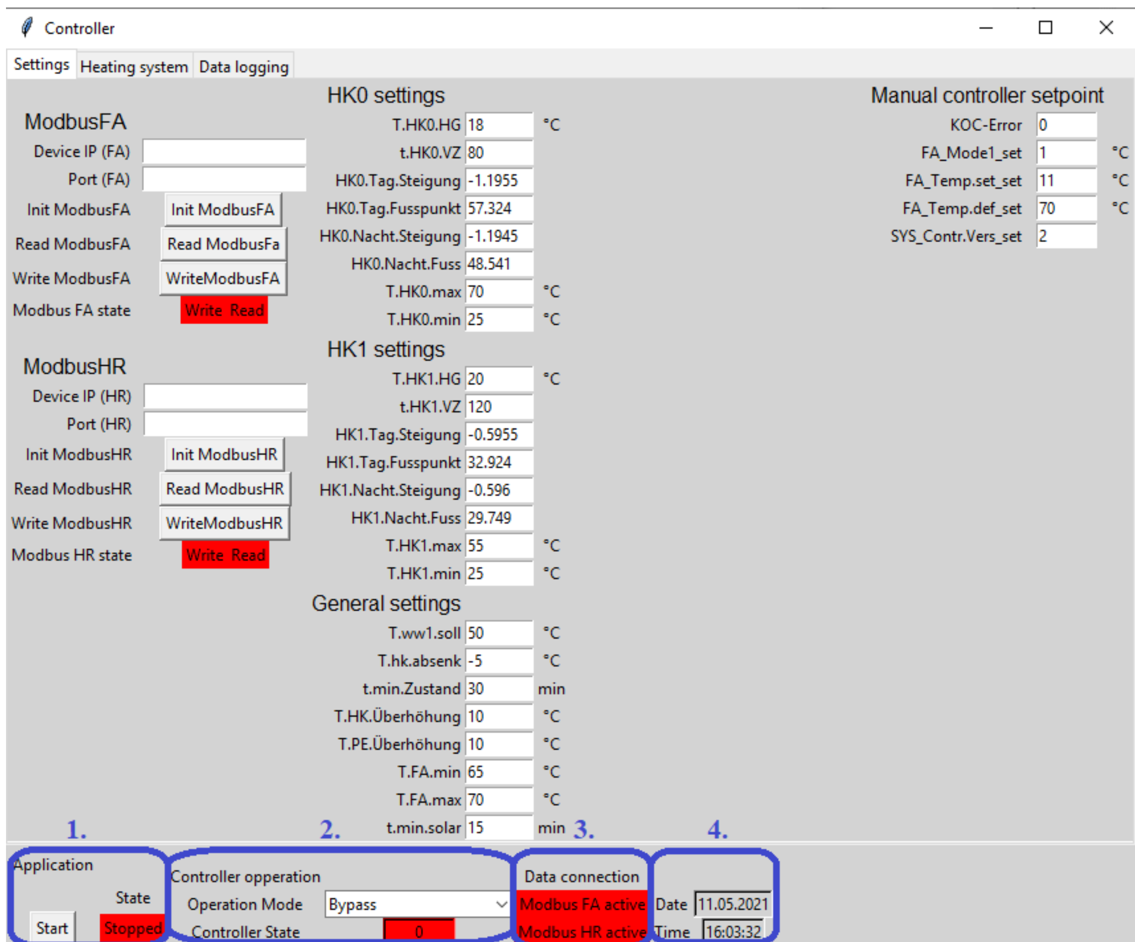

Abbildung 4-2: Python-GUI: Reiter: "Settings"

Die Elemente der GUI sind entsprechend der LabVIEW-Oberfläche angeordnet. In Abbildung 4-2 ist der erste Reiter der Python GUI „Settings" zu sehen. Im oberen Bereich des Fensters kann zwischen den drei Reitern ausgewählt werden. Die GUI verfügt, wie die LabVIEW-GUI, über eine Fußzeile, die eine Übersicht über den Status des Programmes gibt. Die Fußzeile setzt sich aus vier Feldern zusammen. Diese sind in der Abbildung 4-2 jeweils mit einem blauen Rahmen umgeben. Das erste Feld (1.) zeigt den Status der Regler-Modbus-Schnittstelle an. Das Drücken des „Start"-Knopfs startet die Logik der Regler-Modbus-Schnittstelle, woraufhin das Feld „Stopped" grün wird und der Text zu „Running" wechselt. Mit dem Start-Knopf kann die Regler-Modbus-Schnittstelle wieder gestoppt werden. In dem zweiten Feld (2.) kann der Modus des Reglers über ein Drop-down-Menü angepasst werden. In dem Feld „Controller State" wird der Status des Reglers angezeigt. In dem dritten Feld (3.) wird der Status der Modbus-Verbindung angezeigt. Bei „rot" besteht keine Verbindung, bei „grün" besteht eine Verbindung. Das vierte Feld (4.) zeigt das aktuelle Datum und die Uhrzeit.

In dem Reiter „Settings“ können auf der linken Seite IP und Port der Modbus-Verbindungen eingegeben werden. Außerdem lassen sich über die entsprechenden Knöpfe die Clients initialisieren und die Lese- und Schreibfunktion aktivieren. Ist die Lese- bzw. Schreibfunktion bereits aktiv, kann diese deaktiviert werden, wenn der entsprechende Knopf nochmals betätigt wird. In der Mitte befinden sich die Reglerparameter, die für die Steuerungslogik wichtig sind. Auf der rechten Seite sind die Eingabefelder für den manuellen Betrieb.

Jeder Reiter ist in Python ein Frame. In diesem können andere Frames oder die Anzeigeelemente eingebunden und positioniert werden. Für die GUI wurden vier verschiedene Elemente genutzt.

Die **Toolbox** ist ein Drop-Down-Menü, in dem aus unterschiedlichen Möglichkeiten ausgewählt werden kann. Dieser Objekttyp wurde nur für die Auswahl der unterschiedlichen Reglermodi genutzt. (Abbildung 4-2 (2.))

Labels sind Anzeigefelder. Diese können von dem Bediener nicht direkt beeinflusst werden. In der GUI sind alle Beschriftungen und Anzeigefelder von Werten, die nur angezeigt werden, Labels.

Buttons sind Knöpfe. Nach Betätigung wird eine mit dem Knopf verknüpfte Funktion ausgeführt

Entrys sind Eingabefelder. Über Entry-Felder können Eingaben wie Texte oder Zahlen entgegengenommen werden.

Geometriemanager

Bei der GUI wurden zwei Arten von Geometriemanagern benutzt. Geometriemanager dienen dazu, die erzeugten Elemente (Toolbox, Label, Button, Entry) und gegebenenfalls Frames innerhalb von einem Frame anzuordnen. Die verwendeten Geometriemanager sind „.pack()“ und „.grid()“. In den Klammern stehen die Attribute der Geometriemanager.

.pack()

Dieses Tool sorgt dafür, dass einem Element so viel Platz zugewiesen wird ,wie das Element benötigt. Über das Attribut „side“ wird festgelegt, an welchen Rand des Fensters das Element gebunden wird.

.grid()

Dieses Tool ermöglicht das Aufteilen des Fensters in eine Tabelle. Über die Attribute „row“ und „column“ wird festgelegt, in welcher Reihe und Spalte der Tabelle sich das Element befindet.

```
    #FAFrame
pg1Frame1 = Frame(pg1Frame3, bg = "light grey")
        #Zuweisung der Elemente des FAFrame
self.hlFA = Label(pg1Frame1, text = "ModbusFA", font = 12, bg = "light grey")
self.nameipfa = Label(pg1Frame1, text = "Device IP (FA)  ", bg = "light grey")
self.nameportfa = Label(pg1Frame1, text = "Port (FA)  ", bg = "light grey")
self.namereadfa = Label(pg1Frame1, text = "Read ModbusFA  ", bg = "light grey")
self.namewritefa = Label(pg1Frame1, text = "Write ModbusFA  ", bg = "light grey")
self.namereinfa = Label(pg1Frame1, text = "Init ModbusFA  ", bg = "light grey")
self.namestafa = Label(pg1Frame1, text = "Modbus FA state  ", bg = "light grey")
                #Eingabefelder FAFrame
self.ipfa = Entry(pg1Frame1)
self.portfa = Entry(pg1Frame1)
                #Knöpfe FAFrame
self.btmreadfa = Button(pg1Frame1, text = "Read ModbusFa", command = self.readFA)
self.btmwritefa = Button(pg1Frame1, text = "WriteModbusFA", command = self.writeFA)
self.btmreinfa = Button(pg1Frame1, text = "Init ModbusFA", command = self.reinFA)
self.stafaw = Label(pg1Frame1, text = "Write",)
self.stafar = Label(pg1Frame1, text = "Read",)
        #in das FAFrame einfügen
self.hlFA.grid()
self.nameipfa.grid(row=1, column=0, sticky=E)
self.nameportfa.grid(row=2, column=0, sticky=E)
self.namereinfa.grid(row=3, column=0, sticky=E)
self.namereadfa.grid(row=4, column=0, sticky=E)
self.namewritefa.grid(row=5, column=0, sticky=E)
self.namestafa.grid(row=6, column=0, sticky=E)

self.ipfa.grid(row=1, column=1, columnspan=2)
self.portfa.grid(row=2, column=1, columnspan=2)
self.btmreinfa.grid(row=3, column=1, columnspan=2)
self.btmreadfa.grid(row=4, column=1, columnspan=2)
self.btmwritefa.grid(row=5, column=1, columnspan=2)
self.stafaw.grid(row=6, column=1, columnspan=1, sticky=E)
self.stafar.grid(row=6, column=2, columnspan=1, sticky=W)

pg1Frame1.pack(side = TOP, pady=20)
```

Abbildung 4-3: Python-GUI: Quellcode "FAFrame"

Der in Abbildung 4-3 gezeigte Quellcode erstellt die GUI (siehe Abbildung 4-2) die zur Bedienung des ModbusFA dient. Die grünen Texte, die mit einer Raute beginnen, sind Anmerkungen, die keinen Einfluss auf den Programmablauf haben. Die Kommentare teilen den Code in unterschiedliche Abschnitte ein. Die Anweisung „self.“ vor jeder Zeile kommen aus der objektorientierten Programmierung. „self.“ weist den Code, dem Objekt der jeweiligen Klasse zu. Die Funktion der Abschnitte wird im folgendem erläutert.

In der ersten Codezeile wird der Frame „pg1Frame1“ erstellt. Dieser Frame ist die Grundlage für den Abschnitt der GUI. In diesem Frame werden alle benötigten Felder verankert. In dem ersten Abschnitt werden die Labels erzeugt, die das Frame beschriften. Die erste Eingabe gib an, in welchem Frame das Element angelegt wird. Das Attribut „text“ gibt den Text an, der in dem Label angezeigt werden soll. Mit dem Attribut „bg“ (Background)

wird der Hintergrund des Labels angepasst. In dem nächsten Abschnitt werden die Eingabefelder für die Modbus-IP und den Modbus-Port erzeugt. Mit der Methode „.get()“ kann auf die Eingaben in den entry-Elementen zugegriffen werden. In dem nächsten Abschnitt werden die Knöpfe und deren Beschriftung erzeugt. Einem Element vom Typen „Button“ wird über „command“ eine Funktion zugewiesen, die bei Betätigung des Knopfes ausgeführt wird. Für den Knopf „btmreadfa“ ist das die Funktion „readFA“. In dem nächsten Abschnitt werden die zuvor erzeugten Elemente mittels „.grid()“ und den Anweisungen „row“ und column“ in das Frame eingefügt. Ist die Klammer wie bei „hlFA“ leer, werden die Elemente von oben nach unten in der ersten Spalte angeordnet. Nachdem alle Elemente positioniert sind, wird das Frame „pg1Frame1“ mit „.pack()“ auf dem Frame „pg1Frame3“ verankert und ist damit sichtbar.

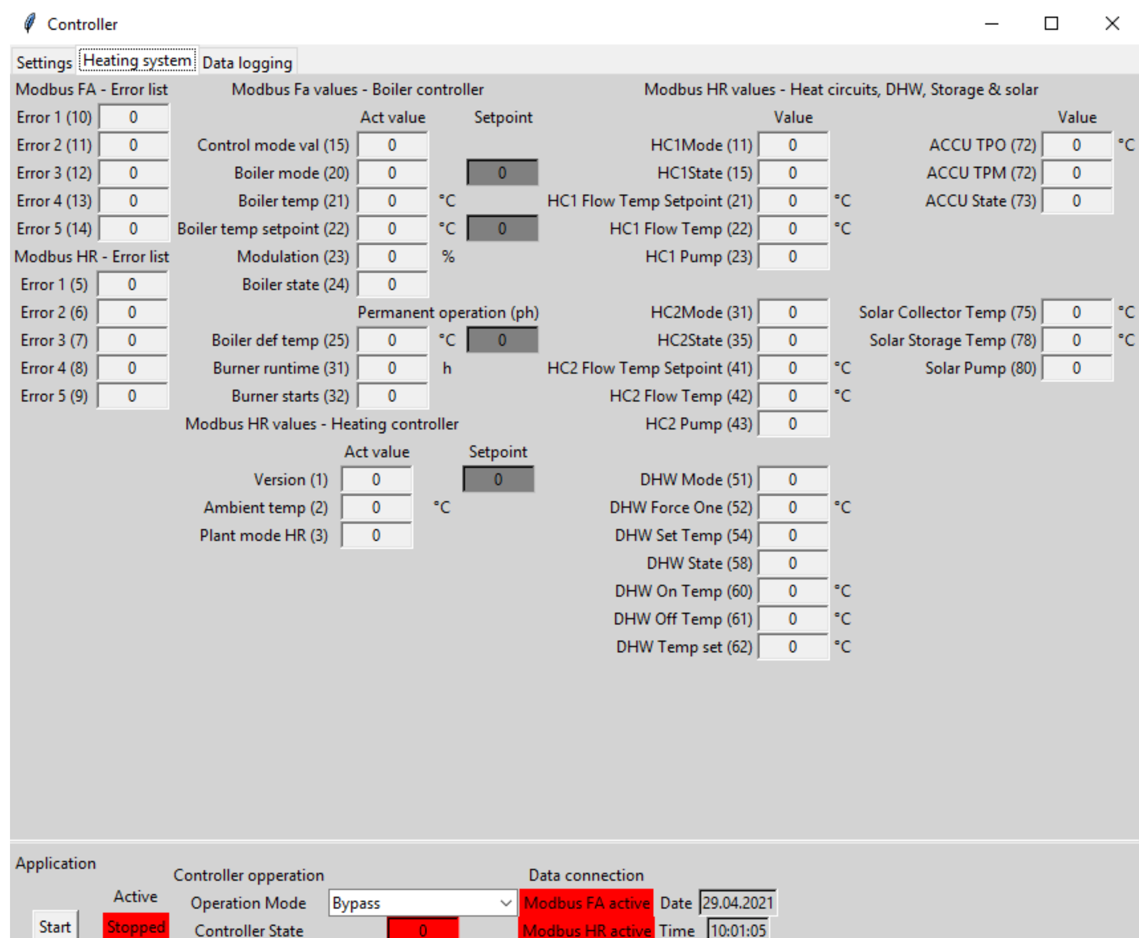

Abbildung 4-4: Python-GUI: Reiter: "Heating system"

Der in Abbildung 4-4 abgebildete zweite Reiter der GUI hat den gleichen Aufbau wie der entsprechende Reiter des LabVIEW-Programmes (vgl. Abbildung 3-2). Auf dieser Seite können nur Werte angezeigt werden. Alle Elemente die zu sehen sind, sind Labels. Über eine Funktion, die alle zwei Sekunden wiederholt wird, werden die Werte mit den Werten

aus dem globalen Speicher aktualisiert. Die Zahl in der Klammer, bei einigen der Anzeigen, beschreibt die Position innerhalb von „DataArray“.

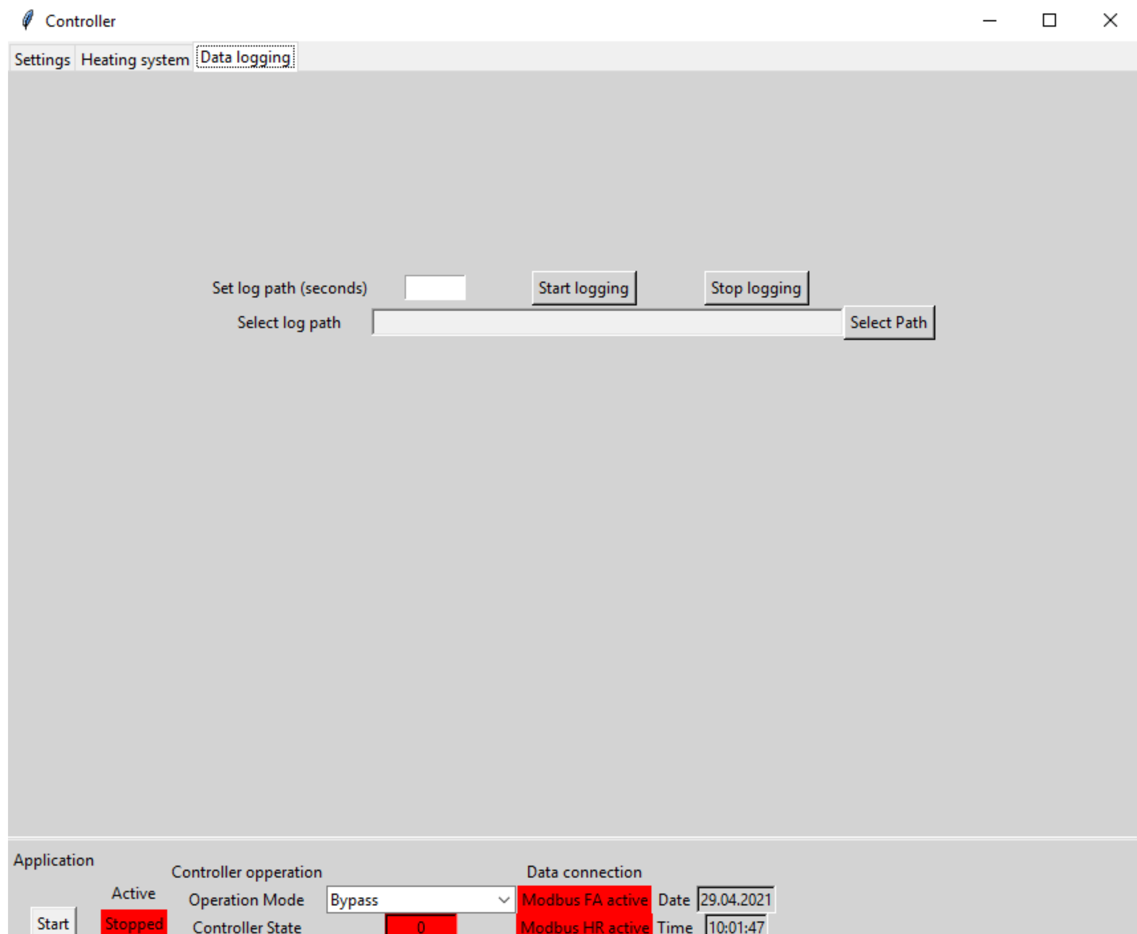

Abbildung 4-5: Python-GUI: Reiter: "Data logging"

In Abbildung 4-5 ist der dritte Reiter des Python-Programmes zu sehen. Auch dieser ist dem entsprechenden LabVIEW-Reiter nachgebildet (vgl. Abbildung 3-6). Der Reiter dient zum Steuern der Datenlogging-Funktion. In dem Feld „Set log path“ wird die Zeit zwischen zwei Zeitschritten eingegeben. Über den Knopf „Select Path“ wird der Speicherort der Datei ausgewählt. Der Dateipfad wird in dem Feld daneben angezeigt. Mit dem Knopf „Start logging“ wird die logging-Funktion gestartet. Mit dem Knopf „Stop logging“ wird die logging-Funktion gestoppt. Der Name der Datei hat das Format „Loggingfile[Startdatum][Startuhrzeit]“.

4.5.2 GlobaleDaten

Bei „GlobaleDaten“ handelt es sich um verschiedene Datensätze, die in einer separaten Datei hinterlegt sind. Die Daten und dazugehörigen Datentypen sind in Tabelle 4 hinterlegt. Über die Veränderung der in dem Speicher hinterlegten Werte kommunizieren die

beiden Threads miteinander. Dabei ist es wichtig die Funktionen, die auf die gemeinsamen Daten zugreifen mit Sperren, so genannten Locks, zu versehen. Wird das nicht gemacht, kann es sein, dass zwei Funktionen gleichzeitig auf denselben Datensatz zugreifen. Dabei kann es zu Datenverlust oder einem Mix aus beiden Ergebnissen der Funktionen kommen. Ein Lock sperrt den Bereich, auf den die Funktion zugreift, solange für andere Threads bis die Lock-Anweisung wieder aufgehoben wird.

Tabelle 4: Daten im globalen Speicher

Datenname	Typenname in Python[3]
DataArray	lst
CSettings	lst
Interfacesettings	bool und str
Manuelle Werte	float
Datalog	bool, str und int
Controller	bool

„DataArray“ hat seinen Namen aus dem LabVIEW-Programm. In LabVIEW handelt es sich dabei um ein Array, in Python hingegen wurde der Datentyp zu einer Liste geändert, aber der Name wurde beibehalten. In „DataArray“ sind alle Daten enthalten, die aus den Bussen gelesen werden und die auf die Busse geschrieben werden sollen. Die Liste DataArray hat einen größeren Umfang als es für die Python-Reglerplattform nötig wäre. Dadurch ist die Möglichkeit gegeben, die Plattform um Funktionen zu erweitern, ohne dass die Liste bearbeitet werden muss.

„CSettings“ ist ebenfalls eine Liste. In dieser sind die Reglereinstellungen gespeichert. Die Werte in der Liste werden, sobald der Reglelbetrieb aufgenommen wird, von der GUI mit den eingegebenen Werten aktualisiert, diese werden anschließend von der Regler-Modbus-Schnittstelle entgegengenommen.

„Interfacesettings“ sind sechs boolesche Werte die für die Modbus-Steuerung benötigt werden. Für ModbusHR und ModbusFA gibt es jeweils drei unterschiedliche Variablen („init“, „read“ und „write“). Die Variable „init“ gibt an, ob der Modbus initialisiert werden soll. Standardmäßig hat die Variable den Wert „False“. Über einen Button in der GUI kann der Wert auf „True“ geschalten werden. Daraufhin wird der Client initialisiert, anschließend wird „init“ wieder auf „False“ gesetzt damit der Client gegebenenfalls neu

[3] „lst“=Liste, „bool“=boolescher Wert, „float“=Gleitkommazahl, „int“=Integer

initialisiert werden kann. Die beiden anderen Variablen „read“ und „write“ geben an, ob der Modbus-Server gelesen und/oder beschrieben werden soll.

„Manuelle Werte“ sind die Werte, die beim manuellen Betrieb an die Modbus-Kommunikation übergeben werden. Beim manuellen Betrieb werden die Werte „FAmodeset“, „FAtempset“, „FAtempdef“ und „SYSYContr“ als Gleitkommazahlen übergeben.

„Datalog“ umfasst alle Daten, die für das Speichern einer externen Datei notwendig sind. Die boolesche Variable „Datalog“ gibt an, ob Daten extern gespeichert werden sollen. „filepath“ ist ein String in dem der Dateipfad gespeichert wird. Mit der Variable „starttime“ wird die Zeit angegeben, bei der die Logging-Funktion gestartet ist. Die Variable „lograte“ gibt die Zeit zwischen den gespeicherten Zeitschritten an.

„Schaltungen“ umfasst die booleschen Variablen „interface“ und „controller“, die das An- und Abschalten der Steuerungslogik und der Modbus-Kommunikation regeln. Die Variable „interface“ schaltet die Modbus-Verbindungen aus und deaktiviert die Lese- und Schreibfunktionen. Die Variable „controller“ deaktiviert die Regler-Modbus-Schnittstelle.

4.5.3 Regler-Modbus-Schnittstelle

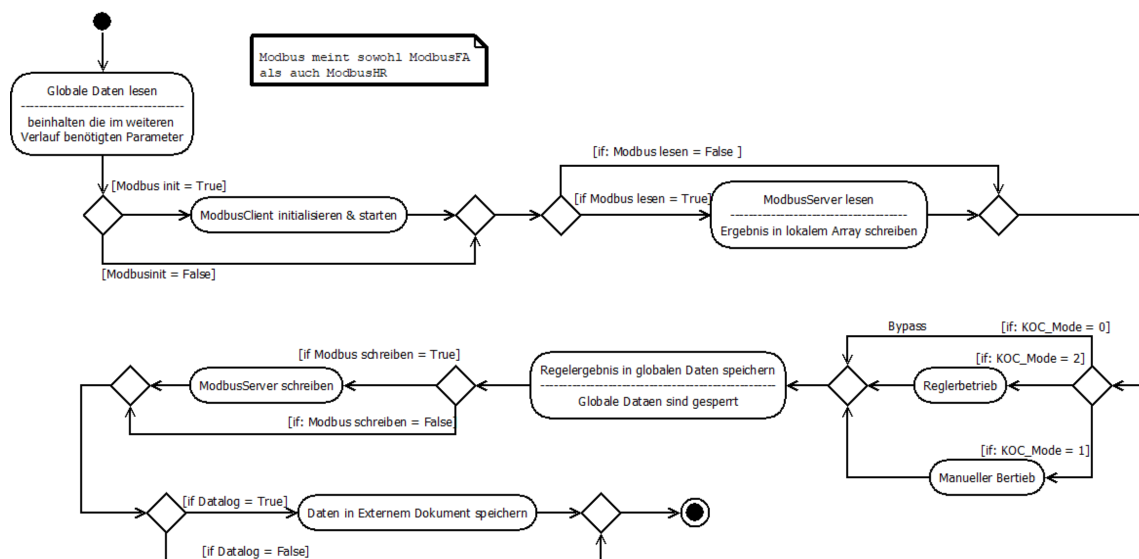

Abbildung 4-6: UML-Diagramm: Python „Regler_Modbus_Schnittstelle“

In Abbildung 4-6 ist der Programmablauf der Regler-Modbus-Schnittstelle dargestellt. Die beiden Modbusse sind nicht separat aufgeführt. Eine Aktivität, die sich auf Modbus bezieht, wird immer sowohl für den ModbusFA als auch für den ModbusHR ausgeführt. In der ersten Aktivität werden die globalen Daten ausgelesen. Diese beinhalten zum einen die Werte, die für die Steuerungslogik notwendig sind, und zum andern die Parameter die festlegen, welche Schritte innerhalb der Regler-Modbus-Schnittstelle ausgeführt werden sollen.

Tabelle 5: Daten Regler_Modbus_Schnittstelle

Datenname	Datentyp[4]	Funktion
GolbalData	Liste	Speichern
Regler Einstellungen	Liste	Einstellungen
FAinit	Boolesche Variable	FA Initialisieren (Ein/Aus)
FAread	Boolesche Variable	FA Lesen (Ein/Aus)
FAwrite	Boolesche Variable	FA Schreiben (Ein/Aus)
HRinit	Boolesche Variable	HR Initialisieren (Ein/Aus)
HRread	Boolesche Variable	HR Lesen (Ein/Aus)
HRwrite	Boolesche Variable	HR Schreiben (Ein/Aus)
Datalogg	Boolesche Variable	Extern speichern (Ein/Aus)
lograte	Integer	Zeitschritt von Datalogging
Filepath	String	Dateipfad für Datalogging
starttime	time	Startzeit für Datalogging
IPFA	String	IP-Adresse ModbusFA
PortFA	Integer	Port ModbusFA
IPHR	String	IP-Adresse ModbusHR
PortHR	Integer	Port ModbusHR
Kopfzeile	Liste	Kopfzeile für Datalogging

Tabelle 5 zeigt die Werte, die von der Regler-Modbus-Schnittstelle entgegengenommen werden. „DataArray“ umfasst mehr Daten als für die Steuerung benötigt werden (siehe Kapitel 4.5.). Die Schnittstelle erstellt eine Kopie der Liste und nimmt die für die Steuerung notwendigen Daten entgegen. Nach jedem Reglerzeitschritt wird die global verfügbare Liste „DataArray“ durch die lokale Kopie aus dem Regler ersetzt. Der Zugriff auf die Globalen Daten in der Zeit, in der der Regler arbeitet, für die GUI gesperrt. In dem nächsten Schritt werden die Modbusse initialisiert bzw. re-initialisiert. Haben die Variablen „FAinit“ bzw. „HRinit“ den Wert „True“, werden die Modbus-Clients initialisiert. In der Aktivität „Modbus-Server lesen“ wird bestimmt, ob die Modbus-Server ausgelesen werden. Bestimmt wird das unter Beachtung der Variablen „FAread“ und „HRread“. Haben diese Variablen den Wert „True“ werden die Server ausgelesen. Die gelesenen Werte ersetzen einen Teil der lokal gespeicherten Liste. In dem vierten Schritt wird anhand des Wertes „KOC_Mode“ der Betriebsmodus der Steuerung bestimmt. Der Wert „KOC_Mode“ steht an erster Stelle in „DataArray“. „KOC-Mode“ in der aktuellen Version kann drei unterschiedliche Werte annehmen (siehe Tabelle 6).

[4] Boolesche Variable-„bool“, Integer-„int“, String-„str“, Liste-„list“

Tabelle 6: Werte für KOC-Mode

Wert	Funktion
0	Bypass
1	Manuell
2	Steuerungsbetrieb

Bei „Bypass“ wird keine Aktivität ausgeführt und die Eingangswerte werden unverändert weitergereicht. Diese Funktion dient in erster Linie um zu überprüfen, ob außerhalb des Reglers eine Funktion einen Fehler ausgibt. Ist der Reglermodus auf „Manuell“, werden die von der GUI vorgegebenen Werte in die Liste „DataArray“ geschrieben. Die manuellen Werte werden nur, wenn der Regler auf „Manuell“ geschalten ist, aus den globalen Daten gelesen. Ist „KOC-Mode“ = 2 wird die normale Steuerungslogik ausgeführt. Dafür werden die gesamten Reglerparameter und die entsprechenden Werte aus dem DataArray verwendet. In dem nächsten Schritt wird das Ergebnis der Steuerung, die manuellen Werte oder die durch „Bypass“ weitergegebenen Werte in den lokalen Daten gespeichert und anschließend werden die globalen Daten durch die lokalen Daten ersetzt. In dem vorletzten Schritt werden die lokalen Daten in Abhängigkeit von der Variable „FAwrite“ bzw. „HRwrite“ auf den Modbus-Server geschrieben. Die letzte Funktion dient dazu, die Liste „DataArray“ bei Bedarf in einer externen Datei zu speichern. Über die GUI lassen sich die Zeitabstände zwischen zwei Zeitschritten und der Speicherort der Zieldatei einstellen.

4.5.4 Modbus-Schnittstellen: ModbusFA und ModbusHR

Die Plattform verfügt über zwei Modbus-Schnittstellen: ModbusFA und ModbusHR. Die IP und der Port werden über die entsprechenden Eingabefelder in der GUI entgegengenommen. Die Modbus-Module nehmen die Werte nicht direkt entgegen, sondern bekommen diese von der Regler-Modbus-Schnittstelle übergeben. Einmalig wird die Initialfunktion aufgerufen, in der zwei Listen erstellt werden. Die Modbus-Kommunikation, wird in beiden Modulen mit sechs Funktionen realisiert. ModbusHR verfügt zusätzlich über eine Funktion zur Umrechnung von negativen Temperaturen.

Tabelle 7: Übersicht der Funktionen der Modbus-Module

Name	Modbus	Aufgabe
__init__	FA und HR	Initialfunktion vom Modbus-Objekt
Initialisieren	FA und HR	Initialisiert den Modbus-Client
Read_Modbus	FA und HR	Liest das Holding Register
Write_Modbus	FA und HR	Schreibt in das Holding Register
Stop_client	FA und HR	Stopt den Modbus-Client
status	FA und HR	Gibt den Status des Client wieder
Trim_Amp_Temp	HR	Rechnet negative Temperaturen um

Tabelle 7 gibt eine Übersicht über die Funktionen der Modbus-Module und welcher Modbus auf die Funktionen zugreift. Die Funktionen werden nachfolgen genauer erklärt.

```
def __init__(self):
    self.lesewerte = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    self.holding_Registers = []
```

Abbildung 4-7: Quellcode der Funktion „__init__“ ModbusFA

In Abbildung 4-7 ist die Initialfunktion von dem ModbusFA-Modul zu sehen. Diese Funktion wird einmal ausgeführt, wenn ein Objekt mit diesem Modul erstellt wird. In dieser Funktion werden zwei Listen erstellt. Die obere Liste (lesewerte) hat genau so viele Einträge, wie Werte von dem Modul an die Schnittstelle weitergegeben werden. Die zweite Liste dient als Zwischenspeicher für das gelesene Holding Register.

```
def initialisieren(self, IP, Port):
    self.ClientHR = ModbusClient(IP, port = Port)
    self.ClientHR.open()
    self.status = True
```

Abbildung 4-8: Quellcode der Funktion „initialisieren“ ModbusFA

In der in Abbildung 4-8 ist die Funktion abgebildet die den Modbus-Client initialisiert. Die Funktion nimmt den Port und die IP-Adresse entgegen, erstellt den Client und öffnet die Verbindung zu einem Server.

```python
def read_ModbusFA (self):
    self.holding_Registers = self.ClientFA.read_holding_registers(1, 32)
    self.lesewerte[0] = self.holding_Registers[9]
    self.lesewerte[1] = self.holding_Registers[10]
    self.lesewerte[2] = self.holding_Registers[11]
    self.lesewerte[3] = self.holding_Registers[12]
    self.lesewerte[4] = self.holding_Registers[13]
    self.lesewerte[5] = self.holding_Registers[14]
    self.lesewerte[6] = self.holding_Registers[19]
    self.lesewerte[7] = round(self.holding_Registers[20] * 0.1, 1)
    self.lesewerte[8] = round(self.holding_Registers[21] * 0.1, 1)
    self.lesewerte[9] = self.holding_Registers[22]
    self.lesewerte[10] = self.holding_Registers[23]
    self.lesewerte[11] = round(self.holding_Registers[24] * 0.1, 1)
    self.lesewerte[12] = self.holding_Registers[30]
    self.lesewerte[13] = self.holding_Registers[31]
```

Abbildung 4-9: Quellcode der Lesefunktion von ModbusFA

Bei der in Abbildung 4-9 dargestellten Funktion werden die in Abbildung 4-7 gezeigten Funktionen benötigt. Mit der ersten Anweisung werden, in der Liste „Holding_Registers", die Werte 1 bis 32 aus dem Holding Register des Servers gespeichert. Anschließend werden die für die Regelung notwendigen Werte in der Liste „lesewerte" gespeichert. Die Liste „lesewerte" nimmt die Modbus-Schnittstelle entgegen. Die Werte 7, 8 und 11 werden mit dem Faktor 0,1 multipliziert und das Komma um eine Nachkommastelle verschoben. Dies ist notwendig, da wie in Kapitel 2.1.2 beschrieben, in einem Holding Register nur Integer gespeichert werden. Bei den Werten 7, 8 und 11 handelt es sich um Temperaturen, die, um eine Nachkommastelle zu speichern, mit dem Faktor 10 multipliziert in dem Holding Register gespeichert sind. Eine Temperatur von 23,5 °C ist im dem Holding Register also als 235 gespeichert. Die Modbus-Module speichern die gelesenen Werte in der Liste „lesewerte" die anschließend einen Teil der Liste „DataArray" ersetzt.

```python
def write_ModbusFA (self, FA_Mode_set, FA_Temp_set, FA_Temp_def):
    self.ClientFA.write_single_register(20, FA_Mode_set)
    self.ClientFA.write_single_register(22, FA_Temp_set * 10)
    self.ClientFA.write_single_register(25, FA_Temp_def * 10)
```

Abbildung 4-10: Quellcode der Schreibfunktion von ModbusFA

Abbildung 4-10 zeigt die Schreibfunktion von ModbusFA. Dieser Funktion werden die Werte, die in das Holding Register geschrieben werden sollen, übergeben. Anschließend werden in drei separaten Anweisungen die Werte auf das Holding Register geschrieben. Die Zahl innerhalb der Klammer gibt den Speicherort des Wertes an. Die Werte „FA_Temp_set" und „FA_Temp_def" werden aus den oben genannten Gründen mit dem Faktor 10 multipliziert.

Die Funktionen „Status“ und „Stop_Client“ dienen der Statusabfrage des Modbus-Clients und dem Schließen der Modbus-Verbindung. Die ModbusHR-Schnittstelle hat den gleichen Aufbau. Unterschiede bestehen nur in der Anzahl der gelesenen und geschriebenen Werte.

ModbusFA

ModbusFA sammelt die Daten der Feuerungsautomatik, darunter fallen der aktuelle Kesselstatus, die Brennraumtemperatur, die Laufzeit des Kessels und eventuelle Fehlermeldungen.

Tabelle 8: Gelesene und geschriebene Werte von ModbusFA

DataArray	Variablennamen	Einheit	Zugriffsart	Register
4	FA_Error.1	-	lesend	11
5	FA_Error.2	-	lesend	12
6	FA_Error.3	-	lesend	13
7	FA_Error.4	-	lesend	14
8	FA_Error.5	-	lesend	15
9	FA_Ext.Contr_act	-	lesend	16
10	FA_Mode1_act	-	lesend	21
11	FA_Temp	°C	lesend	22
12	FA_Temp.set_act	°C	lesend	23
13	FA_Mod	%	lesend	24
14	FA_State	-	lesend	25
15	FA_Temp.def_act	°C	lesend	26
16	FA_Runtime	h	lesend	32
17	FA_Starts	-	lesend	33
18	FA_Mode1_set	-	schreibend	20
19	FA_Temp.set_set	°C	schreibend	22
20	FA_Temp.def_set	°C	schreibend	25

In Tabelle 8 sind die Werte abgebildet, die von dem ModbusFA-Client gelesen und geschrieben werden. Die Namen der Variablen sind aus dem LabVIEW-Programm übernommen worden. Die Zugriffsart beschreibt, ob der entsprechende Wert von dem Server gelesen oder auf den Server geschrieben wird. In der Spalte „DataArray“ ist aufgeführt, an welcher Position in den Liste „DataArray“ der Reglerplattform die jeweilige Variable gespeichert ist. Die Spalte „Register“ gibt die Position der Variable in dem Holding Register des Servers an. Zu dem jetzigen Stand der Reglerplattform wird nur ein Teil der Variablen genutzt. Die genutzten Werte sind die Soll- und die Isttemperatur der Feuerungsautomatik „FA_Temp.def_act“ und „FA_Temp.set_act“ sowie der Modus der Feuerungsautomatik „FA_Mod“.

ModbusHR

Der zweite Bus ist der ModbusHR, dieser sammelt die Daten der Heizregister. Darunter fallen Werte wie die Umgebungstemperatur, die Temperatur der beiden Heizkreisen und des Warmwassers, der Status der Kreislaufpumpen und die Temperaturen in dem Pufferspeicher.

Tabelle 9: Gelesene und geschriebene Werte von ModbusHR

DataArray	Variablennamen	Einheit	Zugriffsart	Register
21	SYS_Contr.Vers_act		lesend	1
22	SYS_T.Amb	° C	lesend	2
23	SYS_Plant.Mode_act		lesend	3
24	SYS_Error 1		lesend	5
25	SYS_Error 2		lesend	6
26	SYS_Error 3		lesend	7
27	SYS_Error 4		lesend	8
28	SYS_Error 5		lesend	9
29	HC1_HC.Mode_act		lesend	11
30	HC1_State		lesend	15
31	HC1_Current-Flow-	° C	lesend	21
32	HC1_Current-Flow-Temp	° C	lesend	22
33	HC1_Pump		lesend	23
34	HC2_HC.Mode_act		lesend	31
35	HC2_State		lesend	35
36	HC2_Current-Flow-	° C	lesend	41
37	HC2_Current-Flow-Temp	° C	lesend	42
38	HC2_Pump		lesend	43
39	DHW_DHW.Mode_act		lesend	51
40	DHW_DHW.Force.Once_act		lesend	52
41	DHW_DHW.Set.Temp_act	° C	lesend	54
42	DHW_DHW.State		lesend	58
43	DHW_DHW.On.Temp_act	° C	lesend	60
44	DHW_DHW.Off.Temp_act	° C	lesend	61
45	DHW_DHW.Temp.Set	° C	lesend	62
46	ACCU_ACCU.TPO	° C	lesend	73
47	ACCU_ACCU.TPM	° C	lesend	74
48	ACCU_ACCU.State		lesend	75
49	Solar_Collector.Temp	° C	lesend	76
50	Solar_Storage.Temp	° C	lesend	78
51	Solar_Solar.Pump		lesend	80
52	SYS_Contr.Vers_act		schreibend	1

Tabelle 9 hat den gleichen Aufbau wie Tabelle 8. Für die Reglerfunktion notwendig sind die Umgebungstemperatur „SYS_T.Amb_act“, die obere und die mittlere Temperatur im Pufferspeicher der Heizanlage „ACCU_ACCU.TPO“ und „ACCU_ACCU.TPM“, sowie die Temperatur der Solaranlage „Solar_Collector.Temp“. Die Umgebungstemperatur ist

die einzige Temperatur, die einen negativen Wert anzeigen kann. Um die negativen Temperaturen aufzufangen, verfügt die Klasse ModbusHR über eine Umrechnungsfunktion.

```
def Trim_Amb_Temp (self, input):
    if input > 65000:
        output = round((input - 65536) * 0.1, 1)
    else:
        output = input * 0.1
    return output
```

Abbildung 4-11: Funktion Trim_Amb_Temp in ModbusHR zur Umrechnung negativer Temperaturen

Abbildung 4-11 zeigt die Umrechnungsfunktion „Trim_Amb_Temp“ der Klasse ModbusHR. Der Datentyp, mit dem der Modbus arbeitet ist ein vorzeichenloser 16-bit Integer. Negative Zahlen werden im Dualsystem im Zweierkomplement dargestellt. Zur Berechnung des Zweierkomplements wird der um 1 erhöhte Wertebereich des Intergers von dem Ergebnis abgezogen. Zur Berechnung wird zunächst überprüft, ob eine Umrechnung vorgenommen werden muss. Ist der Inputwert höher als 65000, wird die Umrechnung vorgenommen. In dem Fall wird 65536 von dem Input abgezogen. Anschließend wird der Wert zur Anpassung der Nachkommastelle noch mit 0,1 multipliziert. Der maximale Wertebereich eines 16-bit Integer liegt bei 65535. Damit ergibt sich ein Wertebereich für die negativen Temperaturen von –53,5 C bis -0,1 °C

$$T_{neg\ max} = (65535-65536) * 0{,}1\ °C = \mathbf{-0{,}1\ °C}$$

$$T_{neg\ min} = (65001-65536) * 0{,}1\ °C = \mathbf{-53{,}5\ °C}$$

4.5.5 Regler

Der Regler nimmt die Daten von der Regler-Steuerung entgegen, bearbeitet sie entsprechend der Logik und gibt das Ergebnis an die Plattform zurück. Der Aufbau des Reglers ist in der Praktikumsarbeit „Programmierung eines Heizungsreglers in Python“ [11] ausführlich beschrieben.

Die Regler-Steuerung übergibt dem Regler die Listen mit den „Reglereinstellungen“ und „DataArray“. Von „Reglereinstellungen“ werden alle Werte innerhalb des Reglers genutzt. Von „DataArray“ verwendet der Regler acht Werte für die Steuerungslogik und fünf Werte werden als Ergebnis der Steuerungslogik ausgegeben und wieder in der Liste „DataArray“ gespeichert.

Tabelle 10: Schnittstelle Regler-Reglersteuerung

DataArray	Variablenname	Zugriffsart	Datentyp
0	KOC_Mode	lesend	Int
12	FA_Temp.set_act	lesend	Float
13	FA_Mode_act	lesend	Int
22	SYS_T.Amb	lesend	Float
43	DHW_DHW.On.Temp_act	lesend	Float
46	ACCU_ACCU.TPO	lesend	Float
47	ACCU_ACCU.TPM	lesend	Float
49	Solar_Collector.Temp	lesend	Float
1	KOC_State	schreibend	Int
18	FA_Mode_set	schreibend	Int
19	FA_Temp.set_set	schreibend	Int
20	FA_Temp.def_set	schreibend	Int
52	SYS_Contr.Vers_act	schreibend	Int

Tabelle 10 gibt eine Übersicht der Daten, die der Regler entgegennimmt und die der Regler ausgibt. Alle Temperaturen werden als Gleitkommazahlen entgegengenommen und ausgegeben. Alle anderen Werte sind als Integer gespeichert. Die Umwandlung in die für die Steuerung notwendigen Datentypen findet innerhalb des Reglers statt.

5 Testläufe

Um die Funktionalität der entwickelten Reglerplattform zu zeigen, werden Funktionstests durchgeführt (siehe. Kapitel 2.3.1). Nicht-funktionale Tests müssen weiterführend in einem Feldtest durchgeführt werden. Die detaillierte Durchführung der Tests wird ausführlich im Anhang beschrieben. Dies umfasst das sowohl das Testziel, den Ablauf des Tests, die Ausgabe der Werte im Editor oder in der GUI als auch das Testergebnis. In diesem Kapitel werden hingegen nur die Testergebnisse zusammengefasst.

5.1 Komponententests

Komponententests wurden für die „GUI“ und die Modbus-Module der Plattform durchgeführt. Dafür wurden die Schnittstellen mit den anderen Modulen simuliert. Die Schnittstelle der Module zu den „GlobalenDaten, wird durch eine „print“-Anweisung ersetzt. Der Wert, der sonst an die „GlobalenDaten“ weitergegeben wird, wird dann auf der Konsole ausgegeben. Für den Regler wird kein Funktionstest durchgeführt. Dessen Funktionalität ist in „Programmierung eines Heizungsreglers in Python“ [11] bereits ausführlich dokumentiert. In Form von Integrationstests wird im Anschluss die Funktionsfähigkeit der einzelnen Threads getestet.

5.1.1 Komponententest: GUI

Im Komponententest der GUI, wird die Funktionalität der Knöpfe und der Eingabefelder getestet. Die Anzeigen werden in dem darauffolgenden Integrationstest überprüft. Der komplette Funktionstest ist im Anhang A geschildert. Geprüft wurden die Funktionen des Reiters „Settings“. Die Funktion der Eingabeelemente und der Knöpfe des Reiters „Settings“ konnten in diesem Test gezeigt werden. Weiterhin wurde gezeigt, dass die Visualisierung der „read“- und „write“-Variablen zur Modbus-Steuerung innerhalb der GUI stattfindet. Der Hintergrund der entsprechenden Label ist grün, wenn die Variable den Wert „True“ hat und rot wenn die Variable den Wert „False“ hat. (siehe Anhang A Abschnitt GUI)

Der Reiter „Heating system“ zeigt die Werte des globalen Speichers an. Die Funktion dieses Reiters wird deswegen anhand des Integrationstests im Zusammenspiel mit den „GlobalenDaten“ getestet. Gleiches gilt für den Reiter „Data logging“, da dieser die Werte aus dem „DataArray“ in einer externen Datei speichert. (siehe Anhang B Abschnitt GUI und Abschnitt Gesamttest)

5.1.2 Komponententest: ModbusFA und ModbusHR

Die beiden Module zur Modbus-Kommunikation wurden identisch getestet. In Anhang B sind beide Tests aufgeführt. Die Herangehensweise wird anhand von dem Modul ModbusFA erläutert.

Die Funktionstests von ModbusFA und ModbusHR zeigten, dass die Module in der Lage sind, einen Modbus-Client zu initialisieren, Werte auszulesen und diese zu schreiben. Weiterhin kann eine Statusabfrage durchgeführt und die Verbindung zu dem Server geschlossen werden. (siehe Anhang A Abschnitt ModbusFA und ModbusHR)

5.2 Integrationstests

In den Integrationstests wurden die einzelnen Threads im Zusammenhang mit „Globale-Daten" getestet. Für die GUI und die „Regler-Modbus-Schnittstelle" wurde Tests durchgeführt.

5.2.1 Integrationstest: GUI

Der Integrationstest der GUI hat gezeigt, dass die GUI die Werte aus den Globalen Daten entgegennehmen und korrekt anzeigen kann. Weiterhin ist die GUI in der Lage, die Werte, die von einem Benutzer in die Eingabefelder eingegeben werden, an die „GlobalenDaten" weiterzugeben und dort zu speichern. (siehe: Anhang A Abschnitt GUI)

5.2.2 Integrationstest: Regler-Modbus-Schnittstelle

Für den Integrationstest wurden „GlobaleDaten", die Modbus-Module und die Regler-Modbus-Schnittstelle verwendet. In dem Test wurde erfolgreich demonstriert, dass die Schnittstelle die Variablen, die von „GlobaleDaten" vorgegeben werden, entgegennehmen kann. Mit den Modbus-Modulen und den Variablen aus „GlobaleDaten" kann die Schnittstelle zwei Modbus-Clients erstellen und über diese mit den Servern von ModbusPal kommunizieren. Die gelesenen Werte werden im Anschluss in der Liste „DataArray" in „GlobaleDaten" gespeichert werden. Die Werte aus „DataArray" können auf die Server geschrieben werden. (siehe: Anhang B Abschnitt Regler-Modbus-Schnittstelle)

5.2.3 Integrationstest Reglerplattform ohne Regler

In diesem Test wurde getestet, ob mit Hilfe der GUI die Modbus-Verbindung hergestellt werden kann und ob die Verbindungen in der GUI visualisiert werden. Weiterhin wurde geprüft, ob der manuelle Modus der Plattform funktioniert.

Die Plattform wurde gestartet und nach der Anleitung in 8Anhang B Integrationstests in Betrieb genommen. Es konnte gezeigt werden, dass sich über die GUI der Betrieb der Modbus-Module starten lässt. Der Status der Verbindung wird in der GUI visuell angezeigt. Ist bei den Modbus-Modulen die Lese- und Schreibfunktion aktiviert, werden die gelesenen Daten in der Liste „DataArray“ in „GlobaleDaten“ gespeichert. In dem Reiter „HeatingSystem“ können diese Daten eingesehen werden. (siehe Anhang B Abschnitt Reglerplattform ohne Regler)

5.3 Gesamttest

In dem Gesamttest wird abschließend die gesamte Reglerplattform zusammen mit dem Regler getestet. Die Funktionalität der Steuerungsfunktion wird an einem Zeitschritt beispielhaft simuliert. Die Ausgangswerte für diesen Zeitschritt werden über ModbusPal an die Plattform übergeben. Dafür wird der entsprechende Wert in den Holding Registern für beide Server verändert. Das Ergebnis des Zeitschritts wird durch die Datalogg-Funktion des Reglers in einer Logdatei gespeichert. Dadurch wird die Funktionalität der Datenspeicherung ebenfalls nachgewiesen. Als Referenz für den Test dienen die Daten aus der Praktikumsarbeit „Programmierung eines Heizungsreglers in Python“.

Neben den Regler-Einstellungen werden für die Simulation eines Zeitschrittes sieben der von den Bussen gelesenen Werte benötigt. In Tabelle 11 sind die bei dem Test verwendeten Variablen zu sehen. Die erste Spalte zeigt die Namen, wie sie als Kommentar in der Datei „GlobaleDaten“ in der Liste „DataArray“ bei den jeweiligen Variablen stehen. In der zweiten Spalte steht, welcher der beiden Busse auf die jeweilige Variable zugreift. Die dritte Spalte zeigt die Zugriffsart. Die vierte Spalte zeigt den Speicherort innerhalb des Servers. In der letzten Spalte ist der Wert der der Variabel in dem Gesamttest zugewiesen wird. Zu beachten ist, dass die Werte in der vierten Spalte, in ModbusPal um eins erhöht werden müssen, da ModbusPal bei „HoldingRegister 1“ zu zählen beginnt. Aus den Werten der letzten beiden Zeilen ergibt sich, welche Variable wie geändert werden muss, um einen Zeitschritt zu simulieren. Für die Simulation müssen die Werte in den Servern angepasst werden, auf die lesend zugegriffen wird. Der Wert für „KOC_Mode“ ergibt sich aus der Reglerlogik.

Tabelle 11: Übersicht der Variablen für den Regler

Variablenname	**Bus**	**Zugriffsart**	**Speicherort**	**Testwert**
FA_Temp.set_act	FA	lesend/schreibend	21	75
FA_Temp.def_act	FA	lesend/schreibend	24	70
FA_Mod	FA	lesend/schreibend	19	2
SYS_T.Amb	HR	lesend	1	2
DWD_DWD.On.Temp	HR	lesend	59	62,1
ACCU_ACCU.TPO	HR	lesend	71	60,9
ACCU_ACCU.TPM	HR	lesend	72	61,2
Solar_Collector.Temp	HR	lesend	74	55
SYS_Contr_Vers_set	HR	schreibend	0	-

In diesem Test wurde der erste Zeitschritt des ersten Testlaufs aus dem Praktikumsbericht simuliert. Dafür wurde den Variablen aus der letzten Spalte aus Tabelle 11 verwendet.

Die Werte für die Variablen werden in der Server-Simulation eingegeben. Die Clients der Plattform lesen diese Werte dann aus und übergeben sie an den Regler. Alle Temperaturen wurden mit dem Faktor 10 multipliziert, damit eine Nachkommastelle dargestellt werden kann (siehe Anhang CGesamttest).

Verglichen wird das Ergebnis der Reglerfunktion. Dafür werden die Werte von „KOC_Mode", „FA_Mode1.set" und „KOC_State" verglichen. Tabelle 12 stelle das Ergebnis der Reglerplattform dem Ergebnis aus dem Praktikum gegenüber. Zu sehen ist, dass beide zu dem gleichen Ergebnis kommen. Durch den Gesamttest konnte gezeigt werden, dass das Ergebnis der Steuerung für einen Zeitschritt bei der Reglerplattform identisch ist dem Ergebnis aus dem Praktikum.

Tabelle 12: Vergleich des Ergebnis der Reglerplattform mit dem Ergebnis aus dem Praktikum

Variablenname	Ergebnis Reglerplattform	Ergebnis Praktikum
KOC_Mode	2	2
FA_Mode1.set	2	2
KOC_State	3	3

6 Fazit

6.1 Zusammenfassung

Das neue GEG-Gesetz schafft neue Anreize, den Wärmeenergiesektor nachhaltiger zu gestalten. In Regionen, in denen die Anbindung an ein Wärmenetz nicht möglich ist, stellt die kombinierte Wärmebereitstellung aus Biomasse und Solarthermie eine sinnvolle Alternative zu Öl- und Gasheizungen dar um CO_2-Emissionen zu verringern. Derartige Heizsysteme haben im Vergleich zu konventionellen Heizsystemen einen erhöhten Bedarf an Steuerungs- und Regelungstechnik. Zusätzlich dazu kann durch eine präzise und genaue Steuerung der Ausstoß von Schadstoffen verringert werden. Aus diesem Grund wird am DBFZ ein Prototyp entwickelt, an dem verschiedene Heizstrategien entwickelt werden können.

Ziel der Arbeit war eine softwaretechnische Umsetzung der wesentlichen Bestandteile einer in LabVIEW vorhandenen Reglerplattform. Die Reglerplattform soll:

- eine GUI besitzen
- die Modbus-Kommunikation mit zwei Modbus-Clients soll überwacht und gesteuert werden
- der Regler in der Plattform soll die Werte entgegennehmen bearbeiten und wieder ausgeben
- das Ergebnis der Steuerung soll an in einer externen Datei gespeichert werden.

Die Funktionen der LabVIEW Reglerplattform wurden mittels eines UML-Diagrammes dargestellt. Auf Grundlage des UML-Diagrammes wurde ein Softwareaufbau für das Python-Programm entwickelt.

Die Reglerplattform verfügt über zwei Threads, die über einen globalen Speicher miteinander kommunizieren. Die Threads nehmen jeweils unterschiedliche Aufgaben wahr. Einer der Threads steuert die GUI und der andere steuert die Modbus-Kommunikation und die Reglerfunktion der Plattform.

Die GUI hat drei Reiter und eine Fußzeile. Über die Reiter können Werte zur Veränderung des globalen Speichers eingegeben, der globale Speicher ausgelesen und das Ergebnis der Regelung in einer externen Datei gespeichert werden. Über die GUI besteht die Möglichkeit, den zweiten Thread zu überbrücken.

Der zweite Thread steuert den Ablauf der Regelungsfunktionen der Plattform. Es können zwei ModbusTCP-Clients erstellt werden. Die Clients können lesen oder schreibend auf den Server zugreifen. Die dafür benötigten Werte für IP und Port werden dem globalen Speicher entnommen. Weiterhin wird der integrierte Regler von der Modbus-Schnittstelle aufgerufen und ausgeführt.

In dem letzten Schritt wurde die Funktionsfähigkeit der Plattform validiert. Dafür wurden die Module der Plattform zunächst in separaten Komponententest getestet, die jeweiligen Schnittstellen wurden simuliert. Anschließend wurde für jeden Thread ein Integrationstest durchgeführt. In diesem wurde getestet, ob die Threads Daten aus den „GlobalenDaten" lesen und in diese schreiben können. Im Gesamttest wurde die vollständige Reglerplattform getestet. Dafür wurde ein Zeitschritt aus der Praktikumsarbeit „Programmierung eines Heizungsreglers in Python" simuliert, indem die entsprechenden Werte in der Modbus-Server-Simulation eingegeben wurden. Mit den Tests konnte die Funktionsfähigkeit der Plattform nachgewiesen werden. Bevor die Plattform eingesetzt werden kann, sollte diese noch um die Funktionen in Kapitel 6.2 erweitert werden.

6.2 Ausblick

Damit die Reglerplattform in einem privaten Gebäude laufen kann, ist es notwendig, die bestehende Plattform um einige Funktionen, wie eine Fehlerbehandlung und einen Puffer zur Datensicherheit, zu erweitern. Zusätzlich ist es notwendig, die Python-Plattform um die Funktionen des LabVIEW-Reglers zu erweitern, die in dieser Arbeit ausgelassen wurden (siehe Kapitel 4.3). Durch eine Fehlerbehandlung können Fehler abgefangen werden und diese führen dann nicht mehr dazu, dass das Programm abstürzt. Hat ein Holding Register keinen Wert, führt das zum jetzigen Zeitpunkt zu einem Fehler und der Thread wird beendet. Mit der Einbindung einer Fehlerbehandlung ist es auch sinnvoll die Modbus-Kommunikation um eine Timeoutabfrage zu erweitern, wodurch die sichere Übertragung von Daten gewährleistet werden kann. Die Timeoutabfrage findet über eine Zeitdifferenz statt. Wird ein Bus-Telegramm innerhalb die Zeitdifferenz nicht beantwortet, wird ein Fehler ausgegeben. Da die aktuelle Reglerplattform keine Fehler ausgibt, ist eine solche Abfrage noch nicht sinnvoll.

Eine weitere Funktion, die in der Reglerplattform ergänzt werden muss, ist ein Puffer. In der aktuellen Version ist es so, dass die beiden Threads das Objekt „DataArray" sperren, wenn sie auf dieses zugreifen. Der jeweils andere Thread läuft in der Zeit jedoch weiter und die Werte, die nicht in zwischengespeichert werden, gehen verloren. An dieser Stelle

kann ein Puffer installiert werden. Jeder der verwendeten Threads würde dabei einen eigenen Puffer benötigen.

Abbildung 6-1: Schema für einen erweiterten Puffer

Abbildung 6-1 zeigt, wie diese Puffer aufgeteilt sein könnten. Der Thread für Modbus-Kommunikation und Regler würde seine Daten an den Puffer „Globale Daten“ weitergeben und auch von diesem beziehen. Gleiches gilt für den Thread „GUI“. Grafische Oberflächen haben, um den Rechner zu entlasten, häufig eine niedrigere Zykluszeit als der eigentliche Programmablauf. Durch einen Puffer wird gewährleistet, dass immer ein Datensatz zur Verfügung steht und trotzdem keine Daten verloren gehen. Insbesondere durch eine Ausgestaltung als Ringpuffer mit Zeigern wäre dies möglich.

Die Reglerplattform sollte auch noch diversen nicht-funktionalen Tests unterzogen werden (siehe: Kapitel 2.3.2). Durch diese Tests wird die Geschwindigkeit, Stabilität, Benutzerfreundlichkeit und Skalierbarkeit der Software geprüft.

7 Literaturverzeichnis

[1] S. Solomon, D. Qin, M. Manning, M. Marquis, K. Averyt, M. M. Tignor, H. L. Miller, Jr. und Z. Chen, „Clima Change 2007, The Physical Science Basis,“ IPCC, 2007.

[2] „Paris Agreement,“ Paris, 2015.

[3] Bundesministerium für Justiz und Verbraucherschutz, „Gesetze im Internet,“ [Online]. Available: https://www.gesetze-im-internet.de/eeg_2014/inhalts_bersicht.html. [Zugriff am 2021 April 2021].

[4] Umweltbundesamt, „Erneuerbare Energien in Zahlen,“ 4 März 2021. [Online]. Available: https://www.umweltbundesamt.de/themen/klima-energie/erneuerbare-energien/erneuerbare-energien-in-zahlen#uberblick. [Zugriff am 14 April 2021].

[5] D. Büchner und C. Schraube, „Entwicklung eines Energiemanagementsystems zur kombinierten Nutzung erneuerbarer Energien (Abschlussbericht KombiOpt),“ Leipzig, 2019.

[6] S. Theurich, D. Büchner und C. Schraube, „Präentation des Projekts SNuKR,“ Leipzig, 2020.

[7] BentoNet GmbH, [Online]. Available: https://www.bentonet.de/bentonet/. [Zugriff am 26 April 2021].

[8] Kiwigrid GmbH, [Online]. Available: https://www.kiwigrid.com/plattform/. [Zugriff am 26 April 2021].

[9] G. Schnell und B. Wiedemann, „Das TCP/IP-Protokoll,“ in *Bussysteme in der Automatisierungstechnik*, 9 Hrsg., Wiesbaden, Springer, 2019, p. 17.

[10] Wachendorff, „Modbus Grundlagen,“ Geisenheim, 2014.

[11] F. Staude, „Programmierung eines Heizungsreglers in Python," Leipzig, 2021.

[12] Python Software Foundation, „Python.org," [Online]. Available: https://www.python.org/. [Zugriff am 14 Abril 2021].

[13] Alan C. Kay, „The early Histoy of smaltalk," 1993. [Online]. Available: http://propella.sakura.ne.jp/earlyHistoryST/EarlyHistoryST.html. [Zugriff am 05 Mai 2021].

[14] Object Managemant Group, „www.uml.org," [Online]. Available: https://www.uml.org/what-is-uml.htm. [Zugriff am 14 April 2021].

[15] Python Software Foundation, „jython.org," [Online]. Available: https://www.jython.org/index. [Zugriff am 19 April 2021].

[16] SourceForge, „SourceForge.net," [Online]. Available: http://modbuspal.sourceforge.net/. [Zugriff am 14 April 2021].

[17] G. Lee, „loadview," 16 Oktober 2020. [Online]. Available: https://www.loadview-testing.com/de/blog/arten-von-softwaretests-unterschiede-und-beispiele/. [Zugriff am 10 Mai 2021].

8 Anhang

Anhang A Komponententest

In den Tests wird den Variablen in den Listen und Registern häufig der Wert zugewiesen, der ihrer Position in der Liste bzw. dem Register entspricht. Dies wird gemacht, damit einen Wert besser zugewiesen werden kann. Ansonsten hätte jede Variable den Wert 0 und könnte nicht zugewiesen werden (siehe Abbildung 8-4).

Komponententest: GUI

In dem Test werden die Eingabefunktionen und die Knöpfe des Moduls GUI geprüft. Die Eingaben sollen auf der Konsole ausgegeben werden. Nach Betätigung der Knöpf soll ein Rückgabewert in der Konsole ausgegeben werden. Für den Komponententest der GUI wurde das Modul um einige Anweisungen erweitert, die die lokalen Variablen in der Konsole ausgeben.

```
print("Tsoll: ", self.Tsoll.get())
print("Thkabsenk: ", self.Thkabsenk.get())
print("T0hg: ", self.T0hg.get())
print("T1hg: ", self.T0hg.get())
print("t0vz: ", self.t0vz.get())
print("t1vz: ", self.t1vz.get())
print("tminzust: ", self.tminzust.get())
print("Thkueber: ", self.Thkueber.get())
print("hk0tagfuss: ", self.hk0tagfuss.get())
print("hk0tagsteig: ", self.hk0tagsteig.get())
print("hk0nachtfuss: ", self.hk0nachtfuss.get())
print("hk0nachtsteig: ", self.hk0nachtsteig.get())
print("hk1tagfuss: ", self.hk1tagfuss.get())
print("hk1tagsteig: ", self.hk1tagsteig.get())
print("hk1nachtfuss: ", self.hk1nachtfuss.get())
print("hk1nachtsteig: ", self.hk1nachtsteig.get())
print("T0max: ", self.T0max.get())
print("T0min: ", self.T0min.get())
print("Tpeueber: ", self.Tpeueber.get())
print("FAmax: ", self.FAmax.get())
print("FAmin: ", self.FAmin.get())
print("tminsolar: ", self.tminsolar.get())

print("FAModeset: ", self.FAModeset.get())
print("FATempset: ", self.FATempset.get())
print("FATempdef: ", self.FATempdef.get())
print("SYSContr: ", self.SYSContr.get())
```

Abbildung 8-1: Zusatzanweisungen für den Funktionstest der GUI

Abbildung 8-1 zeigt die Anweisungen, die die Variablen in der Konsole ausgeben sollen. „Self.“ bedeutet, dass es sich um eine lokale Variable handelt und mit der Anweisung „.get()“ wird auf das jeweilige Entry-Objekt zugegriffen.

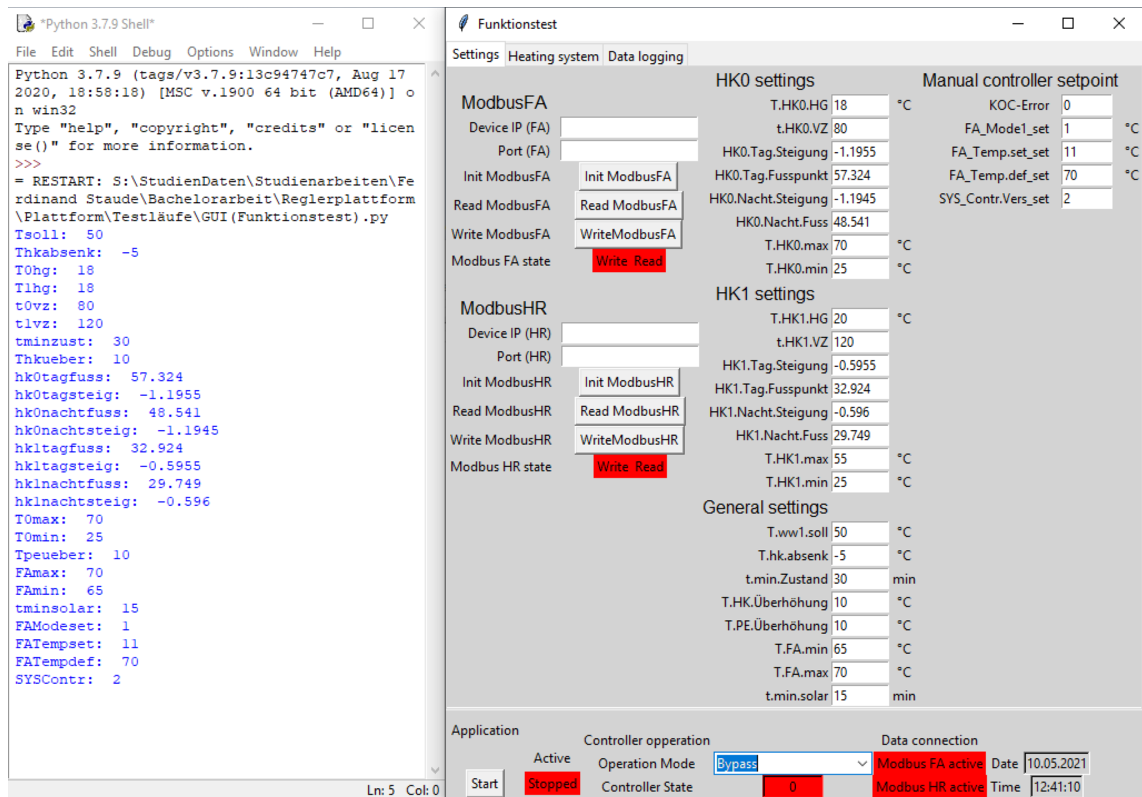

Abbildung 8-2: Funktionstest GUI – Überprüfung der Eingabefunktion

Abbildung 8-2 zeigt, den Reiter „Settings“ auf der rechten Seite und auf der linken Seite die Python-Konsole. Die auf der Konsole angezeigten Werte entsprechen denen, die in den jeweiligen Eingabefeldern in der GUI stehen. Die Reihenfolge der Reglerparameter entspricht nicht der Reihenfolge der GUI, sondern der Reihenfolge, in der die Parameter gespeichert werden.

Um zu testen, ob die GUI die für die Modbus-Steuerung benötigten Werte übergibt, wurden die folgenden Schritte durchgeführt:

1. In die Eingabefelder für IP und Port der jeweiligen Modbusse werden Werte eingegeben.
2. Die Knöpfe „Init Modbus“ werden betätigt, dadurch sollen IP und Port in der Konsole ausgegeben werden.

3. Für beide Modbusse werden die Knöpfe „read“ und „write“ jeweils zweimal betätigt. Dadurch werden die jeweiligen Variablen zuerst auf „True“ und anschießend wieder auf „False“ geschalten. Name und Wert der Variablen sollen auf der Konsole ausgegeben werden.

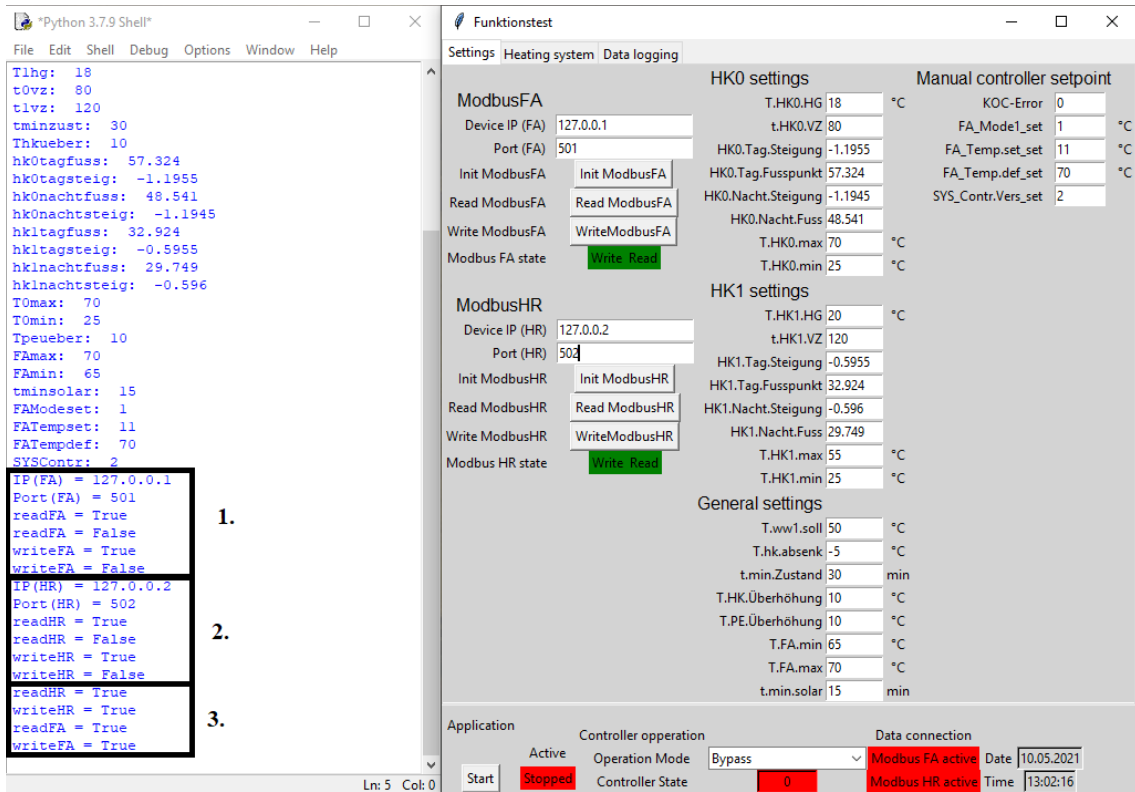

Abbildung 8-3: Funktionstest GUI – Überprüfung der Modbus-Steuerung

Das Ergebnis ist in Abbildung 8-3 zu sehen. In der Konsole sind in dem ersten Rahmen (1.) die Werte von ModbusFA zu sehen. In den ersten beiden Zeilen die IP und der Port, die in den entsprechenden Feldern eingegeben wurden. Anschließend wurden die Knöpfe „Read ModbusFA“ und „Write ModbusFA“ jeweils zweimal betätigt. Das Ergebnis ist in dem ersten Rahmen in den Zeilen 3 bis 6 in der Konsole zu sehen. Zuerst wurde der Knopf „read“ zweimal betätigt. In Zeile 3 hat die Variable den Wert „True“ und in Zeile 4 den Wert „False“. Anschließend wurde der Knopf „write“ zweimal betätigt. Die entsprechende Variable hat in Zeile 5 den Wert „False“ und in Zeile 6 den Wert „True“. Gleiches gilt für die Eingabefelder und Knöpfe von ModbusHR (vgl. Rahmen 2 Abbildung 8-3).

Als nächstes wurden die Knöpfe „read“ und „write“ noch einmal betätigt, damit die Variablen den Wert „True“ haben (vgl. Rahmen 3 Abbildung 8-3). Auf der rechten Seite von Abbildung 8-3 ist zu sehen, dass die Hintergründe der Label „read“ und „write“ im Vergleich zu Abbildung 8-2 jetzt grün sind. Initialisiert werden die Variablen mit dem

Wert „False“. Wie in Abbildung 8-2 zu sehen, ist der Hintergrund der Label in dem Fall rot. Dadurch wird der Wert der Variablen visuell in der GUI dargestellt.

Komponententest: ModbusFA und ModbusHR

Die beiden Module zur Modbus-Kommunikation wurden identisch getestet. Abbildungen zu beiden Tests werden aufgeführt. Die Herangehensweise wird anhand von dem Modul ModbusFA beschrieben. Um die Werte des simulierten Servers zuordnen zu können wird dem „Holding Register 1“ der Wert 1 zugewiesen, dem „Holding Register 2“ der Wert 2, usw. zugeordnet. In Abbildung 8-4 ist der simulierte Server dargestellt.

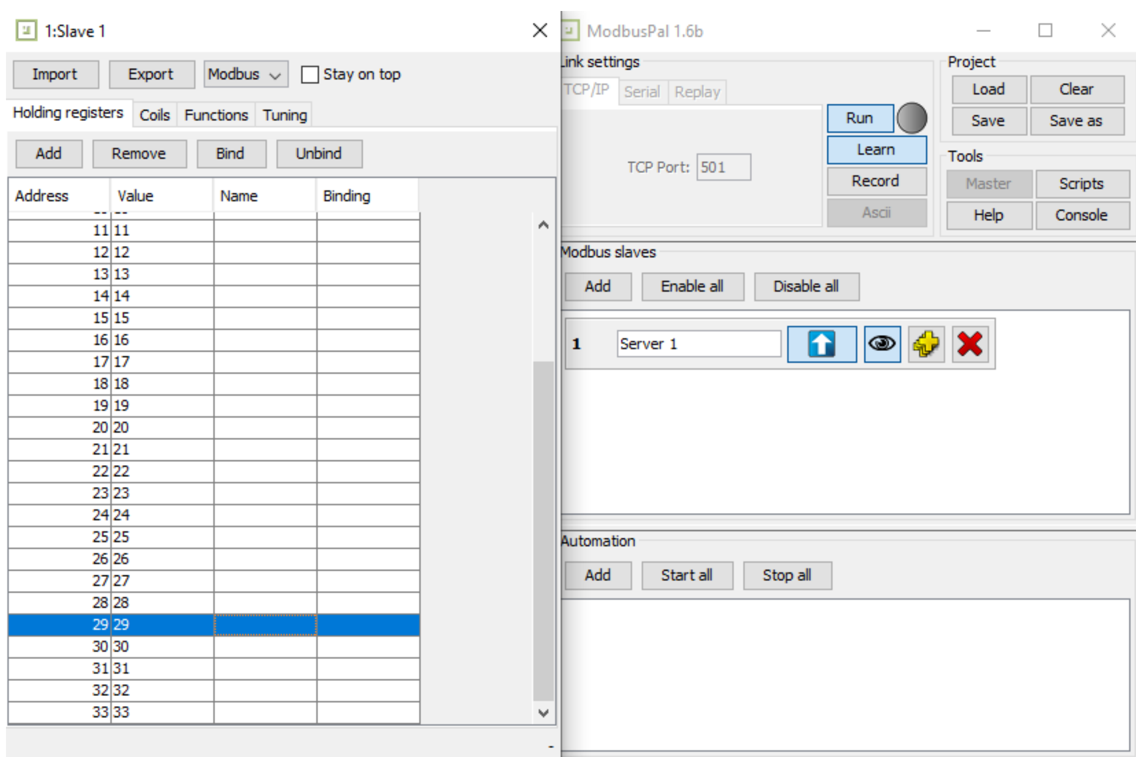

Abbildung 8-4: Funktionstest ModbusFA – HoldingRegister ModbusPal

Der Test der Modbus-Module hatte den folgenden Ablauf:

1. Starten der Anwendung ModbusPal und Eingabe des verwendeten Ports
2. Erzeugung eines Objekts vom Typ ModbusClient
3. Initialisieren eines Clients mit vorgegebenen Werten für IP und Port
4. Lesen des Modbus-Servers
5. Ausgabe des gelesenen Werte in der Konsole
6. Werte auf den Server schreiben
7. Lesen des Modbus-Servers
8. Ausgabe des gelesenen Werte in der Konsole

9. Ausgabe des Modbus-Status in der Konsole
10. Schließen der Modbus-Verbindung
11. Ausgabe des Modbus-Status in der Konsole

```
#Testen
FAClient = ModbusClientFA()

FAClient.initialisieren("127.0.0.1", 501)
FAClient.read_ModbusFA()
print(FAClient.lesewerte)
FAClient.write_ModbusFA(20, 22, 25)
FAClient.read_ModbusFA()
print(FAClient.lesewerte)
print("Status: ", FAClient.statusFA())
FAClient.stop_clientFA()
print("Status: ", FAClient.statusFA())
```

Abbildung 8-5: Funktionstest ModbusFA – Zusatzanweisungen im Quellcode

In Abbildung 8-5 sind die Anweisungen zu sehen, mit denen der Testdurchlauf abläuft. ModbusPal gibt ein visuelles Feedback, wenn eine fehlerfreie Verbindung besteht. In Abbildung 8-6 ist dieses visuelle Feedback, eine grüne Lampe, zu sehen.

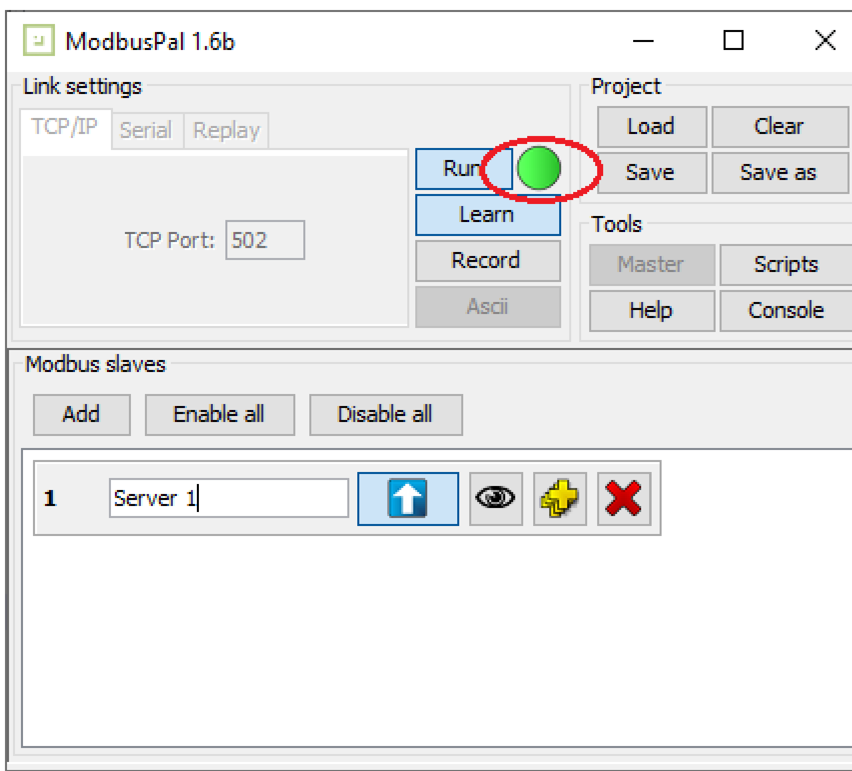

Abbildung 8-6: Funktionstest Modbus – Visuelles Feedback in ModbusPal

ModbusFA

In Abbildung 8-7 ist der Ergebnis des Funktionstests für ModbusFa zu sehen. Die erste Zeile ist eine Liste mit den gelesenen Werten, die zu Beginn in dem Server gespeichert waren. Jede Zahl repräsentiert den jeweiligen Speicherort. In der nächsten Zeile ist die modifizierte Liste, nachdem die Werte auf den Server geschrieben wurden, zu sehen. Das Modul ModbusFA schreibt nur drei Werte auf den Server (siehe Kapitel 4.5.4).

```
= RESTART: S:\StudienDaten\Studienarbeiten\Ferdinand Staude\Bachelorarbeit\Regle
rplattform\Plattform\Testläufe\ModbusFA.py
[11, 12, 13, 14, 15, 16, 21, 2.2, 2.3, 24, 25, 2.6, 32, 33]
[11, 12, 13, 14, 15, 16, 20, 2.2, 22.0, 24, 25, 25.0, 32, 33]
Status:  True
Status:  False
>>>
```

Abbildung 8-7: Funktionstest ModbusFA – Ergebnis in der Konsole

Der Unterschied zwischen der Ausgabe in der ersten Zeile und der Ausgabe in der zweiten Zeile, sind drei Werte. Diese Werte wurden durch die Schreibfunktion des Modbus-Moduls geändert. Die Werte sind in Tabelle 13 aufgeführt.

Tabelle 13: Funktionstest ModbusFA – Änderung zwischen den Ausgaben

Erste Ausgabe	Zweite Ausgabe
21	20
2,3	22,0
2,6	25,0

Bei diesem Test ist aufgefallen, dass es eine Abweichung hinsichtlich der Zuordnung der Holding Register bei ModbusPal gibt. In dem Python-Programm beginnt die Zuweisung mit „HoldingRegister 0“ und bei ModbusPal mit dem „HoldingRegister 1“. Das hat zur Folge, dass die Zuweisung jeweils um eine Stelle versetzt ist. Was in Python „HoldingRegister 0“ ist, ist bei ModbusPal „HoldingRegister 1“ usw. Die Startadresse ist entsprechend der Modbus-Richtlinien nicht fest vorgegeben. Daher kann ein Adressbereich bei 0 oder bei 1 anfangen.

In den letzten beiden Zeilen von Abbildung 8-7 wird der Status zweimal abgefragt. Bei der ersten Abfrage besteht noch eine Verbindung zwischen den Server und Client, weshalb die Ausgabe den Wert „True“ hat. In der letzten Zeile hat wurde die Verbindung geschlossen, weshalb „False“ ausgegeben wird.

ModbusHR

Der Wert der von dem Modul ModbusHR auf den Server geschrieben wird, wird nicht ausgelesen, deswegen gibt es keinen Unterschied zwischen den beiden Konsolenausgaben .Der Unterschied ist bei einem Vergleich zwischen Abbildung 8-8 und Abbildung 8-10 zu sehen. In Abbildung 8-8 hat das „HoldingRegister 1“ den Wert 1 und in Abbildung 8-10 hat das „HoldingRegister 1“ den Wert 20.

```
#Testen
HRClient = ModbusClientHR()

HRClient.initialisieren("127.0.0.1", 502)
HRClient.read_ModbusHR()
print(HRClient.lesewerte)
HRClient.write_ModbusHR(20)
HRClient.read_ModbusHR()
print(HRClient.lesewerte)
print("Status: ", HRClient.statusHR())
HRClient.stop_clientHR()
print("Status: ", HRClient.statusHR())
```

Address	Value	Name	Binding
1	1		
2	2		
3	3		
4	4		
5	5		
6	6		
7	7		
8	8		
9	9		
10	10		
11	11		
12	12		
13	13		
14	14		
15	15		
16	16		
17	17		
18	18		
19	19		
20	20		
21	21		
22	22		
23	23		
24	24		
25	25		
26	26		
27	27		
28	28		
29	29		
30	30		
31	31		
32	32		
33	33		
34	34		
35	35		
36	36		
37	37		
38	38		
39	39		
40	40		
41	41		
42	42		
43	43		
44	44		

Abbildung 8-8: Funktionstest ModbusHR – Zusatzanweisungen und HoldingRegister

```
= RESTART: S:\StudienDaten\Studienarbeiten\Ferdinand Staude\Bachelorarbeit\Reglerplattform\Plattform\Testläufe\ModbusHR.py
[2, 0.3, 4, 6, 7, 8, 9, 10, 12, 1.6, 2.2, 23, 24, 32, 3.6, 4.2, 43, 44, 52, 53, 5.5, 59, 6.1, 6.2, 6.3, 7.3, 7.4, 75, 7.6, 7.9, 81]
[2, 0.3, 4, 6, 7, 8, 9, 10, 12, 1.6, 2.2, 23, 24, 32, 3.6, 4.2, 43, 44, 52, 53, 5.5, 59, 6.1, 6.2, 6.3, 7.3, 7.4, 75, 7.6, 7.9, 81]
Status:  True
Status:  False
>>>
```

Abbildung 8-9: Funktionstest ModbusHR – Ergebnis in der Konsole

In Abbildung 8-9 zeigt analog zu Abbildung 8-7 das Ergebnis der Ausgabe von ModbusHR. Die Listen sind aus den oben genannten Gründen identisch. Das Ergebnis der Status Ausgaben zeigt, dass die Statusfunktion und die Funktion zu Schließen der Modbus-Verbindung Funktionieren.

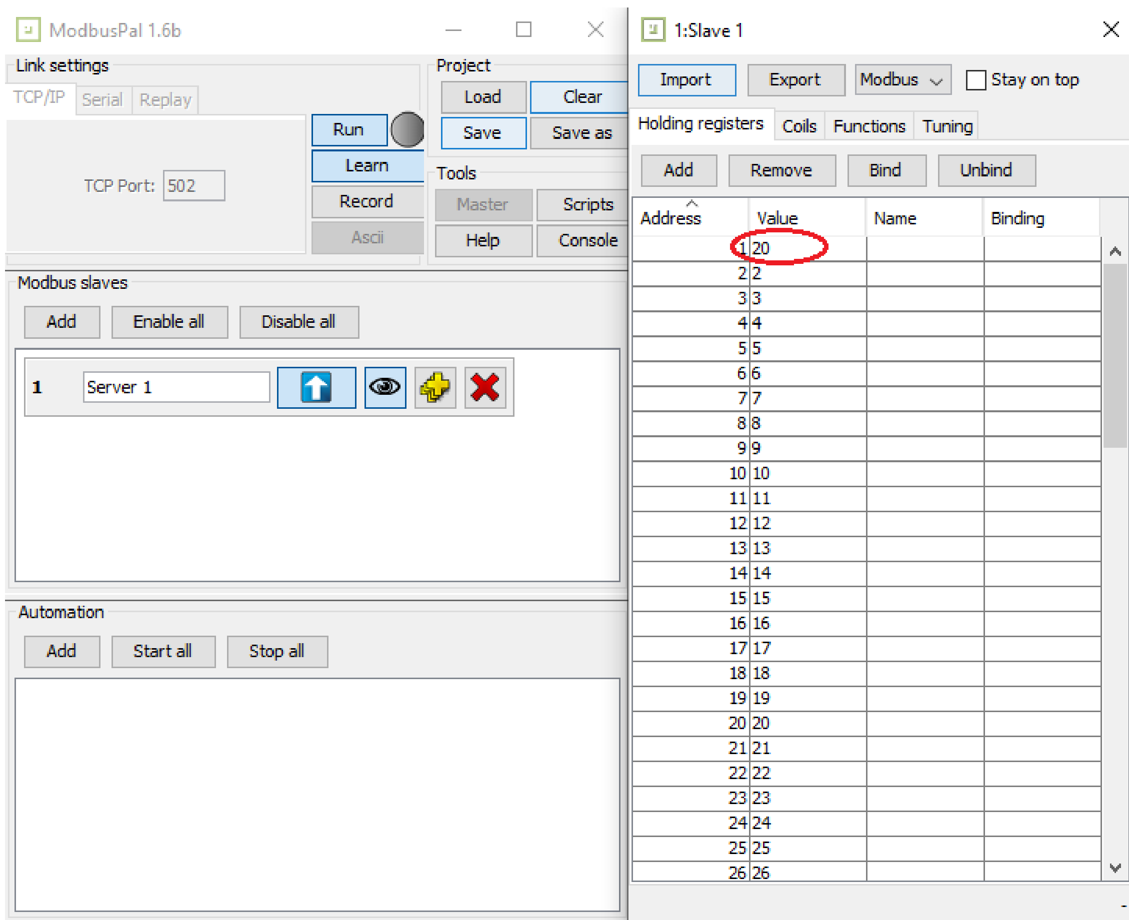

Abbildung 8-10: Funktionstest ModbusHR – Geänderte Werte im HoldingRegister

Anhang B Integrationstests

Bei den Integrationstests wird das Zusammenspiel zwischen zwei oder mehr Modulen getestet. Zuerst wird getestet, ob die GUI die Werte aus den „GlobalenDaten“ entgegennehmen kann und ob die Werte in den „GlobalenDaten“ von der GUI beeinflusst werden. (Abschnitt Integrationstest GUI)

Anschließend werden die „Regler-Modbus-Schnittstelle“ die „GlobalenDaten“ und die Modbus-Module getestet. Dafür wird geprüft, ob die „Regler-Modbus-Schnittstelle“ die Werte aus den „GlobalenDaten“ lesen und mit den Modbus-Modulen eine Verbindung zu den Servern herstellen kann. (Abschnitt Integrationstest Regler-Modbus-Schnittstelle)

In dem nächsten Schritt werden die beiden Threads gestartet und es wird geprüft, ob sich über die GUI eine Verbindung zu den Servern herstellen lässt und ob die gelesenen Werte auf der GUI angezeigt werden. (Abschnitt Integrationstest Reglerplattform-ohne Regler)

In dem abschließenden Test wird der Regler in die Plattform integriert und mit einem Beispielschritt aus der Praktikumsarbeit vergleichen. Mit der Datalogging-Funktion werden die Daten extern gespeichert und ausgewertet. (Anhang C GesamttestGesamttest)

Integrationstest GUI

Mit dem Integrationstest für die GUI wird getestet, ob die GUI und die „GlobalenDaten“ einwandfrei miteinander kommunizieren. Dafür wird jedem Wert in der Liste „DataArray“ der Wert zugeordnet, der seiner Position in der Liste entspricht.

Tabelle 14: Werte für den Integrationstest der GUI

Name	Wert	Name	Wert
KOC_Mode	0	HC2_HC.Current-Flow-Temp.Set	36
KOC_State	1	HC2_HC.Current-Flow-Temp	37
DWD_State	2	HC2_HC.Pump	38
DWD_Date	3	DHW_DHW.Mode_act	39
FA_Error.1	4	DHW_DHW.Force.Once_act	40
FA_Error.2	5	DHW_DWH.Set.Temp_act	41
FA_Error.3	6	DHW_DHW.State	42
FA_Error.4	7	DHW_DHW.On.Temp_act	43
FA_Error.5	8	DHW_DHW.Off.Temp_act	44
Ext.Contr_act	9	DHW_DHW.Temp.Set	45
FA_Mode1_act	10	ACCU_ACCU.TPO	46
FA_Temp	11	ACCU_ACCU.TPM	47
FA_Temp.set_act	12	ACCU_ACCU.State	48
FA_Mod	13	Solar_Collector.Temp	49
FA_State	14	Solar_Storage.Temp	50
FA_Temp.def_act	15	Solar_Solar.Pump	51
FA_Runtime	16	SYS_Contr.Vers_set	52
FA_Starts	17	CNT_Q.Kessel	53
FA_Mode1_set	18	CNT_TV.Kessel	54
FA_Temp.set_set	19	CNT_TR.Kessel	55
FA_Temp.def_set	20	CNT_dT.Kessel	56
SYS_Contr.Vers_act	21	CNT_qV.Kessel	57
SYS_T.Amb	22	CNT_Q.HK1	58
SYS_Plant.Mode_act	23	CNT_TV.HK1	59
SYS_Error 1	24	CNT_TR.HK1	60
SYS_Error 2	25	CNT_dT.HK1	61
SYS_Error 3	26	CNT_qV.HK1	62
SYS_Error 4	27	CNT_Q.HK2	63
SYS_Error 5	28	CNT_TV.HK2	64
HC1_HC.Mode_act	29	CNT_TR.HK2	65
HC1_HC.State	30	CNT_dT.HK2	66
HC1_HC.Current-Flow-Temp.Set	31	CNT_qV.HK2	67
HC1_HC.Current-Flow-Temp	32	ALM_Error.HR	68
HC1_HC.Pump	33	ALM_Temp1	69
HC2_HC.Mode_act	34	ALM_Temp2	70
HC2_HC.State	35	ALM_Temp3	71

Abbildung 8-11: Integrationstest GUI[5]

In Tabelle 14 ist die Liste „DataArray“ mit den jeweiligen Werten aufgeführt, die den Variablen bei diesem Test zugewiesen wurden. In Abbildung 8-11 ist der Reiter „Heating system“ zu sehen. Dieser Reiter zeigt die aus den „GlobalenDaten“ gelesenen Werte an. Bei einem Vergleich der Werte aus Tabelle 14 mit den Werten, die in dem Reiter angezeigt werden, ist zu sehen, dass die Werte an der richtigen Stelle angezeigt werden.

Anschließend wird noch der Zugriff der auf „GlobaleDaten“ getestet. Dazu werden wie in 8 bei den Modbus-Tests print-Anweisungen in den Quellcode eingebaut.

[5] Die Werte in den Klammern zeigen die Position des HoldingRegisters in den Servern und nicht die Position in dem „DataArray“ an.

```
def readFA(self):
    if GlobaleDaten.FAread == True:
        GlobaleDaten.FAread = False
    elif GlobaleDaten.FAread == False:
        GlobaleDaten.FAread = True
    print("readFA: ", GlobaleDaten.FAread)

def writeFA(self):
    if GlobaleDaten.FAwrite == True:
        GlobaleDaten.FAwrite = False
    elif GlobaleDaten.FAwrite == False:
        GlobaleDaten.FAwrite = True
    print("writeFA: ", GlobaleDaten.FAwrite)

def reinFA(self):
    GlobaleDaten.IPFA = self.ipfa.get()
    GlobaleDaten.PortFA = self.portfa.get()
    GlobaleDaten.FAinit = True
    print("IPFA: ", GlobaleDaten.IPFA, "PortFA: ", GlobaleDaten.PortFA, "FAinit: ", GlobaleDaten.FAinit)
```

Abbildung 8-12: Integrationstest GUI – verwendeter Quellcode

In Abbildung 8-12 ist ein Teil des Quellcodes aus dem Integrationstest der GUI zu sehen. Mit „GlobaleDaten." wird, im Gegensatz zu den print-Anweisungen in Abbildung 8-5, nicht auf die lokalen Variablen, sondern auf die Variablen aus „GlobaleDaten" zugegriffen.

```
= RESTART: S:\StudienDaten\Studienarbeiten\Ferdinand Staude\Bachelorarbeit\Regle
rplattform\Plattform\Testläufe\GUI.py
IPFA:  127.0.0.1 PortFA:  501 FAinit:  True
readFA:  True
writeFA:  True
IPHR:  127.0.0.2 PortHR:  502 HRinit:  True
readHR:  True
writeFA:  True
writeFA:  False
readHR:  False
writeFA:  False
readFA:  False
>>>
```

Abbildung 8-13: Integrationstest GUI –Konsolenausgabe

Abbildung 8-13 zeigt die Konsolenausgabe des Tests. Zu sehen ist ein ähnliches Ergebnis wie in Abbildung 8-3. Alle Variablen werden korrekt übergeben und ändern nach mehrmaliger Betätigung der Modbus-Steuerungs-Knöpfe auch ihre Werte.

Integrationstest Regler-Modbus-Schnittstelle

In dem ersten Test wird überprüft, ob die Regler-Modbus-Schnittstelle Werte aus den „GlobalenDaten“ entgegennehmen und an die Module ModbusFA und ModbusHR weitergeben kann. Überprüft wird das, indem Variablen in „GlobalenDaten“ so vorgegeben werden, dass die Modbusse initialisiert werden (InitModbus = True). Weiterhin sollen die Modbus-Module die Funktionen „lesen“ und „schreiben“ ausführen. Zum Testen der Modbus-Module wird immer die Modbus-Server-Simulation ModbusPal benötigt.

```
#Testen
app = Interface()
while True:
    print("Vor dem Schreiben: ", GlobaleDaten.DataArray)
    app.getGlobaleDaten()
    app.main()
    print("Nach dem Schreiben: ", GlobaleDaten.DataArray)
    sleep(2)
```

Abbildung 8-14: Integrationstest: Regler-Modbus-Schnittstelle –Quellcode

In Abbildung 8-14 ist der Quellcode abgebildet, der für den ersten Test der Regler-Modbus-Schnittstelle verwendet wurde. In Anlehnung an die fertige Plattform wurde der Test innerhalb einer While-Schleife durchgeführt. Damit wurde getestet, ob die Module auch nach mehreren Wiederholungen stabil bleiben. Das „DataArray“ wird, bevor die Modbusse schreiben und nachdem die Modbusse geschrieben haben, in der Konsole ausgegeben. Dadurch ist die Veränderung an „DataArray“, durch die Schreibfunktion erkennbar. Zur Übersichtlichkeit wurde die Liste „DataArray“ wie in Anhang B „GUI“ verwendet (Den Werten in der Liste wurde der Wert zugewiesen, der ihrem Speicherort in der Liste entspricht). Zusätzlich wurden die Modbus-Einstellungen in „GlobaleDaten“ so angepasst, dass die Clients initialisiert werden und die Funktionen „lesen“ und „schreiben“ aktiv sind (siehe Abbildung 8-15)

```
#Modbuseinstellungen
FAinit = True
FAread = True
FAwrite = True
HRinit = True
HRread = True
HRwrite = True
```

Abbildung 8-15: Integrationstest Regler-Modbus-Schnittstelle Modbuseinstellungen

In Abbildung 8-16 ist die Rückgabe der Konsole zu sehen. „Vor dem Schreiben:“ meint den Zeitpunkt, bevor die Modbus-Module „schreibend“ oder „lesend“ auf den Server zugreifen. Zu sehen ist, dass die zuerst ausgegebene Liste eine fortlaufende Nummerierung von 0-71 aufweist. Da die Liste „DataArray“ über 72 Einträge verfügt, ist das der zu erwartende Zustand. „Nach dem Schreiben:“ also nach dem Ablauf der Schnittstellenlogik wird die Liste „DataArray“ erneut ausgegeben. In Abbildung 8-16 ist zu sehen, dass diese Liste verändert wurde. Die Veränderung in der Liste entspricht der Veränderung, wie sie nach Tabelle 8 und Tabelle 9 zu erwarten waren.

```
= RESTART: S:\StudienDaten\Studienarbeiten\Ferdinand Staude\Bachelorarbeit\Regle
rplattform\Plattform\Testläufe\Integrationstest RMS\Regler_Modbus_Schnittstelle.
py
Vor dem Schreiben:  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 1
7, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 3
7, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 5
7, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71]
Nach dem Schreiben:  [0, 1, 2, 3, 11, 12, 13, 14, 15, 16, 21, 2.2, 2.3, 24, 25,
2.6, 32, 33, 18, 19, 20, 2, 0.3, 4, 6, 7, 8, 9, 10, 12, 1.6, 2.2, 23, 24, 32, 3.
6, 4.2, 43, 44, 52, 53, 5.5, 59, 6.1, 6.2, 6.3, 7.3, 7.4, 75, 7.6, 7.9, 81, 52,
53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71]
>>>
```

Abbildung 8-16: Integrationstest Regler-Modbus-Schnittstelle – Konsolenausgabe

Integrationstest Reglerplattform – ohne Regler

In diesem Test werden die beiden Threads gestartet und getestet, ob beide miteinander kommunizieren. Durch das Ausführen von „Startanweisung“ werden zwei Threads erzeugt. Zuerst öffnet sich die GUI.

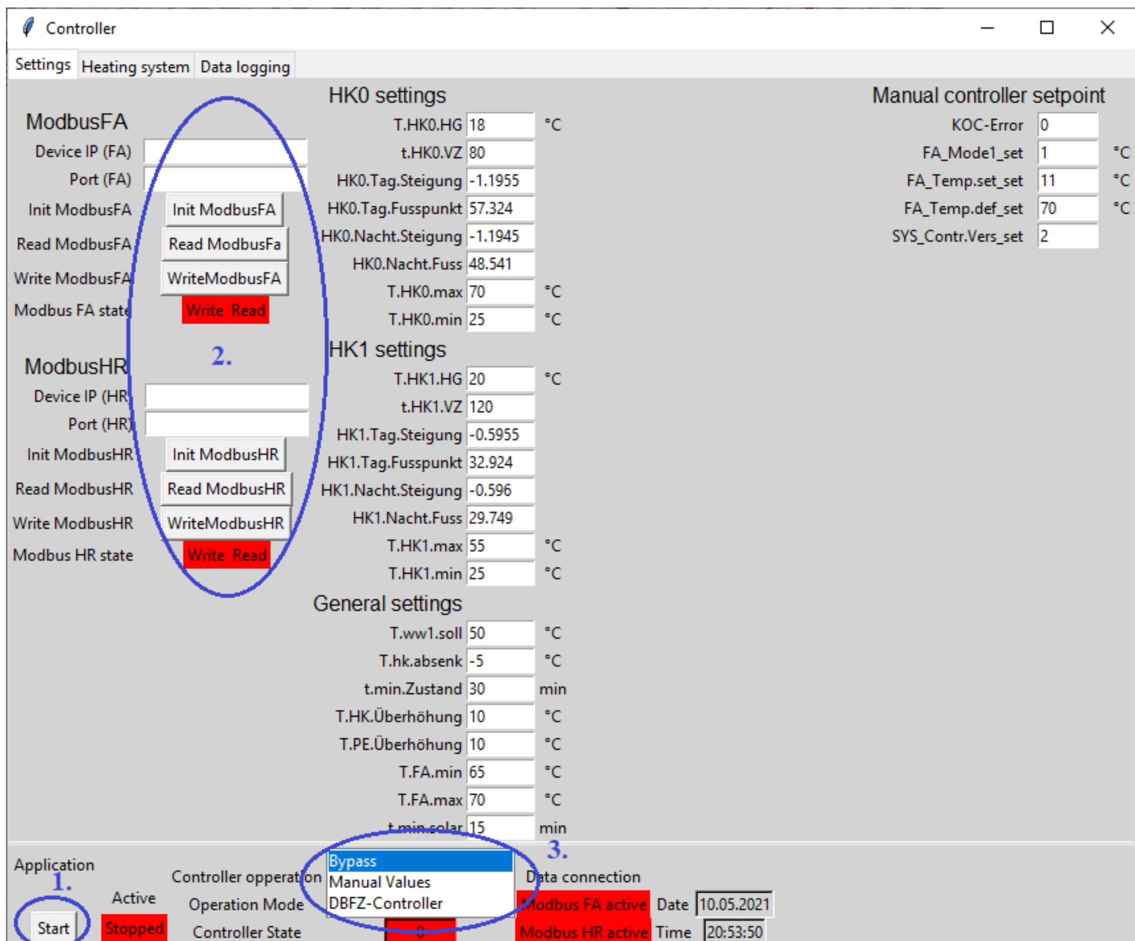

Abbildung 8-17: Integrationstest – ohne Regler GUI Reiter: „Settings“

In Abbildung 8-17 ist die GUI zu sehen. Um die Schnittstelle zu starten, muss jetzt zuerst der Startknopf (1.) betätigt werden. Das Feld „Stopped“ wird grün und der Text ändert sich zu „Running“. Damit ist die Schnittstelle gestartet. Operation Mode (3.) bleibt vorerst in Bypass. Ist die Schnittstelle gestartet, kann die Modbus-Kommunikation gestartet werden (2.). In die entsprechenden Felder werden die Parameter für die IP und die Ports eingegeben. Die lokale IP für die Loopback-Kommunikation ist „127.0.0.1“. Der Port kann frei gewählt werden, er muss aber mit dem Port in der ModbusPal-Anwendung übereinstimmen. Typischerweise werden aber 502 und Werte in Nähe (501, 503) als Portnummern gewählt.

Sind Port und IP eingegeben, wird mit Betätigung des Knopfes „Init ModbusFA“ bzw. „Init ModbusHR“ die Verbindung zu dem Server hergestellt. Kann eine Verbindung hergestellt werden, wechselt der Hintergrund der Felder „Modbus FA active“ und „Modbus HR active“ zu grün. Durch das Betätigen der Knöpfe „Read ModbusFA“, „Write ModbusFA“, „Read ModbusHR“ und „Write ModbusHR“ werden die Lese- und die Schreibfunktion der beiden Modbus-Clients aktiviert.

Abbildung 8-18: Integrationstest – ohne Regler GUI Reiter „Heating System“

In der Fußzeile von Abbildung 8-18 ist die oben beschriebene Veränderung zu sehen. Die Initialwerte der Liste „DataArray“ sind identisch mit den Werten, sie in Tabelle 14 verwendet wurden. Ein Vergleich von Abbildung 8-18 mit Abbildung 8-11 zeigt die Unterschiede. Neben den geänderten Werten ist ein auffallender Unterschied der Hintergrund der „Setpoints“. In diesen Feldern wird visuell dargestellt, ob sich der Wert der Variable seit dem letzten Zeitschritt geändert hat. Da bisher nur starre Werte eingelesen wurden, waren der „Setpoint“ und der Vergleichswert immer unterschiedlich. Lässt man die Umrechnung außer Acht, fällt außerdem auf, dass die Werte in Anzeigefelder immer um eine Stelle höher sind als die Zahlen in den Klammern. Grund dafür ist, wie bereits erwähnt,

dass ModbusPal mit „HoldingRegister 1“ und Python bei „HoldingRegister 0“ zu zählen beginnt. Anhand der Ergebnisse ist bewiesen, dass die Daten aus ModbusPal korrekt eingelesen und angezeigt werden.

Im nächsten Schritt wird die Funktion des Modus (Manual Values) überprüft. Durch diesen Modus sollen die Werte, die in den Felder rechts oben in den Reiter „Settings“ angezeigt werden, in der Liste „DataArray“ gespeichert werden. Anschließend sollen diese in den HoldingRegistern der Modbus-Server gespeichert werden. Zum Testen dieser Funktion werden in den Feldern, die in Abbildung 8-17 rechts oben zu sehen sind, die Werte aus Tabelle 15 eingesetzt.

Tabelle 15:Integrationstest – ohne Regler Werte Manueller Modus

Name	Wert
KOC_Error	0
FA_Mode1_set	1
FA_Temp.set_set	11
Fa_temp.def_set	70
SYS_Contr.Vers_set	2

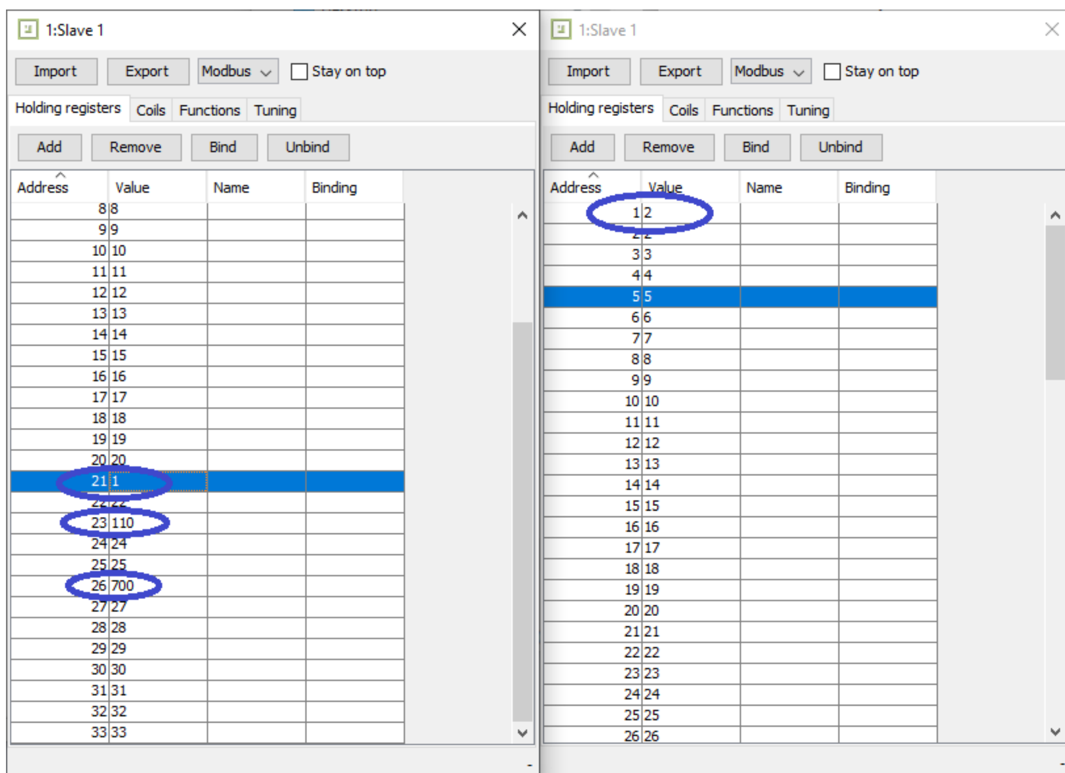

Abbildung 8-19: Integrationstest–ohne Regler – Manueller Modus Ergebnis in ModbusPal

Abbildung 8-19 zeigt die HoldingRegister der Modbus-Server. In den blauen Rahmen sind die Werte markiert, die durch den manuellen Modus in die Server geschrieben wurden.

Abbildung 8-20: Integrationstest – ohne Regler – GUI Manueller Modus

In Abbildung 8-20 ist zu sehen, dass sich diese Werte sich auch in dem Reiter „Heating system“ der GUI wiederfinden lassen.

Anhang C Gesamttest

In dem Gesamttest wurde die gesamte Plattform inklusive des Reglers getestet. In diesem Test werden den Listen und Registern, wie bei den anderen Tests, die Werte zugewiesen, die dem jeweiligen Speicherort entsprechen. Zu Beginn wird die „Regler-Modbus.-Schnittstelle“ aktiviert. Anschließend wird der Speicherort der Logging-Datei ausgewählt und die Logging-Funktion gestartet. Die Schrittweite zwischen den Zeitschritten wurde auf 20 Sekunden festgelegt. Jede Zeile in der Datei entspricht einem den Werten der Liste „DataArray“. In Tabelle 16 ist eine Auswahl der Werte zu sehen.

Tabelle 16: Auswahl vom Ergebnis des Gesamttests

KOC_Mode	KOC_State	FA_Temp_set	FA_Mode1_set	FA_Temp.deft	SYS_T.Amb	DHW_On.Temp_act	ACCU.TPO	ACCU.TPM	Solar.Temp	Uhrzeit
0	0	11	1	70	0,3	6,1	7,3	7,4	7.6	12:50:49
0	0	11	1	70	2	6,1	7,3	7,4	7.6	12:51:08
0	0	11	1	70	2	6,1	7,3	7,4	7.6	12:51:28
0	0	11	1	70	2	6,1	7,3	7,4	7.6	12:51:48
0	0	11	1	70	2	62,1	7,3	7,4	7.6	12:52:09
0	0	11	1	70	2	62,1	7,3	7,4	7.6	12:52:28
0	0	11	1	70	2	62,1	60,9	7,4	7.6	12:52:48
0	0	11	1	70	2	62,1	60,9	61,2	55	12:53:08
2	3	11	2	70	2	62,1	60,9	61,2	55	12:53:29
2	3	11	2	70	2	62,1	60,9	61,2	55	12:53:48

In dem ersten Zeitschritt von Tabelle 16 ist die Modbus-Kommunikation und die manuelle Eingabe von Werten aktiv. In der letzten Spalte ist die Schrittweite von 20 Sekunden des Zeitstempels zu sehen. Die Werte in den Spalten 3 bis 4 zeigen die Werte des manuellen Betriebs an. In den folgenden Zeitschritten wurden die Werte in den Modbus-Servern geändert. Da die Eingabe der Werte länger als 20 Sekunden dauerte sind die Änderungen der Modbus-Werte über einige Zeilen verteilt. In Zeile 2 wird die Außentemperatur (SYS_T.Amb) auf 2 °C gesetzt. In Zeile 5 wird die Warmwassertemperatur (DHW_On.Temp_act) auf 62,1 °C gesetzt. In der 7. Zeile wird die obere Pufferspeichertemperatur (ACCU.TPO) auf 60,9 °C gesetzt. Die mittlere Pufferspeichertemperatur (ACCU.TPM) und die Temperatur der Solarthermieanlage (Solar.Temp) werden in der 9. Zeile auf 61,2 °C und 55 °C gesetzt. Um die Steuerungsfunktion der Plattform zu aktiveren wird in dem Drop-down-Menü das in Abbildung 8-17 zu sehen ist, „DBFZ-Controller" ausgewählt. Das Ergebnis der eingeschalteten Steuerungsfunktion ist in der 9. Zeile von Tabelle 16 zu sehen. (FA.Mode1_set) wechselt auf 2 und „KOC_Mode" hat den gleichen Wert. (KOC_State) hat den Wert 3 was bedeutet, dass die dynamische Ermittlung der Feuerraumtemperatur aktiv ist.

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Alle sinngemäß und wörtlich übernommenen Textstellen aus fremden Quellen wurden kenntlich gemacht.

____________, den ______________

____