

**Fakultät für Elektrotechnik und Technische Informatik (ETTI)
Institut für Verteilte Intelligente Systeme ETTI 2**

Konzeption und Realisierung eines EEBUS Community Converters

BACHELORARBEIT


erstellt an der Professur für IT-Sicherheit

zur Erlangung des akademischen Grades Bachelor of Engineering

Eingereicht von: Maxim 

Matrikel-Nr.: 

Kontakt:


maxim.schumacher@unimuenchen.de

Betreuer: Prof. Dr. Harald Görl

Beginn: 01.05.2022

Abgabe: 31.08.2022

Erklärung

**Gemäß Beschluss des Prüfungsausschusses für Fachhochschulstudiengänge
der Universität der Bundeswehr München vom 25.03.2010**

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, insbesondere keine anderen als die angegebenen Informationen.

Der Speicherung meiner Arbeit zum Zweck der Plagiatsprüfung stimme ich zu. Ich versichere, dass die elektronische Version mit der gedruckten Version inhaltlich übereinstimmt.

Rastede, den 31.08.2022

A black rectangular redaction mark covering the signature of the student.

Maxim Skrypnyk

Inhaltsverzeichnis

Abkürzungsverzeichnis mit Erklärung	6
Kapitel 1 Einleitung	8
1.1 Motivation	8
1.2 Zielsetzung und Struktur der Arbeit	9
1.3 Geeignete Energiespeicher	10
Kapitel 2 Implementierung des Sungrow Hybrid WR in OpenWB	12
2.1 Einführung in die OpenWB Software und Erläuterung der Schnittstellenarbeit	12
2.1.1 Was ist OpenWB und wie bezieht das EMS seine Messdaten	12
2.1.2 Entwicklung des Testnetzwerks	13
2.1.3 Kommunikationsgrundlage zwischen OpenWB einem Wechselrichter	15
2.1.3 Problemlösung Einbindung des Sungrow Residential WR in OpenWB	17
2.1.4 Implementierung des Sungrow Hybrid HV SH10RT WR in OpenWB	18
2.1.5 Fertigstellen des Testnetzwerks	20
2.2 Verschiedene Lademodi und automatische Phasenumschaltung	21
2.3 Sicherheitsuntersuchung der Kommunikation über Modbus TCP	23
Kapitel 3 Implementierung von „legacy“ Geräten in das EEBUS Protokoll	25
3.1 Was ist EEBUS	25
3.2 Aufbau des EEBUS Protokolls anhand des OSI-Modells	26
3.2.1 SHIP das Transportprotokoll	27
3.2.2 SPINE das Informationsprotokoll	28
3.2.3 Umsetzung der Eigenverbrauchsoptimierung	29
3.3 Native EEBUS Nutzung zur Eigenverbrauchsoptimierung	30
3.4 EEBUS „Use Cases“ die mit dem Netzwerk umgesetzt werden sollen	31
3.4.1 Überwachung der Netzeinspeisung	32
3.4.2 Überwachung und Steuerung der Wallbox	33
3.5 Kommunikationsschema	34
3.5.1 Der Energiemanager und die EEBUS Adapter	35
3.5.2 Kommunikationskonzept des EEBUS Client Converters	35
3.6 Implementierung	37
3.6.1 Grundlage	37
3.6.2 Modbus Register mit Python nutzen	37
3.6.3 Modbus Register des WR auslesen	39
3.6.4 Modbus Register der WB beschreiben und kontrollieren	41
3.7 Grundlage der Kommunikation mit den EEBUS Convertern	45

3.7.1 Aufbau einer genormten JSON Instanz	47
3.7.2 Aufbau der Echtzeitfunktion	49
3.7.3 Ergänzungen für den Betrieb mit dem „grid_server“	50
3.7.3 Ergänzungen für den Betrieb mit dem „emobility_server“	53
3.8 Bewertung der Implementierung	54
Kapitel 4 Schlussbetrachtung	55
4.1 Zusammenfassung	55
4.2 Resümee der Implementierung	55
4.3 Resümee der gesteckten Ziele	56
4.4 Ausblick	56
Kapitel 5 Anhang	59
5.1 Abbildungsverzeichnis	59
5.2 Programmcode Verzeichnis	61
5.3 Internetquellen	62
5.4 Quellen	64

Abkürzungsverzeichnis mit Erklärung

EMS -> Energie Management System -> steuert die Verteilung von PV-Überschuss Energie

CEM -> „Custom Energie Manager“ ist ein EMS das die EEBUS EMS „Use Cases“ umsetzt

WR -> Wechselrichter

WB -> Wallbox

EEBUS -> Kommunikationsschnittstelle als gemeinsame und herstellerübergreifende Sprache für Energiemanagement im „Internet of Things“

OpenWB -> Ein Wallbox Hersteller, der sich zum Ziel gesetzt hat, unter Zuhilfenahme der Community alle offenen ansprechbaren Geräte wie E-Autos, Wallboxen, Wechselrichter, Batteriespeicher und Smart Home Geräte auszulesen und wenn möglich diese auszulesen und/oder zu steuern. Die dazugehörige Software steht als Open Source Projekt auf GitHub für den Raspberry PI zur Verfügung. Quelle: <https://github.com/snaptec/openWB>

EVCC -> Eine Open Source Software die noch umfangreicher als OpenWB zur Eigenverbrauchsoptimierung plattformübergreifend genutzt werden kann. Hierbei kann fast jede Grundhardware zum Betreiben verwendet werden. EEBUS befindet sich als Open Source Projekt im Implementierungsprozess. Quelle: <https://evcc.io/>

„legacy“ Geräte -> Ein „legacy“ Gerät ist Hardware, die veraltet und/oder nicht mehr in Produktion ist. Dies schließt alle Geräte ein, die nicht unterstützt werden oder von den meisten Geräten und Softwareanwendungen nicht mehr häufig verwendet werden.

Lademodus/modi -> Einstellung/-en nach der das E-Auto geladen wird. In dieser Abhandlung fokussiert sich der Lademodus auf die Nutzung der Solarenergie im Verhältnis zur netzbezogenen Energie. Hierbei steht die Wahl zwischen wirtschaftlichem oder schnellem Laden im Vordergrund.

Powermeter -> Ungeeichtes Messgerät das schaltungstechnisch hinter dem Zweirichtungszähler im Sicherungskasten sitzt. Dieser besitzt für den Datenaustausch eine meist physische Verbindung zum WR. Das Powermeter ermittelt verlustarm Spannung und Strom der drei Phasen und kann daraus die anliegende Leistung berechnen. Außerdem ist es in der Lage die Richtung des Stromflusses zu ermitteln, um unter Energiebezug (Einkaufen des Stroms) und Einspeisung (Verkaufen des Stroms) unterscheiden zu können.

Smartmeter -> Smartmeter sind Powermeter, die durch ein EMS ergänzt wurden. Das Gerät wird in den meisten Fällen via Ethernet ins Netzwerk angebunden.

GUI (Graphical User Interface) -> Grafische Bedienoberfläche als Ein- und Ausgabeschnittstelle

AC (Alternating Current) -> Wechselstrom

DC (Direct Current) -> Gleichstrom

SOC (state of charge) -> Ladezustand einer Batterie in %

Kapitel 1 Einleitung

1.1 Motivation

Im Laufe der Haussanierung des Autors legte er sich im III. Quartal 2021 eine Solaranlage mit 10.5 kWp¹ und einen 11 kWh Batteriespeicher zu. Die Ausrichtung des Daches ist südlich mit 48% Dachneigungswinkel.

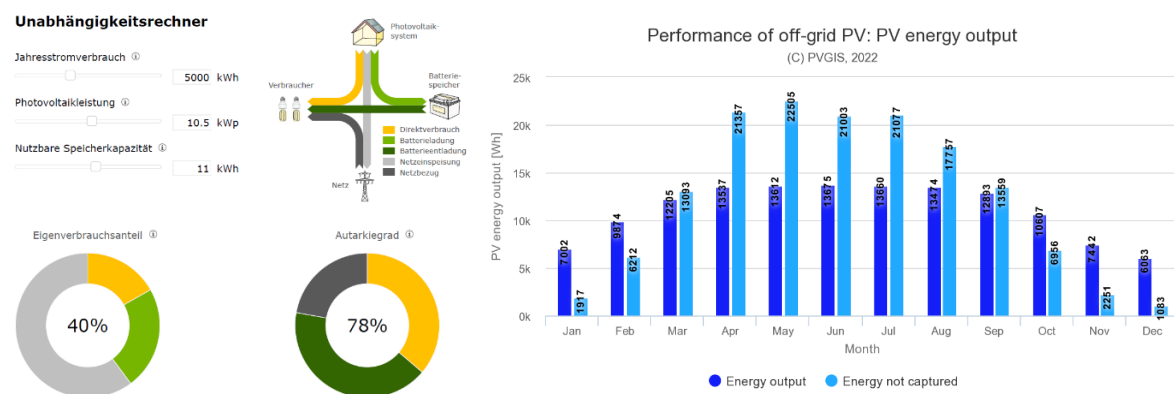


Abbildung 1 Unabhängigkeitsrechner HTW Berlin „Off-Grid“ Berechnung PVGIS EU-Tool

Betrachtet man die reinen Rohdaten der Unabhängigkeitsrechner zu dieser Konstellation, fallen zwei Merkmale auf. Erstens die Autarkie, die ist Dank der großen Solaranlage und des Batteriespeichers mit 78% sehr hoch. Gleichzeitig wird aber nur 40% des erzeugten Stroms selbst verbraucht. Dies begründet sich darin, dass der Strom, sobald er die Grundlast des Hauses abdeckt, eingespeist wird und somit nicht mehr zum Eigenverbrauch zur Verfügung steht.² Eine Optimierung des Eigenverbrauchsanteils ist das Grundmotiv dieser Bachelorarbeit. Die Verlustbilanz liegt bei einer EEG³-Vergütung von angenommen 0,07€ pro eingespeister kWh und Energie-Einkaufskosten von 0,25€ pro kWh bei einer Differenz von 0,18 € pro kWh.⁴ Jede eigenverbrauchte Energie der Solaranlage ist folglich wirtschaftlicher als die Einspeisung.

Zur Steigerung des Eigenverbrauchsanteils ist ein Verständnis der Solarertragszeiträume unerlässlich. Die Wirtschaftlichkeitssteigerung fängt damit an den Geschirrspüler oder die Waschmaschine bzw. den Trockner erst zu solarertragsreichen Zeiträumen einzuschalten. Diese Zeiträume sind um die Mittagszeit herum und/oder bei klarem Himmel mit Sonnenschein. Meist reichen diese Grundregeln schon aus um die Geräte ohne zugekaufte elektrische Energie betreiben zu können.

¹ Kilowatt -Peak: gibt die maximal mögliche Leistung einer Solaranlage an

² Die Verbraucherschutzzentrale belegt, dass „Solar Clouds“ in der Regel teurer sind als die normale Einspeisung. Quelle: <https://www.verbraucherzentrale.de/wissen/energie/preise-tarife-anbieterwechsel/stromclouds-spezialtarife-fuer-prosumer-haben-ihren-preis-56743> letzter Zugriff am 01.05.2022

³ Erneuerbare Energiengesetz

⁴ Dr. Harry Wirth, Fraunhofer ISE, S. 7–11.

Doch die signifikant ausschlaggebende Eigenverbrauchssteigerung sucht man hier vergebens.⁵

Der Hauptteil der Energie wird um die Mittagszeit erzeugt, wenn der durchschnittlich berufstätige Mensch sich am Arbeitsplatz (08-16 Uhr) befindet und diese Solarenergie dementsprechend nicht aktiv genutzt werden kann.

Ein intelligentes EMS wäre in diesem Zusammenhang erstrebenswert, um Verbraucher mit großer Leistungsaufnahme nur mit Solarenergie nutzerunabhängig zu diesen Zeiten betreiben zu lassen. Diese Verbraucher müssen in der Lage sein die zur Verfügung stehende überschüssige Energie sinnvoll zu nutzen. Ein weiterer Aspekt ist, dass die Energie in überschüssiger elektrischer Form in andere Energieformen umgewandelt werden kann, um so zum späteren Zeitpunkt zur Verfügung zu stehen. Als potenzielle Kandidaten für solche Verbraucher lassen sich hier E-Autos, Batteriespeicher, Wärmepumpen oder E-Heizstäbe mit Pufferspeicher, Gartenpumpen zur Gartenwasserbevorratung und Klimaanlage nennen. Zum Zeitpunkt der Forschung stand nur ein E-Auto zur Verfügung, daher beschäftigt sich diese Bachelorarbeit mit dem intelligenten Laden eines E-Autos. Dieses Auto besitzt selbst einen 22 kWh großen Akku, der per Wallbox mit bis zu 11 kWh ladbar ist.⁶ Folglich lässt sich hier aufgrund des großen Speicherungspotentials eine potenzielle Eigenverbrauchsoptimierung durchführen.

1.2 Zielsetzung und Struktur der Arbeit

Ziel der Arbeit ist es herauszufinden in welcher Art und Weise das E-Auto zur Eigenverbrauchsoptimierung genutzt werden kann. Dazu ist es wichtig eine Möglichkeit zu finden oder zu erarbeiten, wie das Auslesen des Wechselrichters und das Ansteuern der Wallbox umgesetzt werden können. Wenn diese Kommunikation gewährleistet ist, sollen die verwendeten „legacy“ Geräte in EEBUS implementiert werden.

In Kapitel 2 wird der Ist-Zustand des vorhandenen Systems dargestellt. Es wird im speziellen der Prozess erläutert im OpenWB-EMS einen nicht unterstützten Wechselrichter selbstständig, unter Zuhilfenahme der Herstellerdokumentation, zu implementieren. Es werden detailliert die Schnittstellen des Wechselrichters und der Wallbox zur Kommunikation untersucht, sowie Möglichkeiten aufgezeigt diese durch OpenWB nutzbar zu machen.

⁵ Die Hauptenergie nutzt die E-Heizgruppe und beansprucht diese nur temporär zum Aufheizen und Halten der Zieltemperatur. Etwa 0,2-0,8 kWh pro Waschgang einer Waschmaschine im ECO-Modus. Quelle: <https://www.bewusst-haushalten.at/artikel/eco-programm/> letzter Zugriff am 01.05.2022

⁶ Die Wallbox ist aufgrund der KfW Förderbedingungen für Wallboxen auf 11 kWh reduziert. Das E-Fahrzeug wäre in der Lage bis zu 22 kWh Ladung anzunehmen. Die genaue Leistungsaufnahme der Batterie ist von vielen Faktoren abhängig wie z.B. Temperatur, Ladezustand, Ladeeinrichtung oder Verschaltung der Batteriezellen. Somit liegt keine lineare und einwandfrei prognostizierbare Ladekurve vor.

Quelle: <https://www.adac.de/rund-ums-fahrzeug/tests/elektromobilitaet/schnellladen-langstrecke-ladekurven/> letzter Zugriff am 01.05.2022

Hierbei werden verschiedene Einstellungen zur Eigenverbrauchsoptimierung sowie deren Vor- und Nachteile beleuchtet.

Abschließend wird auf einige Sicherheitskernelemente eingegangen, die durch diese Implementierung gefährdet sind.

In Kapitel 3 wird der Prozess erläutert die verfügbaren Schnittstellen der verwendeten Geräte in EEBUS zu implementieren. Zunächst wird auf die grundlegende Betrachtung des OSI-Modells mit Bezug zum EEBUS-Standard eingegangen. Hierbei werden Vor- und Nachteile gegenüber dem OpenWB-Ansatz erläutert. Es wird dargelegt welche EEBUS „Use Cases“ mit den genutzten Geräten umsetzbar sind und welche Daten/Befehle in welchem Format dazu benötigt werden.

Im Hauptteil steht die Implementierung der „legacy“ Geräte. Hierbei wird speziell darauf eingegangen wie der Prozess der Inkludierung der Befehle ablief und auf welche Probleme gestoßen wurde.

Abschließend erfolgt die Funktionsprüfung. Hierbei werden die verwendeten Testverfahren beleuchtet und Überprüfung der Latenzen durchgeführt.

In Kapitel 4 wird als Fazit die Forschungsergebnisse zusammengefasst und erläutere aus meiner Perspektive die Möglichkeiten und Risiken die EEBUS mit sich bringt. Abschließend folgt ein Ausblick wie die konstruktiv sinnvolle Eigenverbrauchssteigerung erfolgen kann und wo die wirtschaftlichen Grenzen dieses Verfahrens, aufgrund der hier durchgeführten Forschung, mit einem EMS bewertet werden.

1.3 Geeignete Energiespeicher

In der Einleitung wurden Geräte genannt, die in der Lage sind, elektrische Energie aufzunehmen und diese für eine spätere Nutzung zu speichern. Dieser Prozess muss im Rahmen der Wirtschaftlichkeitsprüfung näher beleuchtet werden. Als Basis steht der Einkaufspreis für elektrische Energie, dieser mag für verschiedene Verbrauchergruppen und Zeitpunkte unterschiedlich sein, betrug jedoch 2021 im Durchschnitt 32,16 Cent⁷. Zieht man hier nun exemplarisch die für diese Anlage geltende EEG-Vergütung von 7 Cent und einer Solarkostenpauschale⁸ von 9 Cent ab, so entsteht eine Ersparnis von 15 Cent pro kWh selbsterzeugter Energie, die im Eigenheim genutzt wird. Dieser Wert wird durch den Hausverbrauch eingespart, wenn die Energie von der Solaranlage genutzt wird und sie nicht extern eingekauft werden muss. Die Berechnungsgrundlage ist für jede Anlage unterschiedlich, weil der Strompreis sehr stark variieren kann und die EEG-Vergütung sowohl davon abhängt, wann eine Solaranlage in Betrieb genommen wurde, als auch eine Solarkostenpauschale je kWh sehr stark variieren kann.

Wenn Energie nun gespeichert werden soll, muss diese zwangsläufig umgewandelt werden. Bei dieser Umwandlung wird Energie in Form von Wärme frei. Genauso wird Wärme bei Transformationsprozessen generiert.

⁷ Quelle: <https://www.co2online.de/energie-sparen/strom-sparen/strom-sparen-stromspartipps/strompreis/> letzter Zugriff am 07.07.2022

⁸ Näherungswert für die Gesamtkosten der Solaranlage auf Ihre Lebenszeit pro erzeugter kWh

An dieser Stelle wird mit einem Beispiel erläutert, wie viele Umwandlungen es geben kann.

1. DC zu DC (Umwandlung des Solarstroms im WR zu passender Spannung für die Batterie)
2. Aufladen der Batterie (Verlust durch Ladeprozess Umwandlung von elektrischer in chemische Energie)
3. Entladen der Batterie (Verlust durch Entladeprozess Umwandlung von chemische in elektrische Energie)
4. Wandlung DC zu AC (Umwandlung von Gleichstrom zu dreiphasigem Wechselstrom)

Bei der Ladung eines E-Autos mit AC Wallbox geht es nun wie folgt weiter:

5. Wandlung AC zu DC (Umwandlung von dreiphasigem Wechselstrom zu Gleichstrom)
6. Aufladen der Batterie (Verlust durch Ladeprozess Umwandlung von elektrischer in chemische Energie)

Daran sieht man, dass ein großes Verlustpotential, durch die unkoordinierte Nutzung von Energiespeichern, gegeben ist. Dieses kann die eben berechnete Ersparnis sehr stark dezimieren und die Wirtschaftlichkeit der Anlage mindern. Es wichtig zu betrachten, welche Speicherform zu welchem Zeitpunkt die beste ist. So wäre es in diesem Fall ratsamer erst das E-Auto, dann den Batteriespeicher aufzuladen und erst dann den Warmwasserspeicher mit der Heizpatrone aufzuwärmen. Würde man zuerst den Warmwasserspeicher aufladen, so würde über die Transmissionswärmeverluste der Speicher wieder abkühlen und es müsste mit fossilen Brennstoffen zu-geheizt werden. Wie im oben gezeigten Beispiel wäre es ebenfalls unratsam erst den Batteriespeicher aufzuladen, wenn das E-Auto gleichzeitig zur Verfügung stünde. Durch die zusätzliche Wandlung würde die effiziente Nutzung des Solarstroms abnehmen. Dieser kleine Exkurs soll beispielhaft zeigen, wie wichtig es ist ein intelligentes EMS mit eindeutiger Priorisierung zur Wirtschaftlichkeitsoptimierung einzusetzen.

Kapitel 2 Implementierung des Sungrow Hybrid WR in OpenWB

Dieses Kapitel befasst sich mit der Implementierung des Wechselrichters in das OpenWB-EMS. Eingeleitet wird das Kapitel mit dem grundlegenden Aufbau des Testnetzwerkes und der Erläuterung der Funktionen der OpenWB Software, sowie der Erläuterung der verwendeten Schnittstellen. Es wird der Prozess der WR Implementierung an zwei konkreten Fällen erläutert. Die grundlegenden Sicherheitsrisiken dieser Umsetzung werden genannt. Die einzelnen Lademodi werden erläutert und im Rahmen der Eigenverbrauchsoptimierung bewertet.

2.1 Einführung in die OpenWB Software und Erläuterung der Schnittstellenarbeit

Dieses Teilkapitel erläutert den Funktionsumfang der verwendeten Software. Mit dieser Software wird der Grundstein in die Welt der Eigenverbrauchsoptimierung gelegt. Die Möglichkeit sich an einem „OpenSource“-Projekt zu orientieren, bietet viele Vorteile, aber auch Nachteile, die im weiteren Verlauf beleuchtet werden.

2.1.1 Was ist OpenWB und wie bezieht das EMS seine Messdaten

OpenWB ist im Grunde genommen eine OpenSource Software für ARM64 Systeme, die speziell für den Raspberry Pi 3/4 unter Debian Buster entwickelt wurde.⁹ Die Nutzer Schnittstelle ist eine GUI, die unter der IP-Adresse des Gerätes + Namenszusatz¹⁰ aufrufbar ist. Im Kern wird das wirtschaftliche Ziel verfolgt ein E-Fahrzeug mit so viel Solarenergie wie möglich zu laden. Dazu ist von Nöten zu wissen, wie viel überschüssige PV-Energie überhaupt zur Verfügung steht. Die meisten konventionell eingesetzten Wechselrichter mit einer Leistungsaufnahme bis zu 15 kWp verfügen über einen Power- oder Smartmeter. Diese sind mit dem WR gekoppelt und bilden nach außen hin in der Regel eine Einheit. Die Schnittstellenlogik kann sowohl im Smartmeter oder auch im WR verbaut sein, dies ist herstellerabhängig.¹¹ Die Hersteller schaffen Schnittstellen für ihre eigenen Zusatzgeräte oder auch andere Monitoring- oder Steuerungszwecke. Das Problem an der Stelle ist, dass verschiedene Hersteller jeweils eigene Ansätze haben, ihre Schnittstelle nach außen zu implementieren. So sind leider alle WB, WR oder E-Autos mit offenen Schnittstellen nach außen nicht einheitlich in ihrer Kommunikation und somit immer als Fremdgeräte zu betrachten.

⁹ <https://github.com/snaptec/openWB> Code + Installationsanleitung für den Raspberry PI unter Buster letzter Zugriff am 01.05.2022

¹⁰ In Testnetzwerk 192.168.188.90/openWB/web/index.php

¹¹ SMA -> EMS und Schnittstelle im Smartmeter Quelle: <https://www.sma.de/produkte/monitoring-control/sunny-home-manager-20.html> oder Sungrow SH hier Schnittstelle im WR

Hier liegt das Kernproblem der smarten Nutzung des PV-Stroms. Die Schnittstellen zum Auslesen der PV-Überschusswerte sind unterschiedlich implementiert. Nur unter sehr großem Aufwand lassen sich diese unterschiedlichen Schnittstellenspezifikationen zusammentragen, implementieren, testen und auch warten. Eine Mammut-Aufgabe, der sich freie Entwickler und Angestellte von OpenWB seit 2018 stellen.

Durch viele Nutzervorschläge und unbezahlter Eigeninitiative ist mit der OpenWB Version 1.9.265, veröffentlicht am 25.04.2022, bis jetzt folgendes möglich:

WR, Batteriespeicher oder Powermeter auslesen		WB ansteuern	Fahrzeugbatterie auslesen
Alpha ESS	Solarview	OpenWB WB	Aiways
Discovery	Solarwatt	Go-eCharger, Sungrow, Fronius etc. ¹²	Audi
Fronius	Solarworld	Keba	BMW / Mini
Huawei	Solax	NRG-Kick	Kia Hyundai
Kostal	Sonnen Eco	SimplexSEWifi	Mercedes EQ
LG ESS	Sungrow	Tesla	Opel
OpenEMS	Sunways		Renault
Powerdog	BYD	Smart Home Auslesen und Befehlen	Nissan Leaf
RCT	Victron	Shelly / Tasmota / Acthor	PSA
Siemens	Youless	ELWA / IDM / Stiebel / AVM	Tesla
SMA	Carlo Gavazzi		VAG
Smart Me	E3DC	Weitere Features	Volvo
Solaredge	Varta	Ladelog/ Logging	VW
Solarlog	Victron	Lastmanagement am Hausanschluss	Zero NG
Janitza		Umschaltung 3 zu 1 Phasen Laden	Peugeot

2.1.2 Entwicklung des Testnetzwerks

Am Anfang wurde, wie in der Einleitung erwähnt, die Solaranlage mit WR¹³ und Batteriespeicher¹⁴ aufgebaut und das E-Auto¹⁵ erworben. Zum Laden wurde eine 11 kWh Wallbox¹⁶ verwendet. Die Leistungsdaten der PV Anlage konnten über die „MySolarCloud“, eine WEB und APP GUI, eingesehen werden. Hier standen diverse Möglichkeiten der Auswertung zur Verfügung.

¹² Der Go-e Charger der Firma Go-e GmbH wird von diversen Firmen in „gebrandeter“ Version verwendet. <https://go-e.com/de-de/ueber-uns/presse/go-e-intensiviert-zusammenarbeit-mit-fronius> letzter Zugriff am 01.05.2022

¹³ Sungrow Hybrid HV SH10RT WR <https://ger.sungrowpower.com/productDetail/905> letzter Zugriff am 01.05.2022

¹⁴ BYD HVM 11 <https://www.eft-systems.de/de/B-BOX%20PREMIUM/product/Battery%20Box%20HVS-HVM/6> letzter Zugriff am 01.05.2022

¹⁵ Renault Zoe Q210 (Bj. 2013) 23,3 kWh nutzbarer Batteriekapazität durch Degradierung Batterie 21 kWh

¹⁶ go-eCharger HOMEfix 11 kW <https://shop.go-e.co/go-eCharger-HOMEfix-11-kW> letzter Zugriff am 01.05.2022

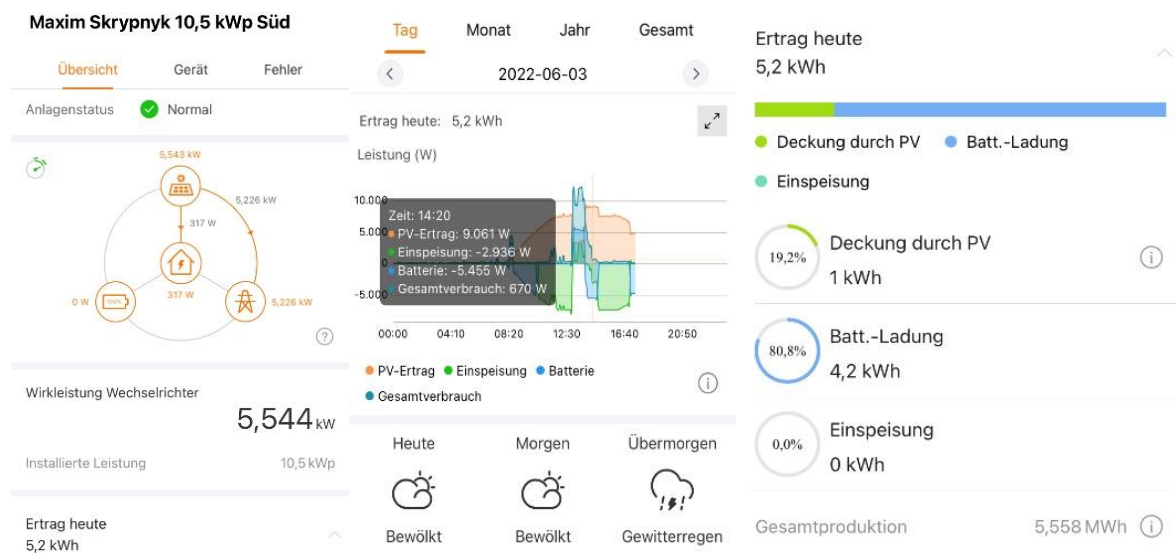


Abbildung 2 Auszug aus der „MySolarCloud“ APP vom 03.06.2022

Gleichzeitig war es möglich, über die go-eCharger APP die Wallbox zu steuern, Ladeströme zu regulieren und die verwendete Energiemenge auszulesen. Zu Beginn wurde eine vereinfachte Effizienzoptimierung versucht. Dabei wurden die Ladeströme der Wallbox so eingestellt, dass sie den PV-Überschusswert aus der „MySolarCloud“ entsprachen und so ein pseudo PV-Laden des Fahrzeugs ermöglicht werden sollte. Leider konnten die Werte nicht direkt übertragen werden, sondern es musste eine Umrechnung von Watt dreiphasig in Ampere einphasig erfolgen. Da Anpassungen des Stroms nur in ein Ampere Schritten möglich waren, ergaben sich feste Grenzwerte für die Umschaltung des Stroms. Das E-Auto benötigt einen dreiphasigen Strom von mindestens 12 A, um den Ladevorgang durchzuführen. Der niedrigste mögliche Ladestrom der Wallboxen für E-Fahrzeuge ist technisch bedingt 6 A. Aus den Vorgaben ergaben sich folgende Transferwerte:¹⁷

Leistung in kW	Strom in A
4,14	6
4,83	7
5,52	8
6,21	9
6,9	10
7,59	11
8,28	12
8,97	13
9,66	14
10,35	15
11,04	16

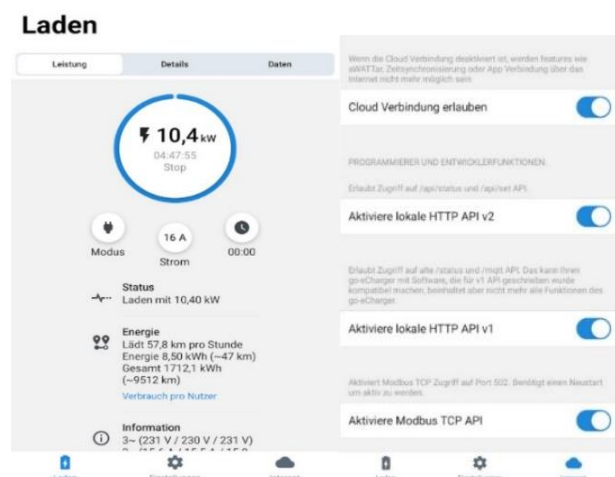


Abbildung 3 Go-eCharger App Oberfläche

¹⁷ Berechnung = kW(dreiphasig) / 3 => kW(einphasig) x 1000 => W(einphasig) / 230V => A(einphasig)

Es fiel auf, dass es eine undefinierbare Latenz einerseits beim Auslesen der PV-Leistung und andererseits beim Umschalten des Ladestroms gab. Durch eine zeitliche Latenz konnten die angezeigten PV Überschusswerte weit von den aktuell vorliegenden abweichen. Dies begründet sich darin, dass PV-Nutzung aufgrund seiner Abhängigkeit von solarer Strahlungsintensität anfällig für Beeinträchtigungen ist. Diese Einflussfaktoren können Bewölkung, Nebel, Luftfeuchtigkeit oder auch im generellen der Winkel zur Strahlungsquelle sein. Folglich verfehlte es den Effekt des angestrebten PV-Überschuss-Ladens.

Aufgrund der Latenzen war eine Nutzung der durch die Cloud zur Verfügung gestellten Daten nicht sinnvoll und es musste eine Möglichkeit gefunden werden lokal Daten auszulesen und zu verwerten. Nach kurzer Recherche wurde die vorher genannte OpenWB Software entdeckt und gemäß der GitHub Anleitung auf dem Raspberry PI installiert. Dabei ergab sich bei der Erstinstallation das Problem, dass das Installationsskript nicht korrekt ausgeführt wurde. Nach intensiver Fehlersuche und dem Verfassen eines Forumseintrags, mit bitte um Unterstützung, wurde schnell klar, dass die falsche Debian Kernel Version installiert wurde. Das Problem wurde durch eine Neuinstallation mit richtiger Debian Kernel Distribution gelöst. OpenWB funktioniert als Software nun einwandfrei. Nach dem Auswählen des hinterlegten Sungrow WR wurden keine Daten in der Software angezeigt.¹⁸ Um zu verstehen, warum das Auslesen der Daten zwischen OpenWB und dem WR nicht zustande kommt, ist ein Verständnis der technischen Abläufe notwendig.

2.1.3 Kommunikationsgrundlage zwischen OpenWB einem Wechselrichter

Die Programmierer der OpenWB-Software benötigen Kenntnis über die Schnittstellen Technologie und die Strukturierung des Modbus Registers, um neue WR in der Software anlegen zu können. Vorprogrammierte Algorithmen allein reichen nicht aus, um eine erfolgreiche Verbindung herzustellen. Der Nutzer muss zwangsläufig die IP-Adresse des Zielgerätes herausfinden und in OpenWB eingeben. Die Aufgabenbereiche splitten sich somit in einen Nutzer- und einen Programmierer-Anteil. Folgende Parameter müssen für eine erfolgreiche und richtige Registerauslesung vorliegen:

1. Die IP-Adresse

Diese wird, bei dynamischen Vergabeverfahren vom Router automatisch vergeben. Diese kann der Nutzer über die GUI des Routers im Regelfall auslesen und der OpenWB-Software zur Verfügung stellen. (Nutzeranteil)

2. Port (optional – Standard: 512)

Die angestrebte Verbindung für den Sungrow WR ist Modbus/TCP. Diese ist standardisiert auf Port 502 für UDP oder TCP Verbindungen. Somit kann der Port vorkonfiguriert werden.

3. Register Nummer/n

Alle registerspezifischen Inhalte legt der Hersteller ohne äußere Konvention fest. Die nötigen Informationen finden sich in den Modbus-Spezifikationen, die die Hersteller grundsätzlich zur Verfügung stellen. Die Arbeit des Programmierers beginnt damit die Dokumentation zu verstehen und zu filtern welche Register die angestrebten Inhalte speichern.

4. Datentyp des Inhalts

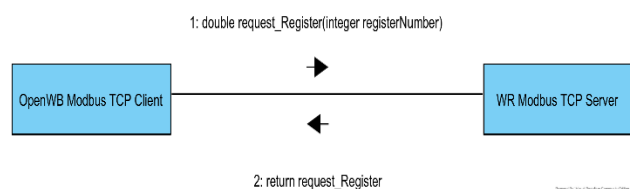
Aus der Dokumentation folgt auch eine Überprüfung des Datentyps, dabei ist wichtig zu erkennen, ob der Wert von einem oder mehreren Registern in aufgesplitteter Form gehalten wird. Nach dem Auslesen des Registers und dem potenziellen Zusammensetzen des Wertes muss dieser gegebenenfalls noch konvertiert werden, um ihn gemäß den OpenWB Konventionen nutzen zu können.

„Modbus TCP ist ein Client/Server-Protokoll für den verbindungsorientierten gesicherten Austausch von Prozessdaten. Dies bietet ein hohes Maß an Flexibilität für dezentrale Automatisierungsarchitekturen: Die Rollen des Clients und des Servers sind nicht fest zugeordnet. Jedes Gerät im Netzwerk kann beide Rollen spielen. Somit können Zugriffe auf Daten – lesend oder schreibend – flexibel den jeweiligen Aufgabenstellungen angepasst werden.

Der Server bearbeitet eine Anfrage des Clients – den so genannten Request – und quittiert die Anfrage mit einer Erfolgsmeldung (Response), der gegebenenfalls angefragte Daten oder Statusinformationen mitgegeben werden, bzw. mit einer Fehlermeldung, die Informationen über die Fehlerursache enthält. Die Bearbeitung dieser Anfrage läuft für den Anwender völlig transparent im Hintergrund ab. Bei üblichen Implementierungen ist kein Applikationsprogramm auf der Client-Seite erforderlich.“¹⁹

	OSI architecture	Modbus protocol
7	Application layer	Modbus
6	The presentation layer	
5	Session Layer	
4	Transport layer	TCP
3	Network layer	IP, ARP, RARP
2	Data link layer	Ethernet, CSMA/CD, MAC
1	Physical layer	Ethernet Physical layer

Abbildung 4 Modbus Read Anfrage



Modbus im OSI-Modell

¹⁹ Zitatquelle https://www.feldbusse.de/Modbus/TCP/Modbus/TCP_protokoll.shtml Zugriff am 01.05.2022

2.1.3 Problemlösung Einbindung des Sungrow Residential WR in OpenWB

Der Solateur der die Solaranlage installiert hatte, verwendete ebenfalls die OpenWB Software in Form eines OpenWB „standalone“ Produkts. Dieses ist nichts anderes als ein Raspberry Pi 3b mit vorinstallierter OpenWB Software und geschlossenem Gehäuse. Der Vorteil gegenüber des eigen „geflashten“ Raspberrys ist, dass Garantieansprüche für dieses Gerät inklusive der Software bestehen.

Der Solateur hatte ebenfalls Probleme mit OpenWB seinen Sungrow SR WR zu betreiben. Bevor der Sungrow SH Wechselrichter in OpenWB einpflegt, werden konnte, musste zunächst einmal geprüft werden warum der eigentlich unterstützte Wechselrichter nicht ausgelesen werden kann.

Es bestanden zwei Probleme.

- 1.) Der WR wurde mit einem WLAN Erweiterungs-Modul ins Gast-WLAN eingebunden, wohin gegen die OpenWB Standalone im normalen Heimnetzwerk eingebunden wurden. Die Trennung der Netzwerke erfolgte durch Subnetze, die die Fritz Box selbständig etablierte. Durch die Trennung der Subnetze konnten die Geräte keine Verbindung zueinander aufbauen. Der Fehler fiel erst bei näherer Betrachtung der IP-Adressen auf.

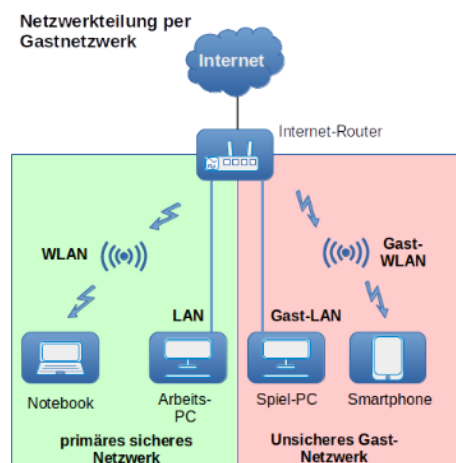


Abbildung 5 Trennung der Teilnetze im Heimnetzwerk

Zur Lösung des Problems wurde der WR statt ins Gastnetzwerk ins Heimnetzwerk hinzugefügt. Nun konnte beim Auslesen der Log-Dateien festgestellt werden, dass eine IP-Verbindung zustande kam. Jedoch wurden immer noch keine Daten an die OpenWB übertragen. Die Anfrage auf Port 502 blieb unbeantwortet und somit wurde die Anfrage durch ein „Timeout“ beendet. Auch erneute Anfragen oder Gerätereustarts führten zu keinem Ergebnis.

- 2.) Dieser Fehler lag in einer fehlenden Freischaltung des TCP Server Sockets begründet. Wenn man die IP-Adresse des WR in den Browser eingibt, öffnet sich die GUI des Lan/WLAN Moduls. Hier war der TCP Socket, über den das Modbus Protokoll arbeitet, ausgeschaltet.

```

AttributeError: 'ModbusIOException' object has no attribute 'registers'
  value1 = resp.registers[0]
File "/var/www/html/openWB/modules/bezug_sungrow/sungrow.py", line 17, in <module>
Traceback (most recent call last):
AttributeError: 'ModbusIOException' object has no attribute 'registers'
  value1 = resp.registers[0]
File "/var/www/html/openWB/modules/wr_sungrow/sungrow.py", line 14, in <module>

```

Abbildung 6 Debug Fehlerrückmeldung der OpenWB

The screenshot displays the configuration page for a Sungrow Sg WR. It is divided into two main sections: 'Device status' on the left and 'Socket Settings' on the right.

Device status:

- Dev-MAC: 60:C5:A8:74:BB:70
- Dev-name: Sungrow
- Firmware version: 1.9.0.10.12-2.13.4
- Access point: ON
 - SSID: SG-A19****2392
 - Encryption: WPA2-AES-PSK
 - IP address: 11.11.11.1
- Station point: ON
 - AP SSID: HomeJS
 - AP BSSID: 00:00:00:00:00:00
 - RSSI: -49
 - Encryption: WPA2-AES-PSK
 - IP address: 192.168.178.93
- Remote connection:
 - SocketA: Not Used
 - SocketB: Connected

At the bottom left, there is a 'Start Wizard' button and a note: '*help: Start Wizard will show you configure steps.'

Socket Settings:

- Open Double Socket: Open (dropdown)
- SocketA Setting / SocketB Setting:
 - SocketA Type: TCP-Server (dropdown)
 - SocketA allow connections num: 3 (dropdown)
 - SocketA ServerIP(Domain): 192.168.1.1
 - SocketA ServerPort: 5000
 - SocketA LocalPort: 502
 - SocketA TCP idle timeout/recon interval: 15 | 3 (S)

A red note at the bottom right states: 'Note: TCPS/C idle timeout,0: disable; TCPC recon interval,0: no interval'. A 'Save' button is located at the bottom right.

Abbildung 7 Auszug aus den Einstellungen des Sungrow Sg WR

Nach dem Einschalten konnte ein reibungsloser Betrieb durchgeführt werden. Die Verbindung konnte einwandfrei hergestellt werden und die Messwerte wurden korrekt in OpenWB dargestellt.

Zusammenfassend wurden die Kommunikationsprobleme durch grundlegende Einbindungs- und Freischaltfehler verursacht. Es lag kein Fehler in der Implementierung vor. Jedoch wurde auch bei Beachtung dieser Punkte der Wechselrichter nicht korrekt ausgelesen.

2.1.4 Implementierung des Sungrow Hybrid HV SH10RT WR in OpenWB

Nach einem Blick ins „Debugging“ von OpenWB wurde klar, dass es wieder Probleme bei der Auslesung der Modbus Schnittstelle gab. Zunächst wurde betrachtet, was die grundlegenden Unterschiede der beiden WR waren. Bei dem SH Gerät wurde ein „Wii Net-S Communication Dongle“ verwendet. Dieser fand an der Unterseite des Gerätes Platz und diente der Einrichtung des WR. Man steckt das Kommunikationsmodul in die RJ45 Buchse mit der Aufschrift WLAN und danach wird dieser per LAN-Kabel an den Router angebunden. Eine Einrichtung nur mit WLAN war ebenfalls möglich. Nach dem Einloggen in die Admin Oberfläche des Dongles wurde angezeigt, dass eine Modbus-Verbindung mit TCP Server bereits zur Verfügung gestellt wurde. Jedoch konnte man hier nun ein „Whitelisting“ durchführen. Dies bedeutet, dass nur IP-Adressen, die dort

eingetragen sind Zugriff auf die Register erhalten können. Eine Sicherheitsmaßnahme gegen Fremdzugriff anderer Geräte.

Nachdem die IP-Adresse der OpenWB dort eingetragen wurde, konnte man erkennen, dass immer noch keine Verbindung möglich war.

Dabei war es Unerheblich, ob die Verbindung kabelgebunden mit dem Router oder per WLAN erfolgte.

Dies wurde genauer untersucht, indem das Tool „Simply Modbus TCP Client“²⁰ installiert wurde.

Dieses Tool generiert einen Modbus TCP Client und kann per Wahl der IP-Adresse und des Ports eine Verbindung zu einem Modbus TCP Server herstellen. Diese Verbindung funktionierte einwandfrei, jedoch konnten keine Register ausgelesen werden.

Nachdem sichergestellt wurde, dass die Einstellungen, denen aus der Modbus Dokumentation entsprachen, wurden die Ergebnisse ausgewertet. Bei allen Registern wurde der Wert „0xFF“ ermittelt. Solch ein Wert in allen Registern entspricht nicht den Tatsachen und somit bestehen nachweislich Probleme in der Auslesbarkeit der Register. Weitere Möglichkeiten das Problem zu lösen, gab es leider nicht. Es standen keine technisch tiefergehenden Informationen zu der Umsetzung des Modbus Servers zur Verfügung. Deswegen wurde ein Ticket für Sungrow geschrieben, in dem das Problem erläutert wurde. Nach der Reproduktion des Fehlers leiteten sie das Ticket mit der Beschreibung der grundlegenden Funktionsstörung an das Entwicklerteam nach China weiter. Das Problem wurde per Firmware Update im August 2022 gelöst. Für die diese Forschung wurde ein „Workaround“ gefunden.²¹

Durch das Umstecken des Lan Kabels in den Lan Port und nicht, wie vorgesehen, in den Adapter, ist die Modbus Funktionalität wieder gegeben. Die Gründe hierfür ließen es sich ohne tiefgreifende Implementierungseinsicht nicht ergründen. Folglich bleibt die Einschränkung, dass Modbus nur über LAN und nicht über WLAN nutzbar ist. Dies ist jedoch schwierig für Personen, die ihren WR an unzugänglichen Orten aufstellen müssen und dort die Verlegung eines LAN-Kabels sehr mühselig wäre. Für weitere Testzwecke wurde am WR Standort im Testaufbau direkt eine LAN-Anbindung mit eingeplant.

Nachdem das Kabel umgesteckt und die Funktionalität der Registerauslesung überprüft wurde, sind unplausible Ergebnisse aufgefallen. Die Register, welche die nötigen Werte des Power-Meters halten waren leer, bzw. nicht belegt. Folglich wurden die Modbus Spezifikationen gegeneinander geprüft und es ließ sich feststellen, dass eine grundlegende Umstrukturierung stattgefunden hat. Die Register, welche die Leistung am Smart-Meter, die Batterieladeleistung und den SOC der Batterie halten,

²⁰<https://www.simplymodbus.ca/TCPclient.htm> letzter Zugriff am 11.07.2022

²¹<https://openwb.de/forum/viewtopic.php?f=11&t=3074&start=30> letzter Zugriff am 11.07.2022

wurden herausgesucht. Das Entwicklerteam von OpenWB legte eine Unterscheidung der Sungrow SG und SH Wechselrichter an und pflegte die gefundenen Register im Code ein. SG Wechselrichter können nur von DC, aus der Solaranlage, in AC des wandeln. Wohingegen SH Wechselrichter zusätzlich Batterien Laden und Entladen können. Es erschien eine neue „Nightly“²²-Version mit den Änderungen. Nach dem Update wurden die Werte ausgelesen und mit denen aus der MySolarCloud verglichen. Der Wert für die Batterieladung sowie das Batterie SOC entsprachen denen aus der App. Jedoch nicht der Wert der Einspeisung/Bezug anzeigt. In der App wird deutlich, wie die Solarenergie sich auf die Batterieladung, den Hausverbrauch und die Einspeisung aufteilen. Aus diesen Werten wurde ersichtlich, dass ein einfacher Vorzeichenfehler in der Open-WB vorlag. Hierbei war der Wert am Smart Meter, ob Einspeisung oder Bezug vorlag, vertauscht. Somit nahm das System immer das Gegenteil an und zeigte bei hoher Solarintensität einen hohen Netzbezug an und umgekehrt. Nachdem dies dem Support Team von Open WB mitgeteilt wurde, konnten sie in weniger Zeit eine neue Nightly Version releasen. In dieser neuen Version wurde der Wert mit „-1“ multipliziert und konnte so korrekt dargestellt werden.

2.1.5 Fertigstellen des Testnetzwerks

Der Go E-Charger ließ sich in OpenWB implementieren. Hierzu reicht es aus bei der Wallbox die Modbus TCP Verbindung in den Einstellungen freizuschalten. Hier wurde dann die IP-Adresse der Wallbox angegeben und die Wallbox nahm die Befehle der OpenWB einfach entgegen. Die Latenz war nicht digital in der Weboberfläche wahrnehmbar und somit bedeutend besser als die Cloud Implementierung aus den ersten Testversuchen.

Um das Batterie-SOC des Fahrzeuges zu überprüfen, wurde es in OpenWB mit den MyRenault Cloud Zugangsdaten hinterlegt. Hierbei handelte es sich um eine Cloud basierte Lösung. Das E-Fahrzeug verfügt über eine Sim-Karte und einer mobilen Datenverbindung, die kontinuierlich Daten an den Hersteller sendet. Der Hersteller wiederum stellt dem Endkunden gegen eine Gebühr, eine App zur Verfügung, in der er den SOC seines Fahrzeugs überprüfen, die Wartungshistorie auslesen und das Auto vorklimatisieren kann. Den Ladezustand SOC liest Open-WB hierbei aus und stellt diesen grafisch dar. Dieser Wert kann ebenfalls genutzt werden, um Ladebegrenzungen festzulegen. Somit lässt sich die Fahrzeugbatterie durch eine begrenzte maximale Aufladung schonen und die Lebenszeit verlängern.²³

Das Testnetzwerk ist an dieser Stelle fertiggestellt. Open-WB verarbeitet korrekt die Daten der Wallbox, des Wechselrichters und des E-Autos. Die ersten Tests bestätigen die Funktionalität.

²² Version mit einer experimentellen Funktionalität, nicht für den normalen Betrieb geeignet (Version 1.9.271)

²³ https://efahrer.chip.de/e-wissen/akku-richtig-laden-so-erhoehen-sie-die-lebensdauer-ihres-e-autos_10757
letzter Zugriff am 11.07.2022

2.2 Verschiedene Lademodi und automatische Phasenumschaltung

Beim angestrebten PV Überschussladen wird der Ladestrom der überschüssigen Energie, welche eingespeist wird, angepasst. Dabei kann es passieren, dass der Ladestrom unter der maximalen Aufladegeschwindigkeit des E-Fahrzeuges liegt. Hierbei werden Ladeleistungen von maximal 11 kW (Sicherung 3x16A) oder in Sonderfällen 22 kW (Sicherung 3x32A) berücksichtigt. Deswegen verlängert sich die Ladedauer des E-Fahrzeugs und dieses steht erst zu einem späteren Zeitpunkt mit maximaler Reichweite zur Verfügung. Bei terminlichen Engpässen kann es dazu kommen, dass man auf eine schnellstmögliche Ladung für die zügige Zurverfügungstellung hoher Reichweite angewiesen ist. In diesen Fällen wäre es nicht zielführend die Ladeleistung zu reduzieren. Folglich hängt es von den persönlichen Lebensumständen ab, ob PV-Überschussladen, durchgeführt werden soll.

Spezielle Wallboxen können nicht nur dreiphasig laden, sondern besitzen eine Umschaltung von dreiphasigem zu einphasigem Laden. Somit beträgt der minimale Ladestrom nicht nur 18 Ampere (3 x 6 A 4,14 kWh), sondern 1 x 6 Ampere (1 x 6 A 1,38 kWh). Dadurch können auch geringe PV- Überschussmengen ausreichen, um das E-Auto langsam, aber kontinuierlich zu laden. Leider gibt es bei dieser plötzlichen Umschaltung Probleme mit der Ladeeinrichtung des Renault Zoe sowie des Hyundai Cona. Bei diesen Fahrzeugen ist die Umschaltung während des Ladens untersagt.²⁴ Im Rahmen der Eigenverbrauchs Optimierung sollte dieser Punkt nicht unerwähnt bleiben, da alle E-Auto Besitzer mit anderen Modellen von dieser Funktion profitieren können.

Im Folgenden werden 4 Lademodi vorgestellt, die in Open-WB implementiert sind.

1. Sofortladen

Hierbei wird das E-Fahrzeug mit der maximalen zur Verfügung stehenden Ladeleistung geladen. Diese beträgt bei einer förderfähigen Wallbox 11 KW²⁵. Diese kann bei Wechselstrom WB, die auch zuhause verbaut werden können, auf bis zu 22 KW erhöht werden. Der Zeitpunkt, wann die Ladung abgeschlossen ist, kann somit sehr klar definiert werden. Vorteil dieses Lademodus ist, dass die Ladung so schnell wie möglich durchgeführt wird. Nachteil hierbei ist das keine aktive Eigenverbrauchsoptimierung durchgeführt wird. Dieser Lademodus eignet sich, wenn das E- Auto schnellstmöglich wieder bewegt werden muss.

²⁴ https://openwb.de/main/?page_id=31 letzter Zugriff am 11.07.2022 siehe *6

²⁵ KfW, Merkblatt Ladestation für Elektroautos – Wohngebäude, S.3 Abs. 1

2. PV-Überschuss Laden

Hierbei wird das E-Auto nur geladen, wenn die dafür nötige Solarenergie zur Verfügung steht. Die minimale Stromstärke beträgt technisch bedingt 6 Ampere. Vorteil dieses Lademodus ist, dass aktiv die maximale Eigenverbrauchsoptimierung für die Ladung genutzt wird. Nachteil dieses Lademodus ist, dass das E-Auto entweder gar nicht oder nur teilweise bis zur nächsten Abfahrtszeit geladen sein kann. Somit ist es von Nöten genau zu überlegen, wann dieser Lademodus wirklich sinnvoll ist. Gleichzeitig kann es zu Taktungen des Ladevorgangs kommen. Die minimale Leistung bei der geladen werden kann ist 4,14 kWh. Sollte die zur Verfügung stehende überschüssige Solarenergie um diesen Wert schwanken, so würde das Laden immer wieder an- und abgeschaltet werden. Hierfür lassen sich gewisse Toleranzen einstellen. Jedoch sollten Informationen beim Hersteller bezogen werden, ob solch eine Ladung langfristig die Ladeelektronik oder die Batterie schädigt. Dieser Lademodus eignet sich, wenn eine Ladung des E-Autos zeitlich unbefristet ist oder die nächste Fahrt auch mit der vorhandenen Batteriekapazität durchgeführt werden kann.

3. Min + PV

Der Modus Min + PV ist eine Mischung aus den beiden vorangegangenen Lademodi. Dieser Modus lädt das Fahrzeug kontinuierlich mit einem voreinstellbaren Mindeststrom und erhöht diesen, sobald überschüssige Solarenergie zur Verfügung steht. Vorteil hierbei ist, dass überschlägig bestimmt werden kann, wann das E Auto spätestens fertig geladen ist. Gleichzeitig findet hier, sobald überschüssige Solarenergie zur Verfügung steht, eine Eigenverbrauchsoptimierung durch die Erhöhung des Ladestroms, statt. Ein weiterer Aspekt ist, dass die Batterie sehr schonend mit einem geringen Strom geladen wird und nur die Ladeleistung erhöht wird, wenn dies eine Eigenverbrauchsoptimierung zur Folge hätte. Als Nachteil ist zu nennen, dass, wenn keine Solarenergie zur Verfügung steht, netzbezogener Strom für die E-Auto Ladung verwendet wird. Dieser Modus eignet sich, wenn der nächste Startzeitpunkt des E-Autos feststeht aber noch so weit in der Zukunft liegt, dass man auch mit minimalem Ladestrom den Wagen ausreichend laden könnte.

4. Abschlusszeit

Bei diesem Lademodus handelt es sich um eine Mischform aus mehreren Lademodi unter Berücksichtigung einer Abschlusszeit. Hierbei müssen viele Faktoren als Voraussetzung gelten. Zum einen sollte der Ladezustand des E-Autos ausgelesen und verwertet werden können. zum anderen sollten der Wallbox fahrzeugspezifische Ladegeschwindigkeiten bei unterschiedlichem Fahrzeug SOC bekannt sein und zur Verfügung stehen. Die Wallbox kumuliert die benötigte Ladezeit bei maximal möglicher Leistung. Diesen Zeitfaktor zieht sie von der Abschlusszeit ab und ermittelt den Zeitpunkt, ab dem das E-Auto mit maximaler Leistung geladen werden müsste. Jede Ladung vor diesem Zeitpunkt wird mit dem Lademodus PV realisiert. Die Vorteile dieses Lademodus sind, die klare Kombination der Eigenverbrauchs Optimierung und der Notwendigkeit das E-Auto mit einem definierten SOC zum Abschlusszeitpunkt

führen zu müssen. Der Nachteil ist, dass das E-Auto kontinuierlich und zuverlässig ausgelesen werden muss. E-Autos, ohne EEBUS, übertragen Ihre Daten über das Mobilnetz oder WLAN an eine Cloud API. Die Internet-Kommunikation über Mobilfunk kann in Deutschland noch nicht flächendeckend realisiert werden. Somit kann ein alter Fahrzeugladezustand in der Datenbank hinterlegt sein, doch ein anderer, eventuell viel geringerer, liegt am E-Auto an. Zur Folge hätte das einen zu späten Start der Sofortladung, der zu einer unzureichenden Batterieladung des E- Autos führen kann.

Zusatz: SOC Vorgabe

Wenn die Kommunikation zwischen E-Auto und WB einwandfrei gewährleistet werden kann, dann lassen sich auch Ziel SOC's realisieren. Eine Ladung der Fahrzeugbatterie auf nur 90 oder 80% schont auf lange Sicht die Batterielebenszeit und verlängert somit maßgeblich die Lebenszeit eines E-Autos.²⁶

Als Ergebnis lässt sich zusammenfassen, dass es nicht den optimalen Lademodus gibt. Der Lademodus richtet sich vielmehr nach den persönlichen Bedürfnissen des Nutzers.

2.3 Sicherheitsuntersuchung der Kommunikation über Modbus TCP

Modbus TCP auf Basis von IP ist das hier in der Kommunikation gewählte Protokoll. Im Folgenden wird auf Grundlage des OSI-Modells jede Schicht auf seine sicherheitsspezifischen Ausprägungen zu analysieren.²⁷

Die Schicht 2 ist der sogenannte „Data Link Layer“ auch Sicherungsschicht genannt. Hier wird unter anderem das Mac-Protokoll verwendet. Grundsätzlich haben alle Mac-Adressen die Möglichkeit Teil des Netzwerkes zu werden. Ein MAC Adressenfilter kann im Router implementiert werden.

Die Schicht 3 ist der sogenannte „Network Layer“ oder auch Vermittlungsschicht. Hier wird mit dem IP-Protokoll gearbeitet. Dadurch können alle Geräte in einem Netzwerk miteinander kommunizieren, wenn die Router Richtlinien dies nicht ausschließen. Sungrow Wechselrichter mit Wii-Net Kommunikationsadapter besitzen die Möglichkeit eines IP-Adressen „Whitelisting“ und nur Geräte, die in dieser Liste aufgeführt werden, können Register des Wechselrichters auslesen. Zum Testzeitpunkt hat leider die Modbus Funktionalität über den Dongle nicht funktioniert und somit konnte diese Funktion nicht getestet werden.

²⁶ Quelle: <https://www.pv-magazine.de/2017/04/21/fragen-antworten-zum-webinar-batterielebensdauer-einschaetzen-und-verlaengern-teil-3/> letzter Zugriff am 20.08.2022

²⁷ Quelle: <https://www.feldbusse.de/Modbus/TCP/Modbus/TCP.shtml> letzter Zugriff am 20.08.2022

Alle Geräte innerhalb des Netzwerks dürfen und können miteinander kommunizieren. Schicht 4 ist der sogenannte „Transport Layer“ oder auch Transportschicht genannt. Hierbei sind Protokolle wie TCP zu finden. Aufgrund der Verwendung des TCP Protokolls werden Datenverluste erkannt und automatisch behoben. Zusätzlich sind Datenübertragungen in beide Richtungen möglich und Netzüberlastungen wird vorgebeugt.

Schicht 5 Darstellungsschicht (Presentation Layer) und Schicht 6 die Sitzungsschicht (Session Layer) werden nicht betrachtet.

Schicht 7 ist der sogenannte „Application Layer“, die Anwendungsschicht. Hier sind Protokolle wie das Modbus Protokoll untergebracht. Modbus wird unverschlüsselt und im Klartext übertragen. Es enthält keinerlei Authentifizierung eines Kommunikationspartners.

Zusammenfassend wird festgehalten, dass die IT-Schutzziele Verfügbarkeit und Integrität durch die Verwendung von TCP während der Übertragung gestärkt werden. Vertraulichkeit, Authentifizierung, Verbindlichkeit und Datenschutz als IT-Schutzziele werden durch diese technische Umsetzung nicht gewährleistet. Sobald sich ein Angreifer Zutritt zu dem Testnetzwerk verschafft, hat dieser die Möglichkeit der Kommunikation zu lauschen oder selbst Verbindung mit den Modbus Geräten aufzunehmen. 2018 wurde Modbus um den Verschlüsselungsalgorithmus TLS ergänzt, jedoch kommt diese Implementierung ins Stocken. Bei dem aktuell vorliegendem Testnetzwerk ist es nicht möglich Modbus TCP mit TLS zu nutzen.²⁸

Eine Suche mit der Suchmaschine Shodan²⁹ ergab mehr als 38000 Geräte auf Port 502. Diese sind aus dem Internet frei erreichbar, nur um was es sich für Geräte handelt ist unbekannt. Der Internetdienst Shodan bietet eine Suchmaschine für angreifbare Objekte im Internet.

Ein Schutz vor Fremdzugriff ist aktuell nur durch den Schutz des Netzwerkes nach außen zu gewährleisten. Die Umsetzung einer internen Sicherheit durch Verschlüsselung und Zertifikate ist im Rahmen von Modbus aktuell nicht flächendeckend eingeführt. Dieser Umstand muss bei der Verwendung von Modbus bedacht werden.

²⁸ <https://www.informatik-aktuell.de/betrieb/sicherheit/modbus-angriffe-im-lokalen-netzwerk.html> letzter Zugriff am 11.07.2022

²⁹ <https://www.shodan.io/> letzter Zugriff am 11.07.2022

Kapitel 3 Implementierung von „„legacy““ Geräten in das EEBUS Protokoll

In diesem Kapitel werden die grundlegenden Funktionsweisen von EEBUS erklärt und an einem aktuell funktionierenden Beispiel erläutert. Dabei wird im speziellen darauf eingegangen, was dieses Protokoll vom Modbus Protokoll anhand des OSI-Modells unterscheidet. Es werden die „Use Cases“ des EEBUS Standards genannt, die speziell durch den Testaufbau umsetzbar sind. Konkret werden die Datentypen, Einheiten sowie Größen genannt, die die Spezifikation vorsieht und untersucht ob hierfür Konvertierungen nötig sind. Es werden die zur Verfügung gestellten Tools des Stack Entwicklers KEO genannt und erläutert. Dabei wird der Datenfluss dargestellt und die Schnittstellen zueinander beschrieben. Es wird darauf eingegangen, wie die Implementierung der Schnittstelle für die Kommunikation mit „„legacy““-Geräten gelöst wurde. Es folgt eine umfangreiche Funktionsprüfung, die ebenfalls die Latenz des Systems untersucht.

3.1 Was ist EEBUS

EEBUS ist ein seit 2012 entwickeltes Protokoll der EEBUS Initiative. Es verfolgt das Ziel eine einheitliche globale Sprache für die Energiemanagement Kommunikation zu etablieren. Es soll als Plug and Play Lösung fungieren und einheitliche „Use Cases“ hersteller- und geräteübergreifend implementieren. Folgende Unternehmen und Verbände sind Teil der EEBUS Initiative oder unterstützen diese: (Stand August 2022)



Abbildung 8 EEBUS nutzende oder unterstützende Unternehmen.³⁰

³⁰ Quelle: Öffentliche Präsentation EEBUS_Short_Introduction-v1.25 von <https://www.eebus.org/media-downloads/> letzter Zugriff am 16.07.2022

Ein Kernziel der Initiative ist, äquivalent zu dem Ziel dieser Bachelorarbeit, die Eigenverbrauchsoptimierung von Haushalten und Gemeinschaften, die ihren Strom anteilig selbst erzeugen. Der essenzielle Unterschied zur Open WB Lösung ist, dass die einzubindenden Geräte einen einheitlichen Kommunikationsstandard haben und nach erfolgtem Pairing selbständig die nötigen Daten weiterreichen. Die Datenweitergabe ist bei EEBUS auf das Nötigste beschränkt. Es werden nicht mehr Daten zur Verfügung gestellt, als der dazugehörige „Use Case“ benötigt. Im Fokus dieser Standardisierung steht eine gemeinschaftliche Erarbeitung von sinnvollen „Use Cases“, welche durch Industrie und Endnutzer zielführend eingesetzt werden können. Dort enthalten sind ebenfalls verschiedenste Mechanismen der Verschlüsselung und Authentifizierung, welche ihn gegen äußere Angriffe absichern. Aktuell liegt noch keine Open Source Implementierung vor, so dass die Geräthewahl sehr eingeschränkt ist, wenn die Nutzung von EEBUS beabsichtigt wird. Die Tendenz der Gerätehersteller die EEBUS nutzen steigt, jedoch dominieren Geräte mit konventionellen Kommunikationsprotokollen den Markt. Zum jetzigen Zeitpunkt besteht für Bestandshardware ohne EEBUS Unterstützung, keine Möglichkeit, in Form eines Konverters, mit EEBUS Geräten zu interagieren.

3.2 Aufbau des EEBUS Protokolls anhand des OSI-Modells

Auf den ersten Blick ähneln sich der Aufbau des Modbus TCP Protokolls und der des EEBUS Protokolls. Das Hauptprotokoll teilt sich in zwei Unterprotokolle. SHIP ist für den Transport der Daten zuständig (Bis Schicht 7) und SPINE definiert Datenmodell und Protokoll sowie eine Methode zur Serialisierung (ausschließlich Schicht 7). Als Basis aller Protokolle wird Schicht 2 und 3 identisch verwendet. Auf ihnen baut nun Schicht 4 die Transportschicht auf. SHIP steht statt TCP Verfügung. UDP steht im Rahmen von mDNS dem „announce“ und „discover“ von SHIP Services zur Verfügung. Zur Sicherung der SHIP-Kommunikation wurde TLS verwendet. Wie vorher erwähnt gibt es Modbus auch mit der Verwendung von TLS. Diese Implementierung ist aber im Endanwenderbereich nicht weit verbreitet und wird deswegen im Vergleich nicht betrachtet.

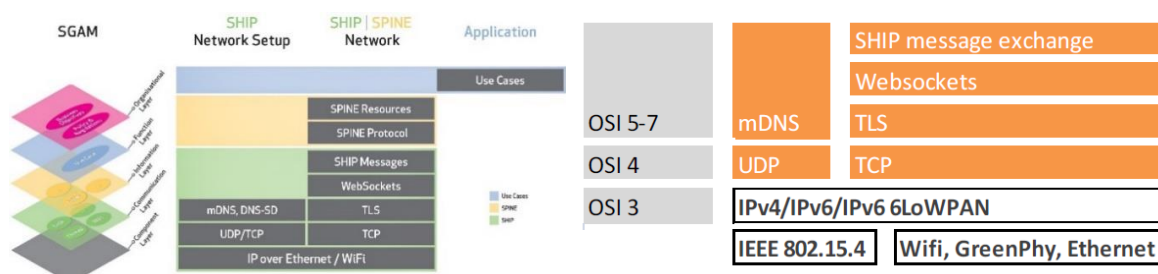


Abbildung 9 EEBUS SHIP/SPINE/APP Übersicht³¹

OSI-Modell Übersicht EEBUS³²

³¹ Quelle: [https://www.eebus.org/technology/#iLightbox\[101d9b6470babe38dff\]/0](https://www.eebus.org/technology/#iLightbox[101d9b6470babe38dff]/0) letzter Zugriff am 16.07.2022

³² Maren Fiege EEBus Initiative e.V., S. 13. Figure 2

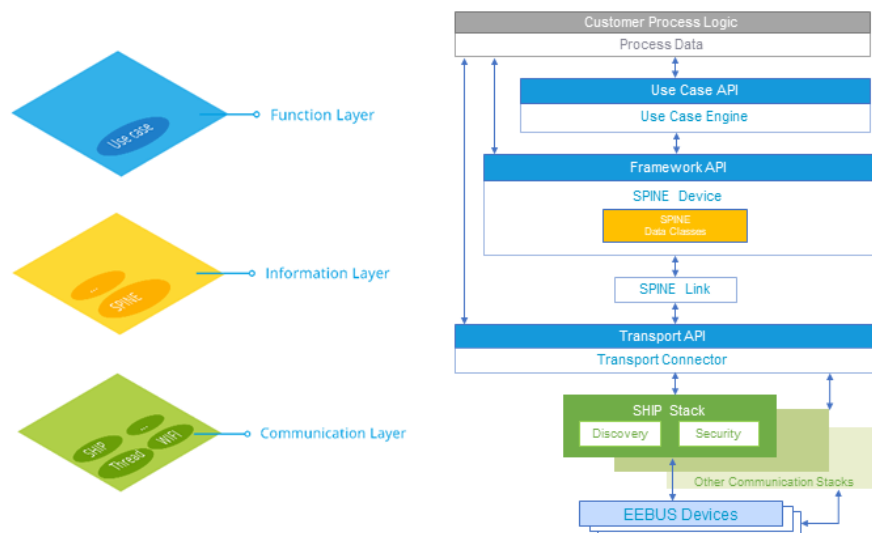


Abbildung 10 Konstruktionsaufbau EEBUS

3.2.1 SHIP das Transportprotokoll

SHIP beschreibt den standardisierten Transport der Daten über TCP/IP. Im Gegensatz zu herkömmlichen Transportprotokollen bieten die SHIP Mechanismen eine Grundlage zur sicheren Kommunikation und Authentifizierung. Alle SHIP Geräte können mit anderen SHIP Geräten im selben Netzwerk unkompliziert nach dem „Plug and Play“ Prinzip kommunizieren. Ein Sicherheitskernelement ist, dass jede neue Kommunikationsanfrage von einem Fremdgerät durch einen realen Nutzer initiiert und bestätigt werden muss: Diese Bestätigung muss nutzerseitig durch das fragende und antwortende Gerät gegeben werden. (Schicht 3)

Wie bei Modbus sorgt TCP in Schicht 4 dafür, dass eine vollständige Datenübertragung gewährleistet wird, Datenverluste hierbei erkannt und automatisch behoben werden. Gleichzeitig ist durch dieses Protokoll eine bidirektionale Verbindung möglich, dieses fördert die Netzwerkentlastung. (Schicht 4)

In Schicht 5 - 7 findet das genannte TLS Verschlüsselungsverfahren statt. Es werden ebenfalls Web-Sockets zur Verfügung gestellt und der SHIP Datenaustausch gewährleistet. Es findet in diesen Schichten auch der mDNS Dienst eine Verwendung. Somit können Netzwerke auch ohne aktiven DNS-Server über das SHIP-Protokoll eine Auflösung von Hostnamen zu IP -Adressen bekommen. Eine Authentifizierungsprüfung findet ebenfalls statt.

Der Vergleich zwischen Modbus und SHIP unter Transportaspekten fällt eindeutig aus. Alle Mechanismen und Protokolle aus Modbus sind in SHIP enthalten. Das Protokoll wird sinnvollerweise durch TLS, mDNS, der Nutzung von Web-Sockets und einer Authentifizierungsprüfung ergänzt. Dieses macht SHIP bedeutend umfangreicher und sicherer.³³

³³ SMA SMART HOME Energiemanagement mit elektrischen Verbrauchern über EEBUS, SMA Solar Technology AG, Kapitel 2.2-3

3.2.2 SPINE das Informationsprotokoll

Dieses Protokoll findet sich größtenteils nur auf der Anwendungsschicht also Schicht 7 wieder. Es ist komplett abgekoppelt von SHIP und jeglichen Transportmechanismen. Es befasst sich mit dem Protokoll zur Kommunikation zwischen zwei SPINE-Devices und der darauf aufbauenden strukturellen Implementierung der „Use Cases“.

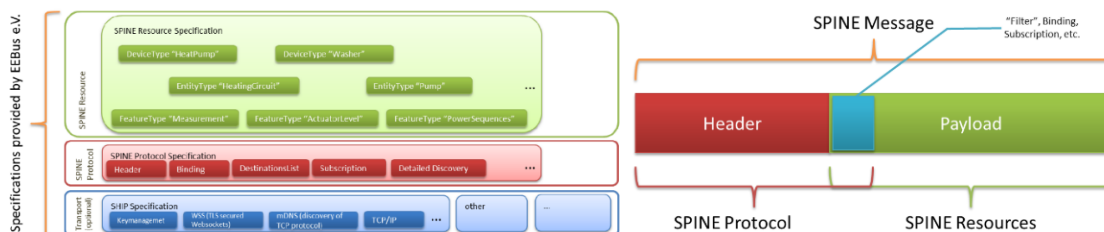


Abbildung 11 EEBUS Spezifikation im Detail³⁴

SPINE Nachricht Überblick

Das Protokoll befasst sich mit der Umsetzung vorher definierter „Use Cases“. Die SPINE Nachricht unterteilt sich in zwei Adressbereiche, das SPINE Protokoll und die SPINE Ressourcen. Die Protokollspezifikation ist ähnlich aufgebaut wie in anderen Smart Home Modellen. Hierbei werden unter anderem der „Header“ und „Subscriptions“ übertragen.

Die Ressourcen wiederum fungieren als umfangreiches Toolkit. Je nach verwendetem „Use Case“ werden einzelne Ressourcen daraus benötigt und verwendet. Die in den „Use Cases“ festgelegten Datenmodelle finden sich in den einzelnen Ressourcen wieder.

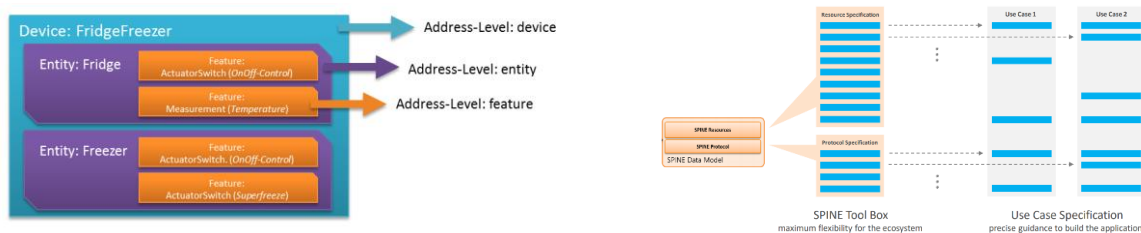


Abbildung 12 Exemplarischer Aufbau SPINE Devices³⁵

SPINE als Toolbox³⁶

SPINE erlaubt die Beschreibung einer Gerätestruktur. Das Gerät steht an oberster Stelle und bildet in sich eine geschlossene Einheit. Es hält in sich weitere Entitäten oder auch Geräte-Untergruppierungen. Diese Entitäten halten verschiedene Funktionen und Features parat, die einzeln adressierbar sind. Dabei kann es sich sowohl um Funktionen zum Ausführen von Befehlen (Waschmaschine anschalten)

³⁴ EEBus SPINE Technical Report Introduction, EEBus Initiative e.V., S. 5 Figure 3

³⁵ EEBus SPINE Technical Report Introduction, EEBus Initiative e.V., S. 9 Figure 5

³⁶ Quelle: [https://www.eebus.org/technology/#iLightbox\[101d9b6470babe38dff\]/0](https://www.eebus.org/technology/#iLightbox[101d9b6470babe38dff]/0) letzter Zugriff am 16.07.2022

oder auch zum Lesen aktueller Werte (PV-Leistung) handeln. SPINE ist hier an der Stelle nur eine sogenannte „Tool Box“. Welche Funktionen und Tools verwendet werden gibt der umzusetzende „Use Case“ vor.

Der „Use Case“ steht an oberster Stelle. Er gibt die funktionalen Anforderungen vor. Daraus ergibt sich wie die erforderlichen Daten, basierend auf dem SPINE Modell, abzubilden sind und welche Komponenten aus SPINE zur Anwendung kommen.

EEBUS liefert mit SHIP und SPINE eine voneinander getrennte Grundlage für das Energiemanagement der Zukunft. Es ersetzt den Modbus Ansatz grundlegend, mit einer klaren Zielsetzung und Struktur.³⁷

3.2.3 Umsetzung der Eigenverbrauchsoptimierung

Anhand eines Netzwerks mit einem EEBUS CEM und mehreren EEBUS Endgeräten wird der grundlegende Vorgang erläutert, wie die EEBUS Geräte miteinander auf SPINE Ebene kommunizieren. Als Grundlage gilt, dass der Kommunikationsaufbau mit SHIP erfolgreich durchgeführt worden ist und weiterhin besteht. Nach dem sich die Geräte einander authentifiziert haben, tauschen CEM und Endgerät miteinander Listen aus, die hergeben welche EEBUS „Use Cases“ vom Gerät unterstützt werden. Überschneiden sich nun welche davon, dann wird selbständig eine „Subscription“ für die Datenpunkte dieses „Use Case“ zum Datenaustausch etabliert. Somit stehen dem CEM immer alle möglichen Daten zur Verfügung, um seine Aufgabe der Steuerung der Energieverteilung, bestmöglich durchführen zu können. Hersteller von Energie Management Software können dem CEM in Ihre Architektur einbetten. Auf dieser Grundlage wird die Businesslogik geschrieben. In dieser können dem Nutzer Einstellmöglichkeiten zur Feinjustierungen der Eigenverbrauchsoptimierung gegeben. Dabei kann der Nutzer Priorisierungen der Geräte festlegen und bei E-Autos Lademodi bestimmen. Bei EEBUS Wärmepumpen, die über den Sunny Home Manager gesteuert werden, muss der Autarkiegrad³⁸ für die Nutzung dieses Gerätes festgelegt werden. Diese Festlegung hat jedoch Grenzen, so muss der Legionellen Schutz im Brauchwasserspeicher³⁹ kontinuierlich gegeben sein. Fällt nun die Temperatur im Brauchwasserspeicher unter einen definierten Wert so startet die Wärmepumpe unabhängig von getroffenen EEBUS Einstellungen. Die getroffenen Einstellungen dürfen keine geltenden Normen und technischen Betriebsvorschriften verletzen. Somit erfolgt die Verbrauchsoptimierung im Rahmen der zulässigen Einstellparameter. Welche „Use Cases“ umgesetzt werden und hierfür zur Verfügung stehen entscheiden einzig die Hersteller der Endgeräte.

³⁷ Torben Diekmann; Andreas Schwackenberg: EEBUS SPINE Protocol Specification

Torben Diekmann; Andreas Schwackenberg: EEBUS SHIP Protocol Specification

³⁸ der Autarkiegrad gibt das Verhältnis aus Nutzung von selbst erzeugtem Strom zu eingekauftem Strom an

³⁹ DIN EN 806-5 | 2012-04

Technische Regeln für Trinkwasser-Installationen - Teil 5: Betrieb und Wartung Quelle:

<https://www.baunormenlexikon.de/norm/din-en-806-5/a893cc21-7436-4e14-9158-8edb77ff24d9> letzter Zugriff am 10.08.2022

3.3 Native EEBUS Nutzung zur Eigenverbrauchsoptimierung

Viele namhafte Hersteller unterstützen die EEBUS Initiative. Hierbei sind auch schon viele Geräte auf dem Markt die EEBUS nutzen wie zum Beispiel Wärmepumpen von Wolf, Nibe und Vaillant oder Wallboxen von Mennekes. Viele andere Hersteller stehen mit ihren Elektrofahrzeugen oder anderen Geräten in den Startlöchern EEBUS zu implementieren. Jedoch unterstützen zum jetzigen Zeitpunkt nur wenige Wechselrichterhersteller den EEBUS Standard. Viele Hersteller setzen weiterhin auf eine offene Schnittstelle wie die Modbus Auslesefunktionalität oder Cloud basierte Visualisierung der Wechselrichterdaten. Noch viel gravierender ist es auf dem Markt der verfügbaren Energiemanager, die einen vollwertigen CEM zur Verfügung stellen. Hier hat es zum jetzigen Zeitpunkt nur SMA geschafft im Sunny Home Manager dies zu inkludieren.

Im Folgenden wird am Beispiel einer Vaillant Wärmepumpe die versprochene „Plug and Play“ Inkludierung des EEBUS Protokolls dargestellt und erläutert.

Zunächst muss die EEBUS Funktionalität sowohl beim Vaillant Gerät als auch im Sunny Home Manager vom Nutzer aktiviert werden. Nun ist es den Geräten möglich sich gegenseitig im Netzwerk zu identifizieren. In der App des Geräteherstellers Vaillant taucht nun der Sunny Home Manager mit seiner ID als verfügbares EEBUS EMS auf. Die IP-Adresseneingabe ist hier nicht erforderlich, da die Detektion automatisch über das SHIP Protokoll im Netzwerk abläuft. Wie in den Sicherheitsmaßnahmen erwähnt muss der Sunny Home Manager nun als vertrauenswürdiges Gerät hinzugefügt werden. Nach dem Abschluss stehen der Wärmepumpe einige Informationen des CEM zur Verfügung. In diesem Beispiel die aktuell gegebene Leistung der Solaranlage und der Ladezustand der Batterie.

Danach wird im Sunny Homemanager ein neues Gerät gesucht. Die WP wird ohne Umwege erkannt und richtig identifiziert. Die Aufnahme des Gerätes in das EMS erfolgt ohne Komplikationen. Während des kurzen dreischrittigen Konfigurationsprozesses lassen sich hierbei Feineinstellungen der Eigenverbrauchsoptimierung durchführen. Nach einem Neustart des Sunny Home Managers ist die Verbindung der Geräte gewährleistet und die Wärmepumpe kann vom Sunny Home Manager unter Berücksichtigung der Eigenverbrauchsoptimierung gesteuert werden.

Das „Plug and Play“-Versprechen wird trotz der gegebenen Sicherheitsstandards weitestgehend eingehalten. Es erschließt sich leider nicht, aus dem technischen Kontext, warum der Sunny Home Manager zum Abschließen der Konfiguration neu gestartet werden muss. Diese Implementierung ist einfach und für jeden Endkunden per Anleitung spielend durchzuführen.

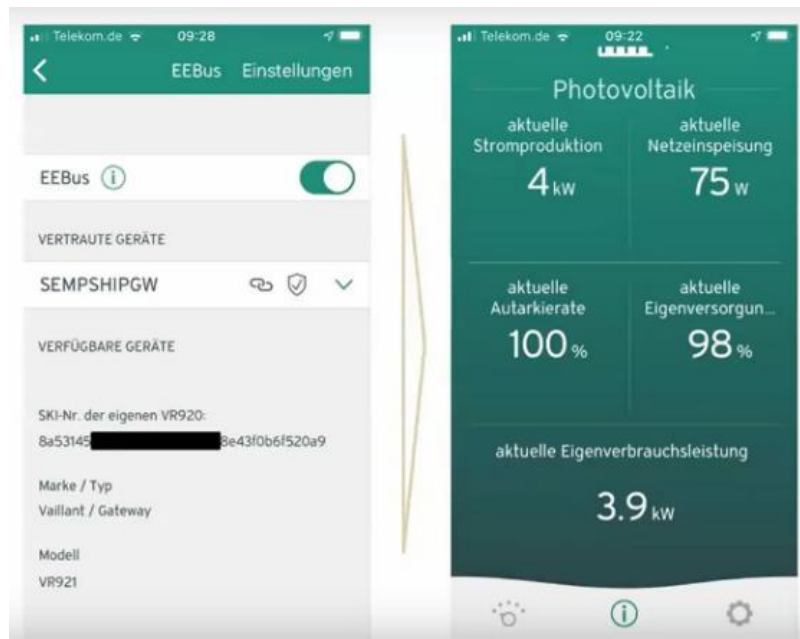


Abbildung 13 Exemplarischer APP Auszug Vaillant⁴⁰

3.4 EEBUS „Use Cases“ die mit dem Netzwerk umgesetzt werden sollen

Geräte wie zum Beispiel Wallboxen, die EEBUS verwenden, werden nicht dazu gezwungen jeden „Use Case“ der jeweiligen Spezifikation zu implementieren. Es gibt hierbei Pflicht „Use Cases“, die für den essenziellen Betrieb nötig sind und optionale „Use Cases“, die je nach Anwendungsfall implementiert werden. Bei einigen Geräten stehen spezielle Sensordaten nicht zur Verfügung, weil der dafür notwendige Sensor nicht verbaut wurde.

Ähnlich sieht es im Bereich der Lesegeräte aus, hierbei ist es komplett herstellerabhängig welche Daten über Modbus übertragen werden und welche Sensoren in welcher Auswertequalität zur Verfügung stehen. Zusätzlich können die ausgelesenen Daten in unterschiedlichen Einheiten, Größen oder mit Vorzeichenumkehr angezeigt werden. EEBUS hat diesen Prozess standardisiert. Jeder Sensorwert wird in seiner Grundgröße angezeigt. Somit ist es von Nöten bei der folgenden Betrachtung der umsetzbaren „Use Cases“ diese Aspekte nicht zu vernachlässigen.

⁴⁰ Quelle:

https://www.youtube.com/watch?v=F3GIvWJd7fE&t=122s&ab_channel=ZentralverbandSanit%C3%A4rHeizungKlima-ZVSHK Minute 5, letzter Zugriff am 19.07.2022

3.4.1 Überwachung der Netzeinspeisung

Der zur Überwachung der Netzeinspeisung verwendete „Use Case“ heißt: „MGCP (Monitoring of Grid Connection Point)“.

Dieser beinhaltet folgende Szenarien:

Scenario 1 - Monitor PV feed-in power limitation factor

Scenario 2 - Monitor momentary power consumption/production

Scenario 3 - Monitor total feed-in energy

Scenario 4 - Monitor total consumed energy

Scenario 5 - Monitor momentary current consumption/production

Scenario 6 - Monitor voltage

Scenario 7 - Monitor frequency

In der Forschung wird Szenario 5 verwendet. Bei diesem Szenario handelt es sich um einen der essenziell wichtigsten „Use Cases“ im Wechselrichter „Power Meter“ Verbund. Szenario 5 von MGCP gibt an, ob Einspeisung oder Netzbezug vorliegt. Dies wird durch einen einfachen Vorzeichenwechsel bestimmt. Es werden drei Werte übergeben, und zwar der Stromfluss für alle 3 Phasen L1, L2 und L3 in Ampere. Dieser Wert ist wohl der Wichtigste, wenn es um die Nutzung von erneuerbaren Energien geht. Über diesen Wert lässt sich detektieren ob überschüssige Solar oder Windenergie vorliegt. Jedes EMS benötigt zur Steuerung der Aktoren, die Energieüberschuss geführt arbeiten, Informationen über diese spezifische Werte. Im weiteren Verlauf der Untersuchung werden nur Ampere Werte benötigt. Stehen jedoch nur Leistungswerte in Watt zur Verfügung stehen, dann ist eine Umrechnung in Ampere als Näherungswert zulässig. Da die Spannungsqualität in Deutschland eine sehr hohe ist und ein reibungsfreier Betrieb ohne eine 230 Volt Versorgung nicht möglich wäre, lässt sich dieser Wert als Konstante annehmen.⁴¹ In allen anderen Fällen riegelt der Wechselrichter ab oder der Energieversorger stellt die Versorgung zum Schutz der Endgeräte ein. Hierbei gilt bei der Spannungsqualität eine Toleranz von +-10%. Diese Toleranz ist aber bei der grundlegenden Berechnung unerheblich da die Spannung am Powermeter und auch am Endgerät gleichzeitig mit fast demselben Wert anliegt. Die Spannung trägt nicht zum Szenario 5 des „Use Cases“ bei, kann jedoch über MGCP mit übertragen werden.

Die verwendete Wallbox benötigt jedoch zur Anpassung des Ladestroms den Stromfluss in Ampere und nicht die Leistung in Watt. Dies kann bei anderen Wallboxen variieren, somit könnten andere Szenarios zielführender sein. Das zweite Szenario wäre hier ein Beispiel, dieser gibt die kumulierte Einspeiseleistung aller drei Phasen als Summe wieder. Um mit der Test Wallbox zu interagieren, muss jedoch der Stromfluss angegeben sein, somit ist eine Umrechnung von Spannung (V) und

⁴¹ Bundesnetzagentur, Bericht zur Spannungsqualität 2020, S.39

Leistung (W) in Strom (A) erforderlich. Folglich wird der Logik die Hin- und Rückrechnung der Leistung zum Stromfluss erspart, dies sollte die Effizienz steigern.

Die Modbus Spezifikation des Wechselrichters belegt, dass in den Registern 13031-13033 die aktuellen Stromflüsse der 3 Phasen hinterlegt sind.⁴² Das Vorzeichen soll hier angeben, ob Einspeisung oder Netzbezug vorliegt. Aus den jeweiligen Dokumentationen geht leider nicht eindeutig hervor welches Vorzeichen für welchen Fall zu wählen ist, somit muss eine Praxisauswertung zeigen, ob das Vorzeichen an den EEBUS „Use Case“ angeglichen werden muss.

3.4.2 Überwachung und Steuerung der Wallbox

Der zur Überwachung der Wallbox verwendete „Use Case“ heißt: „OSCEV (Optimization of Self-Consumption During EV Charging)“.

Dieser beinhaltet folgende Szenarien:

Scenario 1 - CEM informs EV about self-produced current

Scenario 2 - EV checks CEM availability

Scenario 3 - CEM sends error state

Dieser E-Mobility „Use Case“ ist das Kernelement der Nutzung überschüssiger selbsterzeugter Energie mit E-Fahrzeugen. Als Vorbedingung muss das EMS in Verbindung mit dem CEM Kenntnis über den minimalen und maximalen Ladestrom der Wallbox haben. Diese werden als die Grenzwerte betrachtet. Bei dem Lademodus PV Laden, würde das EMS zur Entlastung des Netzwerks keine Werte unter dem Minimum an die Wallbox senden. Erst wenn der kleinste mögliche Ladestrom umgesetzt werden kann, würde dies der WB mitgeteilt werden. Anders verhält es sich mit dem maximal möglichen Ladestrom, dieser kann aus verschiedensten Gründen limitiert werden. Folglich würde überschüssige Energie, die diesen Grenzwert überschreitet, nicht weitergegeben werden. Es würde nur der im EMS eingestellte höchstmögliche Wert gesendet werden. Gründe hierfür können die schonende Ladung der Autobatterie oder technische Limitierungen der Wallbox sein.

Spezifikation benötigt von allen 3 Phasen die aktuelle Stromstärke des Überschusses. Der Go-eCharger des Testaufbaus wird über eine Web API angesteuert. Diese benötigt lediglich einen ganzzahligen Amperewert zwischen 6 und 16 Ampere zur Regelung des Ladestroms. Da die Renault Zoe Ladeströme unter 12 Ampere nicht verarbeiten kann, stellt sich somit der einstellbare Bereich auf ganzzahlige Werte zwischen 12 und 16 Ampere ein. Die maximale Leistung der PV Anlage beträgt 10,5 kWp. Auf die Stromstärke bei 230 Volt kumuliert ergibt sich ein maximaler Ladestrom

⁴² Sungrow, Modbus Hybrid Spezifikation, Walter E. S.11

von 15,21 Ampere. Durch die in der Einleitung genannten Umwandlungsverluste fällt dieser Wert noch geringer aus. Somit gibt es drei bis vier Einstellwerte, die das EMS sinnvollerweise an die Wallbox senden kann.

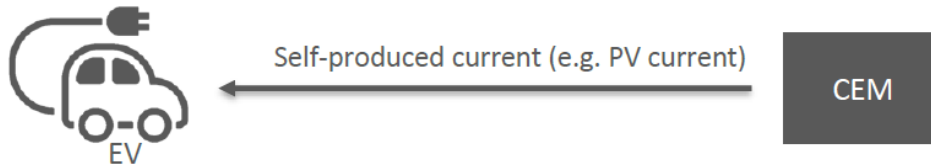


Abbildung 14 Schematische Darstellung Kommunikation EMS -> WB⁴³

Bei der Implementierung einer Web API kann es zu Zeitverzögerungen durch Latenzen kommen. Diese können verschiedenste Ursachen haben. Ein großes Problem hierbei ist, dass die Latenz stark variieren kann, da man auf die Server seitige Ausführungszeit oder Probleme in der Leitungsdimensionierung des Netzbetreibers keinen Einfluss hat. Somit kann eine zügige Umstellung der Ladeleistung erfolgen, allerdings durch diese Einflussfaktoren nicht garantiert werden.

Erst kurz vor Fertigstellung der Planung ist eine Modbus TCP Dokumentation auf GitHub zum Vorschein gekommen. Aus der Registerdokumentation war ersichtlich, dass sich für die PWM Signalisierung ganzzahlige Ampere Werte zwischen 6 und 32 Ampere in das beschreibbare Register 299 eintragen ließen. Dies begrenzt den Ladestrom auf einen dreiphasigen Maximalwert.

Aufgrund der Latenzen und zur Verfügbarkeitssteigerung wurde im weiteren Forschungsverlauf auf Modbus TCP gesetzt.

3.5 Kommunikationsschema

Die Firma KEO stellte verschiedene Tools und Dokumentationen zur Verfügung, um mein Vorhaben umzusetzen. Die gelieferten Applikationen implementieren jeweils die gewählten „Use Cases“. Angesteuert werden die Applikationen mittels JSON Nachrichten. Damit ein Implementierer schnell mit der Arbeit beginnen und Versuche durchführen kann, existiert das Mockup. Dieses ist eine Behelfs-/Improvisations-Implementierung ohne Business-Logik. Hiermit können unkompliziert JSON-Nachrichten zwischen den EEBus Applikationen versendet und empfangen werden. Der Austausch der Daten erfolgt hierbei im JSON Format. Die für den Integrator sichtbare öffentliche Schnittstelle ist auf die wesentliche Funktionalität zur Durchführung der „Use Cases“ beschränkt und befreit den Integrator somit von dem Detailwissen über das Datenmodell.

⁴³ Torben Diekmann EEBus Initiative e.V., S.10

In diesem ersten Versuch geht es darum die oben genannten und detailliert beschriebenen „Use Cases“ umzusetzen. Der Wechselrichter und die Wallbox werden per Modbus TCP angebunden. Das Vorhaben ist die Registereinträge des Wechselrichters auszulesen und die per JSON-Nachricht an die EEBUS Applikation zu senden. Für die WB gilt genau die umgekehrte Reihenfolge, hier wird eine JSON Nachricht empfangen, ausgelesen und der Wert in ein Register geschrieben.

3.5.1 Der Energiemanager und die EEBUS Adapter

Gemäß des Testaufbaus (siehe Abb. 14) hat der zuständige Betreuer bei KEO ein selbständig lauffähiges EMS vorkonfiguriert. Dessen Aufgabe ist es die Standard-Verbindungsmodalitäten eines EEBUS EMS für maximal 2 Geräte zur Verfügung zu stellen. diese Geräte sind bereits in dem „CEM“ vorkonfiguriert. Normalerweise müsste der Sicherheits- „Handshake“ durch den Nutzer des Endgerätes und des Energiemanagers händisch in der Software bestätigt werden. Dieses Sicherheitsfeature wurde zur Vereinfachung des Vorgangs vorkonfiguriert, sodass das Vertrauensverhältnis zu den beiden Geräteadaptern hinterlegt worden ist.

Der Energiemanager ist so konfiguriert, dass kein Einfluss auf die übergebenen Werte genommen wird. Die Applikation erhält Daten vom WR Adapter und reicht diese ungefiltert zum WB Adapter weiter. Die Kommunikation zwischen dem EMS und anderen EEBUS Geräten erfolgt über das SHIP Protokoll.

Die Verbindung zwischen den Adaptern und der nativen Herstellerimplementierung erfolgt entweder über eine C++ API oder erneut über JSON-basierte Schnittstelle. Im Rahmen der Forschung wurde auf sich die JSON API mit Web-Socket als Transport-Technologie gestützt.

3.5.2 Kommunikationskonzept des EEBUS Client Converters

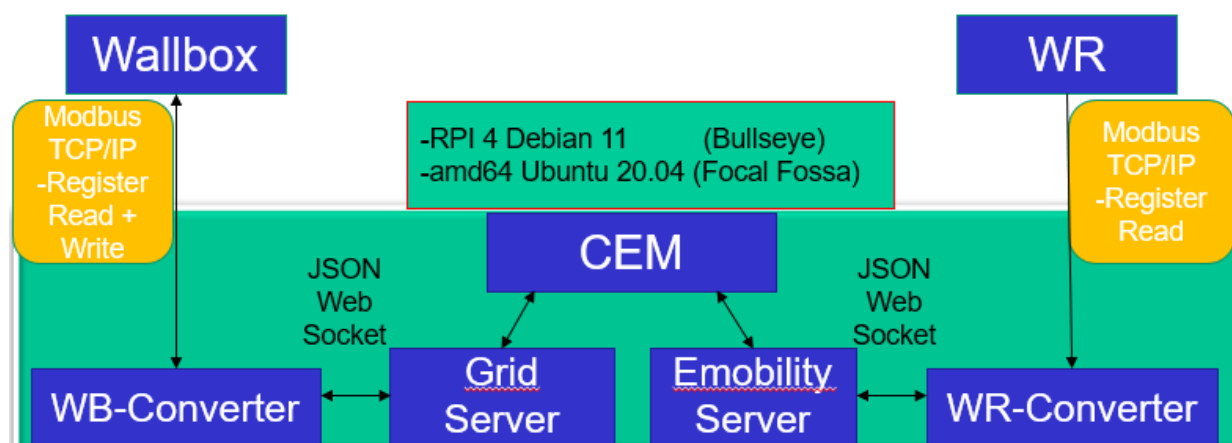


Abbildung 15 Kommunikationsentwurf des Testnetzwerks

KEO lieferte drei kommandozeilenlauffähige Applikationen. Hierbei handelt es sich um das oben genannte CEM und die jeweiligen WB und WR Applikationen. Werden die Applikationen gestartet und eine vorprogrammierte Verbindung zum CEM hergestellt. Die „Use Cases“, welche in den Applikationen zur Verfügung stehen sind inaktiv. Diese werden erst aktiviert, sobald eintreffende JSON Nachrichten einen spezifischen „Use Case“ aktivieren. Diese JSON Nachrichten sind nach strengen Vorgaben und für jedes „Use Case“ Szenario unterschiedlich aufgebaut. Grundsätzlich steht über allem die Spezifikation des „Use Cases“ und im Inhalt werden die benötigten Daten übertragen, um die Funktionalität des „Use Cases“ abzubilden.

Das Senden der JSON Nachricht zum WB-Adapter mit dem PV-Überschusswert in Ampere sollte eine Weiterleitungskette in Bewegung setzt. Der CEM empfängt über SHIP ein SPINE Datagramm als JSON-String und interpretiert die darin enthaltenen Daten und extrahiert den gegebenen Payload. Diese werden dann in ein neues Datagramm im Kontext des „Use Case“ OSCEV überführt und über SHIP an die Wallbox Applikation versendet. Nach dem Eintreffen der Information wird das SPINE Datagramm in eine KEO JSON Nachricht gewandelt und über einen Web-Socket an die Adapterlogik übermittelt.

Die Eigenleistung besteht nun darin

Beim WR:

- die Modbus TCP IP-Schnittstelle des Wechselrichters mit einer eigenen Funktion auszulesen.
- Die ausgelesenen Werte in ganzzahlige Ampere Werte abzurunden und ungültige Werte zu filtern. Gleichzeitig das wiederkehrende Senden eines unveränderten Wertes zu verhindern
- Eine JSON Build Funktion schreiben, die eine JSON Nachricht erzeugt, welche den Anforderungen des „Use Case“ MGCP Szenario 5 entspricht und die passenden Daten enthält
- Die JSON Nachrichten als String bei Wertänderungen an den Web-Socket des WR-Adapters senden

Bei der WB

- Auf einem definierten Web-Socket „Lauschen“ um JSON Nachrichten als String empfangen zu können
- Diesen String nach den PV Überschusswerten filtern
- Die Modbus TCP IP-Schnittstelle der Wallbox mit einer eigenen Funktion nutzen, um die Ladeleistung anpassen zu können und sie mit den filterten Werten aufrufen
- Eine Kontrollfunktion schreiben, um zu prüfen, ob der Wert erfolgreich geändert wurde

Überprüfung und Tests:

- Die an die Wallbox übergebenen Werte überprüfen, indem diese mit den Ursprungswerten verglichen, werden
- Verhalten bei der Eingabe untypischer Zeichen und Buchstaben überprüfen
- Bei Bedarf auftretende Latenzen untersuchen und möglichst minimieren

3.6 Implementierung

Im Folgenden werden die Implementierungen dargestellt und erläutert. Es werden praxisnahe Testverfahren verwendet, um die Plausibilität zu prüfen. Es wird der Lademodus Min+PV implementiert. D.h. die Wallbox versorgt kontinuierlich das E-Auto mit Ladestrom, aber der Ladestrom variiert synchron dreiphasig zwischen den möglichen Maximalwerten 6-16 Ampere.

3.6.1 Grundlage

Als Betriebssystem für die Entwicklung wurde Ubuntu 20.04 Focal Fossa verwendet. Als Programmiersprache kam Python 3.8 zum Einsatz. Als Entwicklungsumgebung wurde PyCham verwendet. Python wurde deswegen als Programmiersprache ausgewählt, weil es im privaten Smart Home Bereich sehr häufig zur Anwendung kommt. OpenWB verwendet im Großteil Python für die Backend Implementierung. Bei der Einrichtung der Entwicklungsumgebung kam es zu einigen Stolpersteinen. Zunächst funktionierte der Paketmanager nicht auf Anhieb. Nach einiger Recherche wurde festgestellt, dass ein zusätzliches Paket in der Kommandozeile des Betriebssystems installiert werden musste, um den Package Manager nutzen zu können.

3.6.2 Modbus Register mit Python nutzen

Da sowohl Wallbox als auch Wechselrichter über Modbus TCP IP eine Schnittstelle bereitstellten, war es essenziell von Nöten diese mit Python nutzen zu können. Alle Implementierungen, die in GitHub mit Modbus in Python angezeigt wurden, verwendeten die Bibliothek „pymodbus“. Zu dieser Bibliothek gab es auch eine umfangreiche Website⁴⁴, die die Funktionalität erläuterte und viele Beispiele bietet. Jedoch fängt hier die grundlegende Problematik im Zusammenspiel von Back End Funktionalität und Dokumentation mit Beispielen an. Zunächst wurde der Versuch unternommen einen Nachbau der OpenWB Realisierung⁴⁵ zu etablieren.

⁴⁴ <https://pymodbus.readthedocs.io/en/latest/> letzter Zugriff am 10.08.2022

⁴⁵ <https://github.com/snaptec/openWB/blob/master/packages/modules/common/modbus.py> letzter Zugriff am 10.08.2022

```

66     def __read_registers(self, read_register_method: Callable,
67                           address: int,
68                           types: Union[Iterable[ModbusDataType], ModbusDataType],
69                           byteorder: Endian = Endian.Big,
70                           wordorder: Endian = Endian.Big,
71                           **kwargs):
72         try:
73             multi_request = isinstance(types, Iterable)
74             if not multi_request:
75                 types = [types]
76
77             def divide_rounding_up(numerator: int, denominator: int):
78                 return -(numerator // denominator)
79
80             number_of_addresses = sum(divide_rounding_up(
81                 t.bits, _MODBUS_HOLDING_REGISTER_SIZE) for t in types)
82             response = read_register_method(
83                 address, number_of_addresses, **kwargs)
84             if response.isError():
85                 raise FaultState.error(__name__+" "+str(response))
86             decoder = BinaryPayloadDecoder.fromRegisters(response.registers, byteorder, wordorder)
87             result = [struct.unpack(">e", struct.pack(">H", decoder.decode_16bit_uint())) if t ==
88                       ModbusDataType.FLOAT_16 else getattr(decoder, t.decoding_method)() for t in types]

```

Abbildung 16 OpenWB Modbus.py Implementierungsausschnitt

An diesem Beispiel wird deutlich, wie verschachtelt die Funktionalität vom Entwicklerteam umgesetzt wurde. Daran ließen sich nur schwer die Kernelemente der Modbus Auslesung extrahieren. Eine Mangelnde strukturelle Übersichtlichkeit und fehlende Kommentare zu den einzelnen Code-Bereichen waren während des Einstieges in die Thematik sehr hinderlich. Eine Dokumentation zur Implementierung lag leider nicht vor, sodass sich von einer Aneignung der pymodbus Bibliotheksnutzung anhand dieses Beispiels distanziert wurde. Es blieb nur die Dokumentation der Bibliothek zu analysieren.

Zunächst wurde die dafür notwendige Bibliothek pymodbus in der Version 2.5.3 installiert. Für die Python Version 3.8 befand sich die Version 3.0.0 mit dem Stand dev 4 in der Entwicklung. Danach wurden die TCP spezifischen Passagen aus dem Beispielcode⁴⁶ kopiert und versucht eine grundlegende Funktionalität des Modbusverbindungsbaus zu erzeugen. Jedoch funktionierte der Code nicht auf Anhieb.

Die Kommandozeile konnte die im Beispiel verwendeten Befehle, trotz korrekter Importoperationen nicht finden. Auch eine händische Suche in der „Library“ blieb völlig erfolglos. Der Fehler war, dass wohl versehentlich Codebeispiele aus der Version 3 in die aktuelle „latest“ Versionsdokumentation gerutscht sind. Folglich wurden Befehle verwendet, die es in der Bibliothek nicht gab. Nach einer Suche in anderen Beispielen wurde der Fehler erkannt und behoben. Alle weiteren Inhalte konnten durch die Dokumentation sowie den einzelnen Modbus Spezifikationen der Hersteller hergeleitet werden.

⁴⁶ https://pymodbus.readthedocs.io/en/latest/source/example/client_sync.html letzter Zugriff am 10.08.2022

3.6.3 Modbus Register des WR auslesen

```
client = ModbusTcpClient('192.168.188.61', port=502) # WiiNet Wifi

result = client.connect()
#print(result)
result2 = client.is_socket_open()
#print(result2)

def CurrentL1():
    value1 = client.read_input_registers(13030, 1, unit=1)
    decoder = BinaryPayloadDecoder.fromRegisters(value1.registers, Endian.Big)
    value2 = decoder.decode_16bit_int()
    value3 = value2 / 10
    return (value3)

def CurrentL2():
    value1 = client.read_input_registers(13031, 1, unit=1)
    decoder = BinaryPayloadDecoder.fromRegisters(value1.registers, Endian.Big)
    value2 = decoder.decode_16bit_int()
    value3 = value2 / 10
    return (value3)

def CurrentL3():
    value1 = client.read_input_registers(13032, 1, unit=1)
    decoder = BinaryPayloadDecoder.fromRegisters(value1.registers, Endian.Big)
    value2 = decoder.decode_16bit_int()
    value3 = value2 / 10
    return (value3)

client.close()
```

Programcode 1 Grundfunktion Auslesen des Modus Registers für den PV-Überschussstrom L1

Aus der Dokumentation für die Modbus Schnittstelle des Sungrow Hybrid Wechselrichters wurden folgende umzusetzende Eigenschaften übernommen. Die Werte des eingespeisten PV-Überschuss-Stroms ließen sich für L1-L3 in den Registern 13031, 13032 und 13033 auslesen. Bei der OpenWB Umsetzung ist aufgefallen, dass alle Register um -1 verschoben waren. D.h. es wurde z.B., um das Register 100 auszulesen der Wert 99 in die Auslesefunktion eingetragen. Diese Erkenntnis bestätigte sich mit dem Lesen der Dokumentationshinweise. „Arrays“ starten bei der Adresszuweisung bei dem Adresswert „0“. Darin begründet sich dann, dass die Adressen um eins verschoben sind.

Die Reihenfolge der Bitanordnung war „big Endian“ und als Wortlänge wurden 16 Bit „signed Integer“ für diese Register verwendet. Die Einheit ist in Ampere * 10^{-1} (0,1A) in dem Register hinterlegt.

Durch den Aufruf „Current L1“ wird, bei erfolgreicher Ausführung der Funktion, der PV-Überschuss an L1 als Stromstärke in Ampere ausgegeben. Zunächst wurden die Grundparameter IP-Adresse und Port mit der Funktion „Modbus/TCP-Client“ in dem Objekt Client instanziiert. Danach wird die Verbindung mit dem „connect()“ Aufruf hergestellt. Dabei wird ein „boolscher“-Wert als Rückgabewert ausgegeben. Dieser gibt Aufschluss darüber, ob eine Verbindung hergestellt werden konnte. Daraufhin wurde der Befehl „is_socket_open()“ verwendet, da der „boolsche“ Rückgabewert Aufschluss darüber gibt, ob ein freier Socket zur Verfügung steht. Wenn beide Werte „1“ oder „true“ sind dann ist die Verbindung hergestellt und die Auslesung beginnt. Durch den Befehl „read_input_registers“ wird nun das Modbus Register ausgelesen. Als Parameter werden das Register, die Anzahl der Auszulesenden Register und die Modbus ID benötigt. Die Modbus ID ist ein Wert der händisch am Modbus Server eingestellt werden kann. Der Standardwert ist hierbei eins. Danach erfolgt durch den Befehl „BinaryPayloadDecoder“ die Umstrukturierung der Bitreihenfolge. Um diese Bitreihenfolge nun als „signed Integer“ darzustellen wurde der Befehl „decode_16bit_int()“ genutzt. Dieser Wert ist theoretisch, gemäß der Modbus-Spezifikation, der erwünschte Wert. Dieser wird in der Größenordnung 0,1 Ampere dargestellt. Da allerdings Ampere Werte benötigt werden, wurde der Wert durch 10 geteilt. Schlussendlich wird zum Abschluss meiner Funktion dieser Wert zurückgegeben.

Die Ausführungszeit beträgt unter 0,5 Sekunden und ist folglich für den Betrieb mit einer Echtzeit Leistungsanpassung geeignet.

Die ausgelesenen Werte sind nicht ganzzahlig, sondern haben Nachkommastellen. Der EEBUS- Converter kann mit den Nachkommastellen behaftete „signed Integer“ verarbeiten, jedoch die Wallbox als Endgerät nicht. Die Modbusspezifikation schreibt ganzzahlige Werte im gültigen Bereich vor.

Es wurde eine Funktion geschrieben, welche diese Abrundung durchführt. Dabei wurde absichtlich auf einen rekursiven und iterativen Ansatz verzichtet. Die Bedingungen wurden mit „if“, „elif“ und „else“ Argumenten umgesetzt. Dies soll jedoch in einer Überarbeitung des Codes in einen rekursiven Ansatz umgewandelt werden. Jedoch erfüllt es zum aktuellen Zeitpunkt die gewünschte Funktionalität.

Ein ausschlaggebendes Kriterium des „Min+PV“ Lademodus ist, dass eine kontinuierliche Versorgung des E-Fahrzeugs gewährleistet wird. Somit müssen drei Bedingungen abgefangen werden.

- Der überschüssige Strom ist über 16 Ampere
- Der überschüssige Strom ist unter 6 Ampere
- der gegebene Wert entspricht nicht dem gewünschten Format

Zunächst wird mit einer „if“ Abfrage jeder Wert gleich oder über 16 auf 16 Ampere gesetzt. Somit werden alle Ströme die größer dem Maximalwert sind hier abgefangen. Die Wallbox könnte Werte bis zu 32 Ampere annehmen, da es diese Box auch in einer 22 kW Version gibt. Da diese Wallboxen fast baugleich sind und sich nur durch ein anderes Leistungsschütz⁴⁷ unterscheiden, ist hier Vorsicht bei dem Setzen der Werte geboten. Die Limitierung des Ladestroms ist nicht hardwareseitig, sondern softwareseitig. Folglich besteht die Möglichkeit, dass durch einen Softwarefehler die Wallbox mehr Leistung abgibt als zulässig wäre. Somit wurde der Maximalwert auf 16 limitiert.

In den folgenden Zwischenabfragen wurde jeweils von 16 absteigend ganzzahlig abgerundet.

Zur Bewältigung der Bedingung, dass immer eine Ladeleistung zur Verfügung gestellt werden muss, wurde mit einem „else“ die Ladeleistung auf 6 Ampere gesetzt. Dies fängt sowohl alle Ströme unter 6 Ampere ab als auch ungültige Eingabewerte und sorgt für einen reibungslosen Betrieb.

Schlussendlich übergibt die Funktion als Rückgabewert einen ganzzahligen Wert zwischen 6-16.

3.6.4 Modbus Register der WB beschreiben und kontrollieren

Der Hersteller des Go-eChargers liefert mit der neu veröffentlichten Modbus TCP Schnittstelle, die ab Firmware Version 040 verfügbar ist, eine offene und leicht zugängliche GitHub Dokumentation⁴⁸. Aus diesen wurden, wie in der Modbus Dokumentation, die notwendigen Informationen extrahiert.

Die Wallbox wird per App mit dem Heimnetzwerk verbunden, hier ist wie in der Problemlösung (Kapitel 2.1.3) darauf zu achten, dass alle Geräte im selben Subnetz des Routers arbeiten. Die vergebene IP-Adresse ließ sich aus der Oberfläche der Fritzbox herauslesen und wurde auch direkt für dieses Gerät blockiert. Dieser Mechanismus bewahrt diese IP-Adresse vor einer Neuvergabe, wenn die WB länger außer Betrieb ist. Als Port der IP-Adresse wird für TCP wieder standardmäßig 502 verwendet.

Die Konstruktion des Verbindungsaufbaus erfolgt äquivalent zur Funktion, die zum Auslesen des Wechselrichters programmiert wurde. Deswegen wird hier auf eine detaillierte Beschreibung des Verbindungsaufbaus verzichtet.

Die Spezifikation beinhaltet 2 Arten von Registern „holding“ und „input“-Register. Holdingregister beinhalten Werte, die von außen nicht veränderbar sind und nur ausgelesen werden können. Inputregister wiederum sind beschreibbar und auslesbar. Alle Registerinhalte werden in dem „Big Endian Format“ gespeichert.

⁴⁷ Quelle: https://www.youtube.com/watch?v=JNRFYtzQPf0&ab_channel=KarlZeilhofer letzter Zugriff am 20.08.22

⁴⁸ Quelle: <https://github.com/goecharger/go-eCharger-API-v1/blob/master/go-eCharger%20Modbus%20TCP%20API%20v1%20DE.md> letzter Zugriff am 20.08.22

```

client = ModbusTcpClient('192.168.188.54', port=502)

if (client.is_socket_open() == False):
    result = client.connect()
    result2 = client.is_socket_open()
    if result2 == False:
        print("Verbindung mit der Wallbox nicht herstellbar")

def sendCurrent(current: int):
    result = client.write_registers(299, current)
    return result

def checkCurrent():
    value1 = client.read_holding_registers(299, 1, unit=1)
    decoder = BinaryPayloadDecoder.fromRegisters(value1.registers, Endian.Big)
    value2 = decoder.decode_16bit_int()
    return (value2)

client.close()

```

Programmcode 2 Funktion zum Schreiben und Überprüfen des PV-Überschuss-Stroms

Die relevanten Register sind das Register 299 und 300. Der Unterschied der beiden Register besteht darin, dass es sich bei dem Register 300 um eine persistente Änderung der Ladeleistung handelt. Wobei Änderungen am Register 299 nach einem Neustart des Gerätes auf den Standardwert zurückgesetzt werden. Da die Wallbox sowohl einen privaten als auch Forschungsnutzen erfüllt ist es wichtig, dass bei Ladevorgängen außerhalb der Forschungszeit der Lademodus „Sofort“ umgesetzt wird. Deswegen wurde sich für das nicht persistente Register 299 entschieden. Die Register halten jeweils „unsigned Integer“-Werte mit einer Länge von 16 Bit.

299	AMPERE_VOLATILE	Holding Register	unsigned integer (16)	1	Ampere Wert für die PWM Signalisierung in ganzen Ampere von 6-32A Wird nicht im EEPROM gespeichert und wird beim nächsten Bootvorgang auf den zuletzt im EEPROM gespeicherten Wert zurückgesetzt. Für Energieregulung
300	AMPERE_EEPROM	Holding Register	unsigned integer (16)	1	Ampere Wert für die PWM Signalisierung in ganzen Ampere von 6-32A Wird im EEPROM gespeichert (max. Schreibzyklen ca. 100.000)

Abbildung 17 Auszug aus der Modbus TCP Dokumentation für den Go-eCharger des Herstellers Go-E

Der Funktion „sendCurrent“ wird der ganzzahlige Integer Wert übergeben. In dieser Funktion wird der Befehl „write_registers(299, current)“ ausgeführt. Der erste Wert der Funktion ist das zu beschreibende Register. Der zweite Wert ist für den Payload des Registers zuständig. Da aus dem Wechselrichter angezeigte Integerwerte herausgelesen und diese dann auch noch ganzzahlig konvertiert werden, ist es nicht vonnöten in der Funktion eine Plausibilitätsprüfung der übergebenen Werte durchzuführen. Der Wert wird mittels der genannten Funktion an die Wallbox übermittelt. Der Rückgabewert dieser Funktion gibt ein Array mit zwei Einträgen wieder. Der erste Eintrag zeigt das beschriebene Register und der zweite Eintrag gibt in „boolescher Form“ an, ob der Wert gesetzt wurde. Wenn der Wert „1“ beträgt wurde das Schreiben erfolgreich ausgeführt.

Um eine unabhängige Prüfung des geschriebenen Wertes zu etablieren, wurde eine separate Auslesefunktion geschrieben. Diese hat die Funktion das Inputregister nach dem Beschreiben auszulesen und den darin gespeicherten Wert zurückzugeben. Die Funktion „checkCurrent()“ liefert den aktuellen Eintrag des Registers 299 zurück. Der grundlegende Funktionsaufbau entspricht dem der Auslesefunktion „CurrentL1“ für den Wechselrichter. Da die charakteristischen Eigenschaften des Registers dieselben sind wie bei der genannten Funktion, konnte diese ohne Abwandlung hier etabliert werden. Der einzige Unterschied bestand darin, dass das Register ersetzt werden musste.

Diese Funktion ist nicht Bestandteil des Workflows der Datenverarbeitung. Sie dient nur dem Zweck zu überprüfen, ob die eingetragenen Werte auch wirklich eingetragen wurden.

Wird nun eine Verkettung der Funktionsaufrufe etabliert so lässt sich untersuchen, wie die direkte Verkettung der einzelnen Bausteine funktioniert.

Zunächst war die Verkettung der Funktionsaufrufe zu untersuchen. Diese wurde in einem UML Diagramm aufgetragen, um zu überprüfen in welcher Reihenfolge die Funktionen aufgerufen werden müssen.

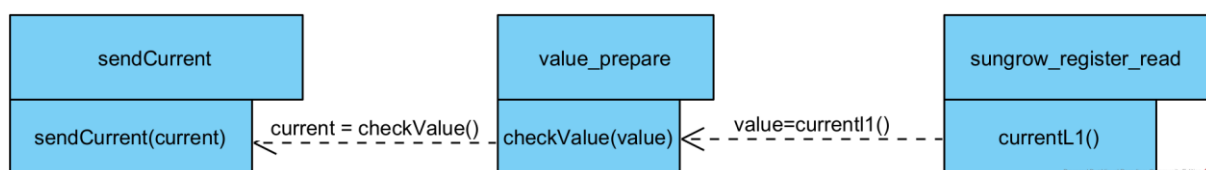


Abbildung 18 UML Diagramm direkte Weiterleitung der Informationen

Aus dieser Untersuchung wurde für Testzwecke eine Test-Funktion geschrieben, die diese Reihenfolge einhält. Nach jedem Funktionsaufruf wird der Wert über „print(Variablenname)“ ausgegeben. Dazu wurde noch ein Hinweistext verfasst, der wiedergibt, was diese Werte Aussagen.

```

def set_Current_from_inverter():
    payload1 = WR_Sungrow_Hybrid_SH_Modbus.sungrow_register_read.CurrentL1()
    payload2 = WR_Sungrow_Hybrid_SH_Modbus.sungrow_register_read.CurrentL2()
    payload3 = WR_Sungrow_Hybrid_SH_Modbus.sungrow_register_read.CurrentL3()
    payload0 = (payload1 + payload2 + payload3) / 3
    payload = WB_GoEChargerV2.valuePrepare.checkValue(payload0)
    check = WB_GoEChargerV2.SendCurrent.sendCurrent(payload)
    check2 = WB_GoEChargerV2.SendCurrent.checkCurrent()
    print("Die Überschüsströme sind L1:", payload1, "A L2:", payload2, "A
L3:", payload2, "A")
    print("Der abgerundete Durchschnitt zwischen (6-16) ist", payload)
    print(check)
    if payload == check2:
        print("Der Wert", payload, "wurde erfolgreich auf", check2, "gesetzt")
    else:
        print("Der Wert", payload, "wurde NICHT erfolgreich auf", check2, "ge-
setzt")
set_Current_from_inverter()

```

Programmcode 3 Testfunktion Test.py mit allen Komponenten

Zunächst werden mit den bereits bekannten Funktionen die einzelnen Überschussströme ausgelesen und auf eine Variable abgespeichert. Danach wird aus allen 3 Strömen der Mittelwert gebildet.

Dies hat folgenden Hintergrund: Aufgrund der ungleichen Lastverteilung von elektrischen Verbrauchern in Bestandsgebäuden, kann der einzelne Wert zu den anderen sehr stark schwanken. Um ein natives PV-Überschussladen zu gewährleisten ist die Berücksichtigung dieses Zustandes wichtig. Jedoch arbeiten die verwendeten Zweirichtungszähler des Energieversorgers im Schaltschrank nach einem anderen Prinzip. Sie summieren alle vorhandenen Überschussströme auf. Erst wenn der Netzbezug in der Summe grösser ausfällt als die PV Einspeisung, detektiert der Zähler dies und rechnet es als externen Verbrauch ab. Somit bleibt es eine moralische Frage, ob man unter ökologischen Aspekten lieber seinen eigenen erzeugten Strom nutzt oder unter wirtschaftlichen Aspekten die Unschärfe des Zählers nutzt. Die ökologisch vertretbarste Variante wäre den kleinsten Wert des PV-Überschusses zu nehmen. Der wirtschaftlichste Ansatz wäre kontinuierlich abzugleichen, ob die Summe der Einspeisung größer ist als die des Netzbezuges und zu versuchen diesen Zustand in Übergangsnähe zum anderen Zustand zu halten.

Die Entscheidung fiel auf den Mittelwert, es ist weder der ökologischste noch wirtschaftlichste Fall. Für Testzwecke ist dies aber vollkommen ausreichend. In dem fertigen Converter wird der ökologische Ansatz genutzt.

Nach der Mittelwertbildung entsteht eine kommabehaftete Zahl die im nächsten Schritt im Intervall zwischen 6-16 abgerundet wird. Der Wert wird nun zur WB gesendet und danach die korrekte Annahme des Wertes durch eine Auslesefunktion überprüft. Passend zu den einzelnen Bereichen wurde ein Ausgabetext vorkonfiguriert.

Bei einem erfolgreichen Test wird folgendes ausgegeben:

```
/home/maxim/PycharmProjects/pythonProject1/venv/bin/python
Die Überschussströme sind L1: 4.1 A L2: 4.1 A L3: 4.1 A
Der abgerundete Durchschnitt zwischen (6-16) ist 6
WriteMultipleRegisterResponse (299,1)
Der Wert 6 wurde erfolgreich auf 6 gesetzt

Process finished with exit code 0
```

Abbildung 19 Ausgabe des Testcodes

Die ersten Ausgaben sind Kontrollen der Schlüssigkeit der gegebenen Werte. Abweichungen würden hier sofort auffallen. Zum Schluss wird der Vergleich des gesendeten Wertes und des ausgelesenen Wertes an der Wallbox durchgeführt und ausgegeben, ob dieser Wert, wie gewünscht, derselbe ist.

Auf Grundlage dieser funktionierenden Implementierung ist es für den nächsten Schritt vonnöten Wallbox und Wechselrichter nicht direkt miteinander kommunizieren zu lassen, sondern nun die EEBUS-Converter zu nutzen.

3.7 Grundlage der Kommunikation mit den EEBUS Convertern

Wie bereits vorher ausgeführt findet die Kommunikation zwischen den EEBUS Convertern und dem CEM selbsttätig statt. Diese Anwendungen werden über das Terminal gestartet und arbeiten im Testsetup unter Linux Focal Fossa vollkommen autark.

Einmal gestartet bauen die Converter selbstständig eine Verbindung zum CEM auf. Die Kommunikation zwischen Integrations-Code / Businesslogik wird durch ein Web-Socket hergestellt.

An den einzelnen EEBUS Applikationen werden folgende essenzielle Informationen ausgegeben:

- Verbindung zur Gegenstelle erfolgreich aufgebaut
- Welche „Use Cases“ unterstützt die Gegenstelle
- Absenden und Empfangen von JSON Nachrichten

Zusätzlich gibt der CEM noch folgende Informationen aus:

- Empfangener Payload mit den Kerninformationen als Direktausgabe.
- Verarbeitung der Empfangenen Daten, in diesem Fall die unverarbeitete Weiterleitung

Durch die Web-Sockets lassen sich JSON Nachrichten als „strings“ übertragen.

Jedes EEBUS Szenario eines „Use Case“ hat einen klar dokumentierten JSON Datenaufbau, den er als Empfänger erwartet oder als Sender an die Gegenstelle verschickt.

Die Aufgabe besteht aus folgenden Teilschritten:

Für den WB-Adapter, dargestellt durch den sog. „grid_server“:

- Eine Funktion zu schreiben die nach einem klar definierten Schema eine JSON Instanz erstellt und die gegebenen PV-Überschusswerte aufnehmen kann
 - Diese in einen String zum Absenden konvertieren
- Eine Funktion als „main“-Methode zu programmieren, die Echtzeitlauffähig ist und eine Web-Socket Verbindung zum „grid_server“ herstellt und hält
 - Den generierten String in regelmäßigen Abständen abschickt
 - Überprüft ob Änderungen an dem PV-Überschuss-Strom vorliegen und nur dann eine Nachricht absendet
 - Beim Beenden der Anwendung soll die Socket Verbindung zum WR über Modbus und zum „grid_server“ technisch korrekt beendet und die Verbindung nicht unnötig offengehalten werden

Für den WR-Adapter, dargestellt durch den sog. „emobility_server“:

- Eine zweite Funktion als „main“-Methode zu etablieren die ebenfalls Echtzeitlauffähig ist, eine Web-Sockets Verbindung zum „emobilty_server“ herstellt
 - Auf eingehende Nachrichten wartet und diese beim Eintreffen empfängt
 - Die Nachricht in ein „Python Dictionary“ umwandelt und die nötigen Ladeströme ausliest
 - Die Werte nach dem niedrigsten Ladestrom filtert und diesen an die WB versendet
 - Überprüft, ob der gesendete Wert auch gesetzt wurde
 - Beim Beenden der Anwendung soll die Socket Verbindung zum WR über Modbus und zum „grid_server“ technisch korrekt beendet und die Verbindung nicht unnötig offengehalten werden

3.7.1 Aufbau einer genormten JSON Instanz

Die erste Aufgabe besteht darin eine genormte JSON Instanz nach der gegebenen KEO Dokumentation für den „grid_server“ zu erstellen. Wenn die Datei nicht den Aufbau aus der Vorgabe hat, dann würde der Inhalt nicht durch den Converter verarbeitet werden können. In den beigefügten KEO Dokumenten gab es Beispiele, die verwendet werden konnten. Die einzige Bedingung war, dass die ID der Nachricht jedes Mal eine andere sein musste.

```
{
  "type": "de.keo-connectivity.grid.mgcp.momentaryCurrent",
  "source": "d:_n:KEO_GCP",
  "id": "3604d9a8-7c0e-4e0a-bb06-df2d72f95e9b",
  "specversion": "1.0",
  "data": {
    "actorId": "d:_n:KEO_GCP",
    "momentaryCurrent": {
      "a": {
        "value": {
          "number": -1,
          "scale": 0
        },
        "state": "normal"
      },
      "b": {
        "value": {
          "number": 5,
          "scale": 0
        },
        "state": "normal"
      },
      "c": {
        "value": {
          "number": -35,
          "scale": -1
        },
        "state": "normal"
      }
    }
  }
}
```

Programmcode 4 Beispielcode JSON GCP Szenario 5

```
def createJSON(current_L1, current_L2, current_L3):
    value_uuid = str(uuid.uuid4())

    json_object = {
        "type": "de.keo-connectivity.grid.mgcp.momentaryCurrent",
        "source": "legacy Inverter",
        "id": value_uuid,
        "specversion": "1.0",
        "data": {
            "actorId": "MyGcp",
            "momentaryCurrent": {
                "a": {
                    "value": {
                        "number": current_L1,
                        "scale": 0
                    },
                    "state": "normal"
                },
                "b": {
```

Programmcode 5 Funktion zum Erstellen einer JSON Nachricht

Es wurde sich an dem Beispiel⁴⁹ orientiert und so die grundlegende Struktur des JSON aufgebaut. Zunächst wird durch einen uuid4 Generator die ID erstellt und als String an der richtigen Stelle in der JSON-Datei eingefügt.

Die Phasen L1-L3 sind hier als a-c gekennzeichnet. Unter dem Unterpunkt „value“ gibt es „number“ und „scale“. Gemäß der Vorgabe sind keine kommabehafteten Integer erlaubt und müssen als Festkommadarstellung unter „number“ eingefügt werden. Der „scale“ sagt **10^{scale}** aus. D.h. der kommabehaftete Integer muss um **10^x** verschoben werden um die Nachkommastellen = 0 zu bekommen. Der Wert x wird vorzeichenbehaftet bei „scale“ eingetragen.

Zum Beispiel kann 3 dargestellt werden als „number“: 3, „scale“: 0 oder „number“: 30, „scale“: -1 oder 3,5 kann dargestellt werden als „number“: 35 „scale“: -1 oder „number“: 350, „scale“: -2.

Da in der Implementierung eine Rundungsfunktion etabliert ist, sind ganzzahlige Integer in der Größe Ampere vorhanden und können unter „number“ ohne Konvertierung hinterlegt werden. Beim „scale“ wird 0 eingetragen. Dies wird für „a“, „b“ und „c“ durchgeführt. Zum Abschluss ist es notwendig fürs Absenden die JSON Instanz zu einem String zu konvertieren und zurückzugeben. Dies wird durch den Befehl „return json.dumps(json_object)“ erledigt.

⁴⁹ Aufgrund der unterschriebenen **Non-Disclosure Agreement** (NDA) wird auf die Verbreitung KEO spezifischer Dokumentation verzichtet.

Die JSON Instanz als String wäre nun in Kombination mit den vorher erläuterten Funktionen bereit zum Absenden.

3.7.2 Aufbau der Echtzeitfunktion

Zur Vereinheitlichung und Vereinfachung der Lesbarkeit wird nun die Grundlegende Funktionalität der echtzeitfähigen Web-Socket Client Implementierung generalisiert dargestellt. Die Ergänzungen für die Kommunikation mit dem „grid_server“ und dem „emobility_server“ werden im Anschluss erläutert.

```
def setup():
    ws = websocket.WebSocketApp("ws://localhost:41062")
    return ws
def on_message(ws, message):
    # CODE for grid_server communication
    return message
def on_error(ws, error):
    print(error)
def on_close(ws, close_status_code, close_msg):
    print("### closed ###")
def on_open(ws):
    print("Opened connection")
def signal_handler(signal, frame):
    global interrupted
    interrupted = True
interrupted = False
if __name__ == "__main__":
    signal.signal(signal.SIGINT, signal_handler)
    websocket.enableTrace(True)
    ws = websocket.WebSocketApp("ws://localhost:41064",
                                on_open=on_open,
                                on_message=on_message,
                                on_error=on_error,
                                on_close=on_close)

    t = Thread(target=ws.run_forever)
    t.daemon = True
    t.start()
    while (True):
        if (interrupted):
            print("Shutting down")
            break
    #CODE for grid_server kommunikation
    ws.close()
    t.join()
    WB_GoEChargerV2.SendCurrent.client.close()
```

Programmcode 6 Grundkonstrukt "Web-Socket Client"

Bei der Implementierung eines echtzeitlauffähigen Web-Socket Client wurde die gleichnamige Bibliothek verwendet.⁵⁰ In der Dokumentation gab es ein lauffähiges Beispiel für eine „long-live“-Echtzeitverbindung⁵¹. Es wurde am Anfang der Funktion ein „Signal-Handler“ ergänzt, der das Behandeln von Interrupts vorbereitet. Die Funktion „enableTrace“ wird eingeschaltet gelassen, um im Terminal eine genaue Ausgabe über die Handlungen des Web-Sockets zu erhalten. In der grundlegenden Konfiguration wurde „localhost“ als IP-Adresse hinterlegt, da die Converter allesamt auf demselben Endgerät laufen. Gleichzeitig werden weitere Funktionen geladen die aktionsbasiert ausgeführt werden. Daraufhin wird ein „Daemon Threads“ vorbereitet und gestartet. In der „while“-Schleife wird ein „Interrupt Handler“ ergänzt der die Ausführung der „while“-Schleife bei Unterbrechungen verlässt. Danach werden die ordnungsgemäßen Schließungen der Sockets durchgeführt und der „daemon Thread“ arbeitet durch das „join()“ solange weiter bis die Aufgaben erfüllt sind. Im Bereich der „while“-Schleife wird zukünftig der Code für die Kommunikation mit dem „grid_server“ implementiert.

Hier dargestellt sind die aktionsbasierten Funktionen. Für die Hauptfunktion ist zum Empfangen des JSON Strings die Unterfunktion „on_message“ zuständig. Zusätzlich ergänzt wurde der „signal-handler“. Beim Aufrufen dieser Funktion wird ein globales Flag gesetzt und das Programm kontrolliert beendet.

Wenn die „main“ nun gestartet wird, dann wird eine Web-Socket-Verbindung zum angegebenen Server hergestellt und gehalten. Nachrichten könnten nun versendet und empfangen werden.

3.7.3 Ergänzungen für den Betrieb mit dem „grid_server“

Folgende Ergänzungen wurden dem oben beschriebenen Code für den Betrieb mit dem „grid_server“ gemacht.

Zunächst wurden in der „main“-Funktion drei Variablen angelegt „old-payload“1-3. Auf diesen wird nach jeder Versendung eines JSON-Strings mit den, in dem Durchgang gültigen Werten des PV-Überschuss-Stroms, überschrieben. Bei der nächsten Prüfung soll mit diesen Werten verglichen werden, ob sich der Wert geändert hat. Dies dient dem Zweck nicht jeden Schleifendurchgang eine Nachricht an den „grid_server“ abzusetzen und den Datenverkehr zu minimieren.

⁵⁰ Quelle und Dokumentation: <https://Web-Socket-client.readthedocs.io/en/latest/> letzter Zugriff am 20.08.2022

⁵¹ Quelle: <https://pypi.org/project/Web-Socket-client/> -> Long-lived Connection letzter Zugriff am 20.08.2022

```

value = update(ws, old_payload1, old_payload2, old_payload3)
    if value[0] != old_payload1:
        old_payload1 = value[0]
    if value[1] != old_payload2:
        old_payload2 = value[1]
    if value[2] != old_payload3:
        old_payload3 = value[2]
    time.sleep(5)

```

Programmcode 7 Ergänzungen der "while"-Schleife

```

def update(ws, old_payload1, old_payload2, old_payload3):
    payload_L1 = WR_Sungrow_Hybrid_SH_Modbus.sungrow_register_read.CurrentL1()
    payload_L2 = WR_Sungrow_Hybrid_SH_Modbus.sungrow_register_read.CurrentL2()
    payload_L3 = WR_Sungrow_Hybrid_SH_Modbus.sungrow_register_read.CurrentL3()
    payload_L1_1 = WB_GoEChargerV2.valuePrepare.checkValue(payload_L1)
    payload_L2_1 = WB_GoEChargerV2.valuePrepare.checkValue(payload_L2)
    payload_L3_1 = WB_GoEChargerV2.valuePrepare.checkValue(payload_L3)
    if old_payload1 != payload_L1_1 or old_payload2 != payload_L2_1 or
old_payload3 != payload_L3_1:
        json_string = WR_Sungrow_Hybrid_SH_Modbus.makeJSON.createJSON(payload_L1_1, payload_L2_1, payload_L3_1)
        ws.send(json_string)

    old_payload1 = payload_L1_1
    old_payload2 = payload_L2_1
    old_payload3 = payload_L3_1
    return old_payload1, old_payload2, old_payload3

```

Programmcode 8 Ergänzte Update Funktion

Die Funktion „update“ wird mit den Werten des vorherigen Durchgangs gespeist und geöffnet. Zunächst werden mit den bereits beschriebenen Funktionen die Werte aus den Registern gelesen und im Bereich zwischen 6-16 abgerundet. Danach werden die frisch ausgelesenen Werte, mit denen aus den vorangegangenen Werten verglichen. Sollte sich eine Änderung ergeben, dann wird ein JSON String mit den aktuellen Werten erzeugt und mit dem Befehl „send(Dateiname)“ an den Web-Socket-Server versendet. Als Rückgabewert liefert die Funktion die ausgelesenen Werte als Array mit drei Einträgen. In der „while“-Schleife wird nun über den Rückgabewert geprüft, welcher Wert sich von „old_payload“ 1-3 geändert hat und dieser wird dann aktualisiert. Werden alle drei EEBUS Applikationen gestartet und diese Funktion ausgeführt, dann schickt die Funktion zunächst den aktuell ausgelesenen Wert ab und überprüft alle fünf Sekunden ob Änderungen vorliegen. Wenn dies vorliegt, dann wird ein neuer JSON String versendet.

Im Terminal wird folgendes angezeigt:

```
-----  
--- response header ---  
HTTP/1.1 101 WebSocket Protocol Handshake  
Server: 1.3.2  
Date: Fri, 26 Aug 2022 11:46:20 GMT  
Access-Control-Allow-Origin: *  
Upgrade: websocket  
Connection: Upgrade  
Sec-WebSocket-Accept: +86i7KHeDzhkZrDykLXCmpvU9Zs=  
-----  
++Sent raw: b'\x81\xfe\x01\x89;\xd4\xa6k@\xf6\xd2\x  
++Sent decoded: fin=1 opcode=1 data=b'{"type": "de.  
++Rcv raw: b'\x81~\x00\xd0{"data":{"errorNumber":0}  
++Rcv decoded: fin=1 opcode=1 data=b'{"data":{"erro  
{"data":{"errorNumber":0},"id":"1dd7bea4-2820-dbbe-
```

Abbildung 20 Terminal Ausgabe während des Betriebes der Web-Socket Verbindung

Hierbei werden die Verbindungseigenschaften und die ein- und ausgehenden Nachrichten durch den automatischen Aufruf der aktionsabhängigen Funktionen aufgerufen. Die empfangenen und gesendeten Nachrichten werden in kodierter und dekodierter Schreibweise dargestellt.

Im Terminal des „grid_server“ wird ausgegeben, dass eine Nachricht empfangen und diese an den CEM weitergeleitet wurde.

Der CEM wiederum gibt nun die relevantesten Informationen aus und leitet wie vorprogrammiert die Nachricht an den „emobility_server“ weiter.

```
[mock_cem] 2022-08-26 13:46:08 INFO Exactly one EV is present  
OSCEV limits to EV d:_n:KEO_json_emobility_server/1/1/.  
[mock_cem] 2022-08-26 13:46:08 INFO OSCEV limits were sent.  
[keo_spine] 2022-08-26 13:46:08 INFO Handling pending datagr  
47859_KEO-UcApi-Grid-Emob-Client/2/1'. Message counter: 166  
[mock_cem] 2022-08-26 13:46:20 INFO Received phase specific  
(MGCP sc. 5):  
    Phase A: (6 * 10^0) A  
    Phase B: (6 * 10^0) A  
    Phase C: (6 * 10^0) A  
  
[mock_cem] 2022-08-26 13:46:20 INFO Exactly one EV is present  
OSCEV limits to EV d:_n:KEO_json_emobility_server/1/1/.  
[mock_cem] 2022-08-26 13:46:20 INFO OSCEV limits were sent.  
[keo_spine] 2022-08-26 13:46:20 INFO Handling pending datagr  
47859_KEO-UcApi-Grid-Emob-Client/2/1'. Message counter: 170
```

Abbildung 21 Ausgabe am CEM Terminal beim Erhalten einer Nachricht des "grid_servers"

Der Empfang der vom CEM weitergeleiteten Daten wird am „emobility_server“ bestätigt. Hier wird keine Nachricht an den WEB-Socket abgesetzt, da noch keine Verbindung zu einem Web-Socket-Client besteht.

3.7.3 Ergänzungen für den Betrieb mit dem „emobility_server“

Folgende Ergänzungen wurden im dem oben beschriebenen Code für den Betrieb mit dem „emobility_server“ gemacht.

Die „while“-Schleife blieb unverändert, wie die Grundfunktion. Wie bereits genannt wurde nur die „on_message“- Funktion komplett überarbeitet.

```
def on_message(ws, message):
    data1 = json.loads(message)
    current_L1 = data1["data"][ 'limits' ][ 'a' ][ 'value' ][ 'number' ]
    current_L2 = data1["data"][ 'limits' ][ 'b' ][ 'value' ][ 'number' ]
    current_L3 = data1["data"][ 'limits' ][ 'c' ][ 'value' ][ 'number' ]
    check_value = 0

    if current_L1 <= current_L2 and current_L1 <= current_L3:
        WB_GoEChargerV2.SendCurrent.sendCurrent(current_L1)
        check_value = current_L1

    elif current_L2 <= current_L1 and current_L2 <= current_L3:
        WB_GoEChargerV2.SendCurrent.sendCurrent(current_L2)
        check_value = current_L2

    elif current_L3 <= current_L1 and current_L3 <= current_L1:
        WB_GoEChargerV2.SendCurrent.sendCurrent(current_L3)
        check_value = current_L3

    check_value2 = WB_GoEChargerV2.SendCurrent.checkCurrent()

    if check_value2 == check_value:
        print("Der Wert", check_value2, "wurde erfolgreich als Ladestrom drei-
        phasig übernommen")

    else:
        print("FEHLER")
```

Programmcode 9 Ergänzung des Codes on_message

Der Funktion wird „message“ als Übertragungsinhalt nach dem Empfangen übergeben. Durch die Funktion „json.loads(message)“ wird der String in ein „Python Dictionary“ übersetzt. Durch die Dokumentation des „E-Mobility „Use Case“ Optimization of Self-Consumption During EV Charging“ war der systematische Aufbau dieser JSON Instanz bekannt. Deswegen konnte durch ein verschachteltes Auslesen des „Python Dictionarys“ der gesuchte Wert extrahiert werden.

Um einen Netzbezug weitestgehend zu vermeiden wurde der niedrigste Wert durch „if“-Abfragen extrahiert und an die WB, mit der bereits erklärten Funktion, gesendet.

Danach wird überprüft, ob der Wert erfolgreich gesetzt wurde. Wenn alles korrekt ausgeführt wurde, dann wird folgendes im Terminal ausgegeben.

```
-----  
--- response header ---  
HTTP/1.1 101 WebSocket Protocol Handshake  
Server: 1.3.2  
Date: Fri, 26 Aug 2022 11:46:17 GMT  
Access-Control-Allow-Origin: *  
Upgrade: websocket  
Connection: Upgrade  
Sec-WebSocket-Accept: AfzpxN2LdeeIKtWbTK8xqgD3N+E=  
-----  
Opened connection  
++Rcv raw: b'\x81~\x01g{"data":{"actorId":"ev","limits":{"a":{"ac  
++Rcv decoded: fin=1 opcode=1 data=b'{"data":{"actorId":"ev","lim  
Der Wert 6 wurde erfolgreich als Ladestrom dreiphasig übernommen
```

Abbildung 22 Terminal Ausgabe nach erfolgreichem Durchlauf

Wenn diese Ausgabe erscheint, dann hat die Weiterleitung der Informationen funktioniert.

3.8 Bewertung der Implementierung

Die Ausführungszeit der Implementierung vom Auslesen der Modbus-Daten bis zu der Kontrolle der WB ist in den Messungen unter einer Sekunde und funktioniert reibungslos. Die Auslesefunktion des WR erfolgt alle fünf Sekunden, diese Abtastrate lässt sich auch auf eine Sekunde hochsetzen ohne, dass die Implementierung Fehler aufweist. Realitätsnah betrachtet wäre auch eine höhere Abtastrate legitim. Hierbei hängt es vom Bewölkungsgrad und dem Wechselintervall Sonne zu Schatten ab, wie hoch die Abtastfrequenz zu wählen ist.

Da das Grundsetup steht, wäre es kein großer Aufwand den implementierten Lademodus „Min+PV“ durch weitere Fahrzeugunabhängige Lademodi zu ergänzen. Jedoch wäre an dieser Stelle eine GUI von Vorteil, in der alle, aktuell im Code statisch vergebenen Daten, dynamisch und änderbar eingeben werden können.

Die Modbus-Funktionen wären zu überarbeiten, um auch hier ein „Deamon Threading“-Betrieb durchführen zu können, um im Hintergrund als Systemdienst lauffähig zu sein. Präzise „Interrupt“-Ausgabe wären für eine effiziente Fehlersuche zu ergänzen.

Des Weiteren wäre eine Modbus über TLS Verbindung zu testen, ob die WB oder der WR auch verschlüsselt kommunizieren können. Damit wären sämtliche Übertragungen verschlüsselt.

Ein „logging“-Mechanismus fehlt noch vollständig, um unerwartete Seiteneffekte zu erkennen und bearbeiten zu können.

Kapitel 4 Schlussbetrachtung

4.1 Zusammenfassung

Auf der Basis einer geplanten Eigenverbrauchsoptimierung der Solaranlage mit Großverbrauchern wurde die Idee geboren, die vorhandene Wallbox in Kombination mit dem E-Auto wirtschaftlicher und ökologisch vertretbarer zu nutzen. Als erster Schritt wurde die Einbindung des Sungrow SG und SH Wechselrichters in die Open Source Software OpenWB implementiert und grundlegende Einstellungsprobleme beseitigt. OpenWB funktionierte danach mit allen Komponenten, die im Testnetzwerk zur Verfügung standen, einwandfrei. Jedoch wurde und wird der EEBUS Standard von der Open-Source Software nicht unterstützt. Dies geändert werden und im Rahmen dieser Machbarkeitsstudie wurde erforscht, ob EEBUS auch mit „legacy“ Geräte verfügbar zu machen sei. Dabei wurde mir von dem Unternehmen KEO, ein Stack Entwickler zur Betreuung und Unterstützung meiner Bachelorarbeit, zur Verfügung gestellt.

Die vielen praktischen und methodischen Hindernisse konnten mit einer intensiven Untersuchung der „Library“-Dokumentation gelöst werden. Viele Beispielprojekte und ausführliche Beschreibungen trugen zum Weg der Lösungsfindung bei. Besonders hilfreich waren die vielen gedanklichen Stützen des fachlichen Betreuers von KEO, Robert Spielmann. Diese haben sehr effizient zur Lösungsfindung beigetragen.

Zuerst wurde der WR und die WB mit eigenem Code nutzbar gemacht. Danach wurden unter zur Hilfenahme der KEO Dokumentation und mit Hinweisen des Betreuers dynamisch die JSON Instanz erstellt und wieder rückkonvertiert, um die nötigen Daten zu extrahieren. Die Echtzeitanwendungen für das Versenden und Empfangen der Nachrichten waren der Kern der Kommunikation mit der EEBUS Welt. Nach dessen Fertigstellung wurden zwei „Use Cases“ der EEBUS Welt für „legacy“ Geräte zugänglich gemacht. Es wird als Erfolg bewertet, „legacy“ Geräte mit EEBUS zu koppeln. Sowohl das logisch praktische Denken an unterschiedlichsten Schnittstellen als auch das Arbeiten mit hochtechnisch geschriebener Dokumentation haben gefordert aber insgesamt das Denken in dieser thematischen Materie gestärkt.

4.2 Resümee der Implementierung

Auf dem Weg zur fertigen Implementierung wurde sich selbstständig aneignet, dass es essenziell wichtig ist, zuerst vollständig die Herstellerdokumentation zu einer Schnittstelle zu lesen und erst dann anzufangen zu arbeiten. Ansonsten hätte man sehr viel Zeit darauf verwenden können einen Pseudofehler unplausibler Wertauslesung korrelieren zu wollen, nur weil man den Sonderfall nicht bemerkt, dass die Register um eine Adresse verschoben sind oder „little“ und „big“-Endian auch mal

gemischt nacheinander verwendet werden. Durch die ersten Erfolge beim Auslesen und Schreiben von Modbus Registern gestärkt, kam sehr viel Geschwindigkeit ins Projekt. Mit jeweils einem Ansatz war es nun möglich einen komplexen Code, auch unter Berücksichtigung von Sonderfällen, zu programmieren.

Das Wissen im Arbeiten mit Python und den Konzeptstrukturen der höheren Programmierung ist um ein Vielfaches angestiegen. Sowohl die strukturierte Lösungsfindung bei der Softwarekonfiguration als auch bei der Softwareentwicklung selbst ist bedeutend präziser geworden und das Problemausschlussverfahren wurde weiter geschärft.

Als Ergebnis der Implementierung wird die Bachelorarbeit als voller Erfolg im Rahmen der persönlichen Weiterbildung geschätzt.

4.3 Resümee der gesteckten Ziele

Ziel der Bachelorarbeit war es eine Konzeption und Realisierung eines EEBUS Community Converters zu ermöglichen. Dieses Ziel ist Teilerfüllt worden. Es wurde bewiesen, dass „legacy“ Geräte im EEBUS Standard verwendbar sind.

Das hochgesteckte Endziel, einen vollumfänglichen interaktiven Converter, aufzubauen wurde nicht erreicht. Dieser müsste eine interaktiven GUI haben, die alle von OpenWB oder EVCC interpretierten WR, Speicher, WB, E-Autos und Smart Home Geräte steuern kann.

Der Grundstein hierfür ist gelegt, doch es bedarf noch einigen Aufwand, bis eine offen nutzbare Schnittstelle für diese „legacy“ Geräte im EEBUS Standard geschaffen ist.

4.4 Ausblick

Aktuell gibt es noch keine von Herstellern geförderte Open Source Implementierung. Die verwendeten lauffähigen Anwendungen wie der CEM sind Eigentum von KEO und nur durch deren API nutzbar. Folglich ist die Untersuchung in dieser Bachelorarbeit sicherlich lehrreich und für Einblicke zielführend, doch für einen freien EEBUS Community Converter benötigt es einen Open Source Stack für den CEM, sowie für die wichtigsten „Use Cases“ aus jeder Gerätekategorie. Zwei Anwendungsfälle muss der Open Source Ansatz abdecken können, damit EEBUS frei genutzt werden kann.

1. EEBUS Gerät (WR, WP, E-Auto, WB) <-> Fremd EMS

Hierbei sollen zum Beispiel Wärmepumpen, die den EEBUS Standard unterstützen können mit einem fremd EMS, das nativ kein CEM ist, kommunizieren können. Es müsste ein CEM Modul geladen werden, dass durch das vorhandene EMS gesteuert

werden kann. Jedoch müsste hierbei die native Verbindung aller EEBUS Geräte mit dem CEM möglich sein.

2. „legacy“ Gerät -> EEBUS Converter -> CEM

Hier sollen „legacy“ Geräte durch einen Konverter in ein EEBUS Netzwerk integriert werden. In dieser Bachelorarbeit wurde der Beweis erbracht das dies für mindestens zwei „Use Cases“ umsetzbar war.

Grundlegend wäre der zukünftige Ansatz, per Auswahlmenü Geräte zu selektieren, die dann selbständig einen Converter Thread starten. Dieser Thread arbeitet dann autark als konvertiertes EEBUS Gerät und versorgt den CEM mit Informationen.

Während der Abschlussphase meiner Bachelorarbeit haben sich aus EVCC heraus zwei Open Source EEBUS Ansätze⁵² etabliert. Beide haben einen funktionalen Basis Ausstattung, die zumindest eine vereinfachte Lauffähigkeit für einen „Use Case“ in Verbindung mit einem eigenem CEM kreieren.

Der aktuelle Implementierungsstand einer der Open Source EEBUS Stacks ist:

Roadmap - Spine specification implementation

General request processing

- ☒ Request / Handle acknowledgement
- ☒ Use maximum response delay to timeout requests
- ☒ Send error result when processing failed
- ☐ Sending heartbeats

Node Management

- ☐ Detailed Discovery
 - ☒ Request and process full data
 - ☒ Response full data
 - ☐ Request and process partial data
 - ☐ Response partial data
 - ☐ Notify subscribers
- ☐ Destination List
 - ☐ Request and process full data
 - ☒ Response full data
 - ☐ Request and process partial data
 - ☐ Response partial data
 - ☐ Notify subscribers
- ☐ Binding
- ☐ Subscription
 - ☒ Add subscription
 - ☒ Delete subscription
 - ☐ Request and process full data
 - ☒ Response full data
 - ☐ Request and process partial data
 - ☐ Response partial data

Use Case Discovery

- ☐ Use Case Discovery
 - ☒ Request and process full data
 - ☒ Response full data
 - ☐ Request and process partial data
 - ☐ Response partial data

General feature implementation

- ☒ Request and process full data
- ☒ Response full data
- ☐ Request partial data
- ☐ Process partial data
 - ☒ Delete Selectors
 - ☒ Update Selectors
- ☐ Elements
- ☐ Response partial data
- ☐ Process write call
- ☐ Request subscription
- ☒ Notify subscribers
- ☐ Handle incoming error results
- ☒ Handle incoming success results

Feature with partial data support

- ☐ ElectricalConnection

Abbildung 23 Roadmap EEBUS Server SPINE Implementierung⁵³

Die Zeit wird zeigen, ob sich einer der beiden Ansätze implementierungsreif entwickelt und ob die Stack Entwickler Bosch, das Fraunhofer Institut oder KEO sich an einer Lizenzrechtlich freien Implementierung für die private Nutzung beteiligen.

⁵² Quelle: <https://github.com/evcc-io/eebus> und <https://github.com/DerAndereAndi/eebus-go> letzter Zugriff am 25.08.2022

⁵³ Quelle: <https://github.com/DerAndereAndi/eebus-go-cem> letzter Zugriff am 25.08.2022

Kapitel 5 Anhang

5.1 Abbildungsverzeichnis

Abbildung 1 Unabhängigkeitsrechner HTW Berlin Off- Grid“ Berechnung PVGIS EU Tool	8
Abbildung 2 Auszug aus der „MySolarCloud“ APP vom 03.06.2022	14
Abbildung 3 Go-eCharger App Oberfläche	14
Abbildung 4 Modbus Read Anfrage Modbus im OSI-Modell	16
Abbildung 5 Trennung der Teilnetze im Heimnetzwerk	17
Abbildung 6 Debug Fehlerrückmeldung der OpenWB	18
Abbildung 7 Auszug aus den Einstellungen des Sungrow Sg WR	18
Abbildung 8 EEBUS nutzende oder unterstützende Unternehmen.	25
Abbildung 9 EEBUS SHIP/SPINE/APP Übersicht OSI-Modell Übersicht EEBUS	26
Abbildung 10 Konstruktionsaufbau EEBUS	27
Abbildung 11 EEBUS Spezifikation im Detail SPINE Nachricht Überblick	28
Abbildung 12 Exemplarischer Aufbau SPINE Devices SPINE als Toolbox	28
Abbildung 13 Exemplarischer APP Auszug Vaillant	31
Abbildung 14 Schematische Darstellung Kommunikation EMS -> WB	34
Abbildung 15 Kommunikationsentwurf des Testnetzwerks	35
Abbildung 16 OpenWB Modbus.py Implementierungsausschnitt	38
Abbildung 17 Auszug aus der Modbus TCP Dokumentation für den Go-eCharger des Herstellers Go-E	42
Abbildung 18 UML Diagramm direkte Weiterleitung der Informationen	43
Abbildung 19 Ausgabe des Testcodes	45
Abbildung 20 Terminal Ausgabe während des Betriebes der Web-Socket Verbindung	52
Abbildung 21 Ausgabe am CEM Terminal beim Erhalten einer Nachricht des "grid_servers"	52

Abbildung 22 Terminal Ausgabe nach erfolgreichem Durchlauf	54
Abbildung 23 Roadmap EEBUS Server Spine Implementierung	57

5.2 Programmcode Verzeichnis

Programmcode 1 Grundfunktion Auslesen des Modus Registers für den PV-Überschussstrom L1	39
Programmcode 2 Funktion zum Schreiben und Überprüfen des PV-Überschuss-Stroms	42
Programmcode 3 Testfunktion Test.py mit allen Komponenten	44
Programmcode 4 Beispielcode JSON GCP Szenario 5	47
Programmcode 5 Funktion zum Erstellen einer JSON Nachricht	48
Programmcode 6 Grundkonstrukt "Web-Socket Client"	49
Programmcode 7 Ergänzungen der "while"-Schleife	51
Programmcode 8 Ergänzte Update Funktion	51
Programmcode 9 Ergänzung des Codes on_message	53

5.3 Internetquellen

https://github.com/snaptec/openWB	6
https://evcc.io/	6
<a href="https://<averbraucherzentrale.de/wissen/energie/preise-tarife-anbieterwechsel/stromclouds-spezialtarife-fuer-prosumer-haben-ihren-preis-56743">https://<averbraucherzentrale.de/wissen/energie/preise-tarife-anbieterwechsel/stromclouds-spezialtarife-fuer-prosumer-haben-ihren-preis-56743	8
https://www.adac.de/rund-ums-fahrzeug/tests/elektromobilitaet/schnellladen-langstrecke-ladekurven/	8
https://www.bewusst-haushalten.at/artikel/eco-programm/	9
https://www.co2online.de/energie-sparen/strom-sparen/strom-sparen-stromspartipps/strompreis/	10
https://www.sma.de/produkte/monitoring-control/sunny-home-manager-20.html	12
https://go-e.com/de-de/ueber-uns/presse/go-e-intensiviert-zusammenarbeit-mit-fronius	13
https://ger.sungrowpower.com/productDetail/905	13
https://www.eft-systems.de/de/B-BOX%20PREMIUM/product/Battery%20Box%20HVS-HVM/6	13
https://shop.go-e.co/go-eCharger-HOMEfix-11-kW	13
https://www.feldbusse.de/Modbus/TCP/Modbus/TCP_protokoll.shtml	16
https://www.simplymodbus.ca/TCPclient.htm	19
https://openwb.de/forum/viewtopic.php?f=11&t=3074&start=30	19
https://efahrer.chip.de/e-wissen/akku-richtig-laden-so-erhoehen-sie-die-lebensdauer-ihres-e-autos_10757	20
https://openwb.de/main/?page_id=31	21
https://www.pv-magazine.de/2017/04/21/fragen-antworten-zum-webinar-batterielebensdauer-einschaetzen-und-verlaengern-teil-3/	23
https://www.informatik-aktuell.de/betrieb/sicherheit/modbus-angriffe-im-lokalen-netzwerk.html	24

https://www.shodan.io/	24
https://www.EEBUS.org/media-downloads/	25
https://www.EEBUS.org/technology/#iLightbox[101d9b6470babe38dff]/0	26
https://www.baunormenlexikon.de/norm/din-en-806-5/a893cc21-7436-4e14-9158-8edb77ff24d9	28
https://www.youtube.com/watch?v=F3GlvWJd7fE&t=122s&ab_channel=ZentralverbandSanit%C3%A4rHeizungKlima-ZVSHK	31
https://pymodbus.readthedocs.io/en/latest/	37
https://github.com/snaptec/openWB/blob/master/packages/modules/common/modbus.py	37
https://pymodbus.readthedocs.io/en/latest/source/example/client_sync.html	38
https://www.youtube.com/watch?v=JNRFYtzQPfO&ab_channel=KarlZeilhofer	39
https://github.com/goecharger/go-eCharger-API-v1/blob/master/go-eCharger%20Modbus%20TCP%20API%20v1%20DE.md	41
https://Web-Socket-client.readthedocs.io/en/latest/	50
https://pypi.org/project/Web-Socket-client/	50
https://github.com/DerAndereAndi	57
https://github.com/evcc-io/EEBUS	57
https://github.com/DerAndereAndi/EEBUS-go-cem	57

5.4 Literaturquellen

Sungrow, Communication Protocol of Residential Hybrid InverterV1.0.21

Go-e, go-eCharger-API-Netzbetreiber

KfW, Merkblatt Ladestation für Elektroautos – Wohngebäude

MB-HWK, Netzspannung+Stecker_Europa und weltweit

Solaranlagen Ratgeber, Ratgeber-Photovoltaik

Dr. Harry Wirth, Fraunhofer ISE: Aktuelle Fakten zur Photovoltaik in Deutschland

Maren Fiege EEBUS Initiative e.V.: SHIP TS Spezifikation 1.0.1

Schwackenberg, Andreas: Monitoring of Grid Connection Point

SMA Solar Technology AG: Technische Information - SMA SMART HOME Energiemanagement mit elektrischen Verbrauchern über EEBUS

Torben Diekmann EEBUS Initiative e.V.: Optimization of Self-Consumption During EV Charging

Torben Diekmann; Andreas Schwackenberg: EEBUS Spine Protocol Specification

Torben Diekmann; Andreas Schwackenberg: EEBUS Ship Protocol Specification

Bundesnetzagentur, Bericht zur Spannungsqualität 2020

KEO Spezifikationen werden aufgrund der NDA (Non-Disclosure Agreement) nicht angegeben.